Міністерство освіти і науки, молоді та спорту України
Прикарпатський національний університет імені Василя Стефаника
кафедра радіофізики і електроніки

Методичні вказівки до виконання лабораторних робіт з курсу «ПРОГРАМУВАННЯ» для студентів спеціальності «Комп'ютерна інженерія»

Затверджено
на засіданні кафедри радіофізики і електроніки
протокол № 2 від 26 вересня 2011 року
та методичною радою фізико-технічного факультету
протокол № 2 від 25 жовтня 2011 року

Розробив:
доц. Запухляк Р.І.

м. Івано-Франківськ, 2011

Укладач: Запухляк Р.І.

Рецензенти: Дудка О.М., канд. пед. наук, доц. кафедри інформатики Прикарпатського національного університету імені Василя Стефаника.
Голота В.І., кандидат технічних наук, доц. кафедри радіофізики і електроніки Прикарпатського національного університету імені Василя Стефаника.

Подані методичні вказівки до виконання лабораторних робіт з курсу «Програмування» для студентів спеціальності «Комп'ютерна інженерія».

ЗМІСТ

РЕКОМЕНДАЦІЇ ДО ОФОРМЛЕННЯ ЗВІТУ ЛАБОРАТОРНОЇ РОБОТИ

Звіт до кожної лабораторної роботи оформляється на листках А4 формату у друкованому чи рукописному варіантах. Кожен звіт повинен містити:

1. Титульний лист формату, що наводиться нижче.
2. Починаючи із другої сторінки звіт містить: тему і мету роботи, текст завдання до роботи, вихідний код програми відповідного завдання, результати виводу програми, відповіді на контрольні запитання та висновки зроблені студентом в процесі виконання роботи.

Форма титульного листа:

Міністерство освіти і науки, молоді та спорту України
Прикарпатський національний університет імені Василя Стефаника
кафедра радіофізики і електроніки

Лабораторна робота № 1
ТЕМА: НАЙПРОСТІША ПРОГРАМА НА С++, ОСНОВИ СИНТАКСИСУ

Виконав:
Студент І-го курсу
Групи КІ-11
ПІП

м. Івано-Франківськ, 2011

Лабораторна робота № 1
ТЕМА: НАЙПРОСТІША ПРОГРАМА НА С++, ОСНОВИ СИНТАКСИСУ

МЕТА РОБОТИ: навчитися формувати вихідний код найпростіших програм на С++, оголошувати змінні і константи, виводити результати роботи програми. Ознайомитися із середовищем програмування Visual Studio C++.

Теоретичні відомості

Кожна програма (якщо вона складна) може бути розбита на декілька простих підпрограм. У мові С++ підпрограми називаються **функціями**, а програма на С++ являє собою набір із одної або більше функцій.

Будь-яка програма на С++ має містити функцію **main**. З неї починається виконання програми.

В кожній функцій ліва **({)** і права **(})** фігурні дужки позначають початок і кінець виконуючих виразів, ці вирази називаються **тілом функції**.

```
int main()
{
        тіло
}
```

Для написання програм використовуються як синтаксичні (граматичні), так і семантичні (логічні) правила.

Найпростіша програма на мові С++ виглядає так:

```
int main( )
{
return 0;
}
```

Ідентифікатори (identificators) використовуються в С++ для найменування різних об'єктів. Ідентифікатори складаються із букв (A-Z, a-z), цифр (0-9) і символів підкреслення (_), але при цьому повинні починатися тільки із букв або підкреслення.

Приклад:

Sum_sguare, J9.

Дані, що необхідні для роботи програми, зберігаються в пам'яті комп'ютера. В С++ є декілька типів даних. Спочатку розглянемо цілі типи (цілі числа), а потім – типи з плаваючою комою.

Цілі типи. Типи **char**, **chort**, **int** і **long** називаються цілими (integral types). Вони призначені для представлення цілих чисел різної довжини. В такому порядку як записані **char** – представляє найменше число, а **long** – найбільше. Найчастіше використовується тип **int**.

Тип **char** ще використовується для опису даних, що є символами. Символи поміщаються в одинарні лапки: **'8'** і **8** відрізняються, перший – це символ, а дугий – число.

Типи з плаваючою комою(крапкою). Типи з плаваючою крапкою (floating point types), або дійсні, є другою основною різновидністю простих типів С++: 18.0; 127.54; 0.57; 4. ; .8.

Типи з плаваючою крапкою мають також різні розміри. У порядку зростання є такі типи з плаваючою крапкою:

float, double і long double.

Числа з плаваючою крапкою можуть представлятися і у експоненціальному вигляді: 1.7453E-12; 1.53E11;7E20.

Останні два типи майже не використовуються, **float** забезпечує необхідну точність.

Ідентифікатори можна використовувати як для іменування констант так і змінних. В С++ кожний ідентифікатор може позначати тільки один елемент. Будь-який ідентифікатор перед використанням має бути описаним.

Змінна – це область пам'яті, яка позначена ідентифікатором, в якій зберігаються дані, значення яких змінюються.

Об'явити змінну означає задати як її ім'я, так і її тип. Наступний вираз об'являє змінну **empNum** типу **int**:

int empNum;

Існує можливість об'явити зразу декілька змінних у одному виразі:

int studentCount, **maxScore;**

Але краще задавати змінні окремими виразами.

Між константами і змінними можна виконувати різні арифметичні операції. В С++ існують такі основні арифметичні операції:

"+" – унарний плюс(додавання)

"-" – унарний мінус(віднімання)

"*" – множення

"/" – ділення з плаваючою крапкою(цілочисельне ділення)

"%" – остача від цілочисельного ділення.(6%2=0; 7%2=1).

В додаток до арифметичних операцій С++ включає операції інкременту (increment) або приросту і декременту (decrement) або від'ємного приросту.

++ – інкремент.

- – декремент.

Результат приросту полягає у збільшенні цілого або дійсного аргументу на одиницю.

num=8, то num++=9

Операції ++ і -- мають дві форми: префіксну ++num; і постфіксну num++; вони виконують одне і те ж, тому вибір запису залежить від програміста.

В С++ результат обчислень виводиться за допомогою змінної cout (сіаут) і оператора вставки (<<).

cout<< „hello";

Рядок **#include** є обов'язковим в програмі і обробляється не компілятором С++, а спеціальною програмою, що називається **препроцесором** (preprocesor). **Препроцесор** – це програма, яка діє як фільтр на етапі компіляції. Перед тим, як попасти на вхід компілятора, вихідна програма проходить через препроцесор.

Рядок, що починається символом дієзу (#) називається **директивою препроцесора** (preprocessor directive). Препроцесор розширює директиву **#include**, вставляючи вміст вказаного файлу безпосередньо у вихідну програму. Файли, що з'являються в директиві, як правило закінчуються на **.h** і називаються **файлами заголовків**. Файли заголовків містять оголошення констант, змінних і функцій, необхідних для роботи програми. Кутові дужки < > вказують препроцесору, що файл знаходиться в стандартному каталозі файлів заголовків (include directory).

Завдання для роботи

1. Написати програму обчислення квадрата і куба числа 27.

2. Написати програму обчислення середнього значення між точками замерзання і кипіння води у градусах Фаренгейта ($T_З$=32.0, $T_К$=212.0).
3. Написати програму, яка знаходила б розхід бензину (в км на літр) під час автомобільної поїздки. Дано кількість пального при заправці (в літрах), а також початкове і кінцеве значення автопробігу (67308.0 і 68750.0 км відповідно). Під час поїздки машину заправляли чотири рази (11.7, 14.3, 12.2 і 8.5 літрів бензину).

Написати вихідні коди програм, відкомпілювати і результати виконання здати викладачу.

Контрольні запитання

1. Що таке програмування?
2. Що таке комп’ютерна програма?
3. Етапи написання програм.
4. Життєвий цикл програми.
5. Що таке мова програмування?
6. Машинна мова або код?
7. Яку функцію виконує компілятор?
8. Функції програми. Тіло функції.
9. Ідентифікатори. Найменування об’єктів.
10. Цілі типи даних.
11. Типи з плаваючою крапкою.
12. Змінні. Об’явлення змінних.
13. Задання коментарів у програмі.
14. Арифметичні операції.
15. Вивід інформації.
16. Що таке пре процесор і для чого він призначений?
17. Директива програми. Файл заголовків.

Лабораторна робота №2
ТЕМА: АРИФМЕТИЧНІ ВИРАЗИ, ФОРМАТУВАННЯ ВИВОДУ

МЕТА РОБОТИ: навчитися синтаксично правильно використовувати арифметичні вирази при розв'язуванні задач та забезпечувати вивід результатів із необхідною точністю.

Теоретичні відомості

Між арифметичними операціями існує в С++ такий пріоритет: **( ), *, /, %, +, -**.

У випадку, коли декілька операцій з однаковим пріоритетом слідують одна за одною, їх порядок групування буде таким же, як і порядок запису цих операцій, тобто зліва на право.

Щоб зробити програми максимально зрозумілими і вільними від помилок використовується явне перетворення типів. Оператор перетворення типів С++ складається із імені типу, після якого в дужках записується перетворюючий вираз:

someFloat=float(3*someInt+2);

Типи даних можна змішувати у арифметичних виразах:

someInt*someFloat, 4.8+someInt-3.

Такі вирази є змішаного типу.

Кожний раз, коли ціле значення і значення з плаваючою крапкою є операндами арифметичної операції, відбувається неявне перетворення типів за наступною схемою:

1. Ціле число тимчасово перетворюється в число з плаваючою крапкою.
2. Виконується арифметична операція.
3. Результатом операції стає число з плаваючою крапкою

Форматувати вивід програми означає керувати його зовнішнім виглядом при відображенні на екрані або принтері.

Розглянемо, як можна керувати горизонтальними і вертикальними інтервалами при виводі.

Один із способів керування вертикальними інтервалами, це оператор endl. Якщо зустрічаються 2-а підряд оператори endl, то у виводі з'являється порожній рядок.

Компілятор визначає кінець виразу за крапкою з комою, а не за фізичним закінченням програмного рядка.

Маніпулятори використовуються тільки у виразах вводу і виводу.

Два інших маніпулятори, **setw** і **setprecision**. **setw** і **setprecision** потребують додатково включити в програму файл заголовків **iomanip.h.**

Маніпулятор **setw** дозволяє керувати кількістю позицій для виводу наступного за маніпулятором елемента даних. **setw** застосовується тільки для форматування чисел і рядків, але не даних типу **char**.

Маніпулятор **setprecision**:

cout<<setprecision(3)<<x;

вказує необхідну кількість знаків після десяткової крапки. Навідміну від **setw**, який діє тільки на елемент, що йде зразу за ним, **setprecision** залишається в силі для всіх наступних елементів вихідного потоку, поки його не буде змінено.

Завдання до лабораторної роботи

1. Написати програму обчислення вартості одного квадратного метра житлової площі будинку, якщо задані розміри будинку, число поверхів, площа не житлових приміщень і повна вартість крім землі. (Ширина будинку 30.0 м,

довжина 40.0 м, число поверхів 2.5 площа підсобних приміщень 825.0 м^2, продажна ціна будинку без вартості землі 150000.0 грн).

2. Нехай Вам потрібно в місті відвідати музей природознавства, картинну галерею, книжковий магазин, а потім сходити на концерт. Ці місця відзначенні на туристичній карті. Потрібно визначити відстань між туристичними об'єктами, а також шлях, який необхідно Вам пройти протягом дня. Результатами виводу мають бути: відстані між відвіданими місцями і повний шлях, заокруглений до десятих кілометра. Крім цього значення які використовуються при обчисленнях також мають бути надруковані для контролю правильності результатів. (Перша відстань 1.5 см, друга 2.3 см, третя 5.9 см, четверта 4.0 см, масштаб карти в 1 см 0.25 км).

Примітка: для заокруглення числа з плаваючою комою до найближчого цілого використовується такий вираз: int (floatValue+0.5). Для заокруглення до десятих спочатку необхідно домножити число на 10, округлити його результат до найближчого цілого, а потім розділити на 10. float(int(floatValue*10.0+0.5))/10.0.

Програмні коди, результати виконання програм та виконуючі файли продемонструвати викладачу у якості звіту до лабораторної роботи.

Контрольні запитання

1. Правила пріоритетів.
2. Неявне і явне перетворення типів.
3. Функції, які вертають значення.
4. Бібліотечні функції.
5. Вставка порожніх рядків.
6. Вставка пропусків всередині рядка.
7. Маніпулятори.

Лабораторна робота №3
ТЕМА: ВИКОРИСТАННЯ БІБЛІОТЕЧНИХ ФУНКЦІЙ, ВИКЛИК ФУНКЦІЇ

МЕТА РОБОТИ: навчитися застосовувати математичні бібліотечні функції при написанні програм на обрахунки.

Теоретичні відомості

Будь-який програмний комплекс С++ містить стандартну бібліотеку функцій, що виконують стандартні обчислення.

Використовувати бібліотечну функцію не важко. Перш за все треба помістити на початку програми директиву **#include** з означенням відповідного файлу заголовків.

Приклади бібліотечних функцій

Файл заголовків	Функція	Тип параметрів	Тип результату	Результат
<stdlib.h>	abs(i)	int	int	Модуль і
<math.h >	cos (x)	float	float	Косинус х (х в радіанах)
<math.h >	fabs (x)	float	float	Модуль числа х
<stdlib.h >	fabs(x)	long	long	Модуль числа х
<math.h >	pow(x, y)	float	float	Піднесення х в степінь у (якщо х=0.0, то у>0, якщо х≤0.0, то у має бути цілим числом
<math.h >	sin(x)	float	float	Синус х (х в радіанах)
<math.h >	sqrt(x)	float	float	Квадратний корінь із х (х>0.0)

Стандартна бібліотека С++ містить десятки функцій.

Завдання для роботи

Задача 1. Скласти і виконати програму, задавши вхідні дані самостійно.

1. Квіткова клумба має форму круга. Обчислити її периметр і площу за заданим радіусом.
2. Обчислити периметр і площу прямокутного трикутника за заданим катетом і гострим кутом.
3. Обчислити довжину кола і площу круга за заданим діаметром.
4. Ділянка лісу має форму рівнобічної трапеції. Обчислити її периметр і площу за заданими сторонами.
5. Ресторан закуповує щодня масло m_1 кг по 8,50 грн. за кілограм, сметану m_2 кг по 2,40 грн., вершки m_3 кг по 4,10 грн. Визначити суми, потрібні для купівлі окремих продуктів, і загальну суму.
6. Скільки секунд мають доба, тиждень, рік?
7. Обчислити кінетичну $E=mv^2/2$ та потенціальну $P=mgh$ енергії тіла заданої маси m, яке рухається на висоті h зі швидкістю v.
8. Ціни на два види товарів зросли на р відсотків. Вивести старі та нові ціни.
9. Обчислити площу поверхні $S=4\pi r^2$ та об'єм $V=4\pi r^3/3$ сфери за заданим радіусом r.
10. Швидкість світла 299792 км/с. Яку відстань долає світло за годину, добу?

11. Ввести врожайність трьох сортів пшениці (36, 40, 44 т/га) і площі трьох відповідних полів (га). Скільки зібрали пшениці з кожного поля і трьох полів разом.
12. Радіус Місяця 1740 км. Обчислити площу поверхні $S = 4\pi r^2$ та об'єм планети $V=4\pi r^3/3$.
13. Обчислити довжину гіпотенузи та площу прямокутного три кутника за заданими двома катетами.
14. Обчислити об'єм та площу бічної поверхні куба, якщо відоме ребро.
15. Ввести продуктивності роботи трьох труб, які наповнюють басейн, і час їхньої роботи. Скільки води набрано в басейн.
15. Яку площу і периметр матиме квадрат, описаний навколо круга заданої площі S.
16. Тіло падає з прискоренням g. Визначити пройдений тілом шлях $h= gt^2/2$ після першої та другої секунд падіння.
17. Обчислити периметр і площу прямокутного трикутника за заданими катетами.
18. Телефонні розмови з трьома населеними пунктами коштують c_1, c_2, c_3 коп/хв. Розмови тривали t_1, t_2, t_3 хв відповідно. Яку суму нарахує комп'ютер до оплати за кожну і всі роз мови?
19. Обчислити площу бічної поверхні $S=2\pi rh$ та об'єм $V=\pi r^2h$ діжки за заданою висотою h та радіусом основи r.
20. Квіткова клумба має форму квадрата. Обчислити її периметр і площу за заданою стороною.
21. Обчислити катет та площу прямокутного трикутника за заданими гіпотенузою та другим катетом.
22. Обчислити сторону та площу $S=d^2/2$ квадрата, якщо відома його діагональ d.
23. Обчислити площу бічної поверхні $S=\pi rl$ та об'єм $F=\pi r^2h/3$ конуса за заданою висотою h, твірною l та радіусом основи r.
24. Поїзд їхав t_1 год зі швидкістю v_1 км/год, t_2 год зі швидкість v_2 і t_3 год зі швидкістю v_3. Визначити пройдені шляхи з різною швидкістю і весь шлях.

Задача 2 (про трикутник). Трикутник задано координатами вершин A(0; 0), B(i; i-1) та C(-i; i+1), де i - номер варіанту.

Довідка. Для розв'язування типових задач про трикутник наведемо формули обчислення деяких величин:

Відстань d між точками: $d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$.

Координати середини відрізка: $x = \frac{(x_1+x_2)}{2}$, $y = \frac{(y_1+y_2)}{2}$.

Півпериметр трикутника: $p = \frac{(a+b+c)}{2}$.

Площа трикутника: $S = \sqrt{p(p-a)(p-b)(p-c)}$.

Висота трикутника: $h_a = \frac{2S}{a}$, $h_b = \frac{2S}{b}$, $h_c = \frac{2S}{c}$.

Бісектриса трикутника: $W_\alpha = \frac{2}{b+c}\sqrt{bcp(p-a)}$, $W_\beta = \frac{2}{a+c}\sqrt{acp(p-b)}$,

$W_\gamma = \frac{2}{a+b}\sqrt{abp(p-c)}$.

Радіус описаного кола: $R = \frac{abc}{4S}$.

Радіус вписаного кола: $r = \frac{S}{p}$.

a, b, c – сторони трикутника, α, β, γ – відповідні кути трикутника.

1. Обчислити висоту h_a та бісектрису W_c.
2. Обчислити медіану та і бісектрису W_b,.
3. Обчислити бісектрису W_a та радіус вписаного кола r.

4. Обчислити висоту h_a і медіану m_b.
5. Обчислити медіану m_b та бісектрису W_c.
6. Обчислити бісектрису W_a і радіус описаного кола R.
7. Обчислити висоту h_b та бісектрису W_a.
8. Обчислити висоту h_b і медіану m_c.
9. Обчислити висоту h_a та радіус вписаного кола r.
10. Обчислити медіану m_c і бісектрису W_a.
11. Обчислити висоту h_b, та бісектрису W_c.
12. Обчислити медіану m_c і радіус вписаного кола r.
13. Обчислити висоту h_b та медіану m_a.
14. Обчислити медіану m_a і радіус описаного кола R.
15. Обчислити медіану m_a та бісектрису W_c.
16. Обчислити висоту h_c і бісектрису W_a.
17. Обчислити медіану m_b та радіус вписаного кола r.
18. Обчислити висоту h_c і медіану m_a.
19. Обчислити медіану m_b та бісектрису W_a.
20. Обчислити медіану m_c і радіус описаного кола R.
21. Обчислити висоту h_b, та бісектрису W_b.
22. Обчислити висоту h_b і медіану m_b.
23. Обчислити висоту h_a та радіус описаного кола R.
24. Обчислити висоту h_a і бісектрису W_b.
25. Обчислити висоту h_a та медіану m_c.

Контрольні запитання

1. Як розширити функціональні можливості програми?
2. Який файл заголовків використовується для включення в програму математичних функцій?
3. Які файли заголовків ви знаєте?
4. Чи можна функції фалу заголовків замінити своїми?

Лабораторна робота № 4
ТЕМА: ВВІД ДАНИХ У ПРОГРАМУ, ІНТЕРАКТИВНІ І НЕІНТЕРАКТИВНІ ПРОГРАМИ

МЕТА РОБОТИ: навчитися отримувати вхідні дані для програми із клавіатури та писати програми з інтерактивним вводом.

Теоретичні відомості

Процес поміщення значень із стороннього набору даних в програмні змінні називається **вводом** (input). Дані можуть поступати в програму з пристрою вводу або із файлу, що зберігається на зовнішньому запам'ятовуючому пристрої.

Файл заголовків **iostream.h** містить визначення двох типів даних: **istream** і **ostream.** Ці типи представляють собою відповідно вхідні і вихідні потоки.

```
istream cin;
ostream cout;
```

cin пов'язаний із стандартним пристроєм вводу – клавіатурою, а **cout** – зі стандартним пристроєм виводу – як правило монітором.

Як уже відомо, дані можна виводити в **cout** за допомогою операції вставки "<<". Подібним чином дані із **cin** можна вводити за допомогою операції добування: ">>". Так, оператор **cout<<someInt;** пересилає дані із змінної **someInt** у вихідний потік. А оператор **cin>>someInt;** поміщає дані із вихідного потоку у змінну **someInt** .

При вводі даних із клавіатури необхідно перевіряти кожне значення яке вводиться на відповідність типу змінної.

Коли необхідно зчитувати всі символьні дані включаючи пробіли, використовується функція **get()**, яка читає кожний знак. Звернення до цієї функції виглядає так:

```
cin.get(someChar);
```

Виклик функції є окремим оператором, а не частиною більшого виразу.

Функція **ignore()** застосовується для того, щоб ігнорувати, тобто зчитувати і відкидати символи вхідного потоку. Вона містить два параметри, а звернення до неї виглядає так:

```
cin.ignore(200,'\n');
```

Перший параметр функції є цілим виразом, а другий – символьною величиною. У приведеному прикладі звернення до функції повідомляє комп'ютеру, що треба пропустити двісті почергових символів із вхідного потоку або ігнорувати всі символи до тих пір, поки не буде зчитаний символ нового рядка – в залежності від того, яка із цих подій відбулася першою.

Інтерактивною (interactive) називається програма, за допомогою якої користувач обмінюється інформацією безпосередньо з комп'ютером. Для того, щоб інтерактивна програма отримала дані, необхідно спочатку надрукувати так звану вхідну підказку (input prompt) – повідомлення, що повідомляє користувачу, які дані необхідно ввести. Програма також повинна роздрукувати всі величини, які отримані ззовні, для того щоб користувач міг переконатися в тому, що вони введені правильно.

Наведемо приклад програми, що використовує підказки:

```
cout<<"Введіть код товару:"<<endl; //Підказка
cin>>partNumber;
```

Багато програм пишуться з використанням неінтерактивного вводу-виводу. Типовим прикладом неінтерактивного вводу-виводу у великих комп'ютерних

системах – це пакетна розробка. При пакетній розробці користувач не взаємодіє з комп'ютером під час роботи програми.

Коли програма повинна зчитувати багато даних, звичайним прийомом є попередня підготовка цих даних з їх збереженням у деякому файлі. Це дає користувачу можливість повернутися до даних і внести необхідні поправки і зміни в файл до запуску програми.

Програми без діалогового вводу-виводу не друкують підказки для вводу даних. Але корисно використовувати еходрук всіх введених даних. Він дає користувачу можливість впевнитись у правильності вводу. Основна різниця між інтерактивними і неінтерактивними програмами полягає у вимогах до вводу і виводу. Неінтерактивні програми є більш вимогливими.

Завдання для роботи

1. Написати програму, яка знаходила б розхід бензину (в км на літр) під час автомобільної поїздки. Дані взяти із клавіатури у вигляді інтерактивного вводу із підказками: 11.7 14.3 12.2 8.5 67308.0 68750.5 (де перші чотири числа це заправка автомобіля у літрах, два інші це початкове і кінцеве значення автопробігу відповідно у км). Результат має бути виведений з подвійною точністю, а також із відповідними підписами.
2. Знайти середньовагове значення трьох екзаменаційних оцінок студента. Для кожного екзамену або заліку дана пара чисел: оцінка (ціле число) і її ваговий коефіцієнт (дійсне число), причому кожна така пара вводиться у програму у вигляді: 90 0.30 85 0.25 78 0.45. Для отримання результату треба здійснити інтерактивний ввід цих даних. Результатами виводу мають бути вхідні дані із відповідними підписами, середньо вагове значення оцінки із відповідним підписом. Всі дійсні числа мають виводитися із подвійною точністю.

Вихідні коди програм, результати виводу та виконуючі файли продемонструвати викладачу у якості звіту.

Контрольні запитання

1. Потоки вводу і операція добування.
2. Що таке маркер зчитування і символ нового рядка?
3. Для чого призначені функції get() і ignore()?
4. Що таке інтерактивний і не інтерактивний ввід-вивід?

Лабораторна робота № 5
ТЕМА: ФАЙЛОВИЙ ВВІД І ВИВІД У ПРОГРАМАХ НА С++

МЕТА РОБОТИ: навчитися використовувати файловий ввід-вивід у програмах.

Теоретичні відомості

Програми можуть зчитувати дані із дискового файлу так само, як вони зчитують їх з клавіатури.

Для того, щоб програма могла використовувати файловий ввід-вивід, необхідно зробити наступне:

1. Повідомити препроцесору, щоб він включив у програму файл заголовків **fstream.h**.

2. Оголосити відповідні файлові потокові змінні типу **ifstream** і **ofstream**:

ifstream infile;

ofstream outfile;

3. Оголосити файли, які будуть використовуватись та відкрити їх до читання або запису за допомогою функції **open**.

infile.open(“indata.dat”);

outfile.open(“outdata.dat”);

4. Отримати дані із файлу:

infile>>a>>b;

5. Опрацювати вхідні дані та вивести результати у файл:

outfile<<”Результат: ”<<a+b<<endl;

Завдання для роботи

1. Написати програму, яка знаходила б розхід бензину (в км на літр) під час автомобільної поїздки. Дані взяти із файла infile.dat і записати у файл outfile.dat. Дані у файлі infile.dat мають вигляд: 11.7 14.3 12.2 8.5 67308.0 68750.5 (де перші чотири числа це заправка автомобіля у літрах, два інші це початкове і кінцеве значення автопробігу відповідно у км). Результат має бути виведений з подвійною точністю, а також мають бути виведені із відповідними підписами вихідні дані.
2. Знайти середньовагове значення трьох екзаменаційних оцінок студента. Для кожного екзамену або заліку дана пара чисел: оцінка (ціле число) і її ваговий коефіцієнт (дійсне число), причому кожна така пара записана у окремому рядку. Дані зберігаються у файлі scoreFile.dat у вигляді: 90 0.30 85 0.25 78 0.45. Для отримання результату треба здійснити інтерактивний ввід цих даних у файл. Результатами виводу мають бути вхідні дані із відповідними підписами, середньо вагове значення оцінки із відповідним підписом. Всі дійсні числа мають виводитися із подвійною точністю. Результати вивести у файл і на екран.

Вихідні коди програм, результати виводу та виконуючі файли продемонструвати викладачу у якості звіту.

Контрольні запитання

1. Сформулювати алгоритм використання файлів у програмах.
2. Коли варто використовувати файловий ввід-вивід?
3. Який файл заголовків використовується для файлового вводу-виводу?
4. Які є способи використання файлового вводу-виводу?

Лабораторна робота №6
ТЕМА: КЕРУЮЧИЙ ОПЕРАТОР IF

МЕТА РОБОТИ: вивчити синтаксис умовного оператора if та особливості його застосування.

Теоретичні відомості

Порядок, у якому вирази виконуються в програмі, називається **потоком керування** (flow of control). Потік керування як правило є послідовним. У місцях, де потрібно змінити напрям потоку керування, використовуються **керуючі структури** (control structures). Це спеціальні оператори, які передають керування виразу, що не є наступним за порядком. Керуюча структура вибору використовується для вибору між взаємовиключаючими діями. Для цього формулюється деяке твердження, яке або істинне, або ні. Якщо воно істинне, то комп'ютер виконує один вираз, якщо ні, то інший. Оператор if, або умовний (relational) оператор, дозволяє маніпулювати потоком керування. В С++ умовний оператор має дві форми: if-then-else і if-then. Розглянемо його представлення у вигляді if-then-else. Синтаксис:

```
if (деяка умова істинна)
then виконати певну дію;
else виконати іншу дію;
```

then можна опускати для полегшення читання коду

```
if (hours<=40.0)
        pay=rate*hours;
else
        pay=rate*(40.0+(hours-40.0)*1.5);
cout<<pay;
```

Помістивши послідовність операторів у фігурні дужки {}, ми перетворюємо ці оператори в єдиний вираз:

```
if (divisor !=0)
{
        result=dividend/divisor;
        cout<<"Істина"<<endl;
}
else
{
        cout<<"Помилка"<<endl;
        result=9999;
}
```

Інколи можна стикнутися з ситуацією, яка описується так: „якщо деяка умова істинна, необхідно виконати певну дію, в іншому випадку ніяких дій застосовувати не потрібно".

```
if(a<=b)
c=20;
else
;
```

А краще зовсім опустити else, в результаті отримаємо умовний оператор у формі if-then.

```
if (age<18)
   cout<<"Не може бути виборцем";
```

```
cout<<”Виборець”<<endl;
```

Умовний оператор if-then так як і if-then-else може містити блок. Наприклад: „Відніміть значення у рядку 23 від значення у рядку 17 і запишіть результат у рядок 24; якщо результат від’ємний, запишіть нуль і відмітьте пункт 24А”.

```
result=line17-line23
if (result<0.0)
{
      cout<<”пункт 24А”<<endl;
      result= 0.0;
}
line24=result;
```

В С++ не існує обмежень на тип виразів, що входять в умовний оператор. Отже, всередині одного оператора if може знаходитись інший оператор if. При розміщенні if всередині if створюється так звана вкладена керуюча структура (nested control structure).

Коли умовні оператори вкладені, легко заплутатися до якого if належить те чи інше else. Існує просте правило якого дотримується компілятор С++, при відсутності фігурних дужок, else завжди відноситься до найближчого if, що немає парного else. Нам хотілося б, щоб else відносилося до зовнішнього, а не до внутрішнього if, але це не так, бо форматування не впливає на приналежність.

```
if(average>=60.0)
      if (average<70.0)
        cout<<”Добре”;
else
      cout<<”Не здано”;
```

Треба записати так:

```
if (average>=60.0)
{
      if (average<70.0)
            cout<<”Добре здано”;
}
else
      cout<<”Не здано”;
```

Фігурні дужки вказують, що внутрішній умовний оператор закінчений, тому **else** ставиться у відповідність зовнішньому if.

С++ забезпечує спосіб перевірки, чи знаходиться потік у стані відмови, чи ні. Для цього потокова змінна просто підставляється у логічний вираз, так ніби це є булева змінна:

```
if (cin)
...
if (! inFile)
...
```

Це називається перевіркою стану потоку (testing the state of a stream). Результатом такої перевірки є або ненульове значення, що означає, що остання операція вводу-виводу з даним потоком була успішною, або нуль, що відповідає помилці вводу-виводу.

Треба пам’ятати, що потік, перейшовши у стан відмови, в ньому і залишається. Покажемо, як перевірити, чи був вхідний файл успішно відкритий.

```
#include <iostream.h>
#include <fstream.h> // Для файлового вводу-виводу
    int main()
{
    int neight;
    int width;
    ifstream inFile;
    inFile.open(“mydata.dat”); // Спроба відкрити файл для читання
if (!inFile) // Чи відкритий файл?
{
    cout<<”Файл не відкритий”; // Вивід повідомлення
    return 1; // і завершаємо програму
}
inFile>>height>>width; // Зчитуємо дані
    .
    .
    .
return 0;
}
```

Кожен раз, коли відкривається файл даних для читання, треба перевіряти стан потоку до того, як приступити до роботи з цим файлом.

Завдання для роботи

Ввести довільне значення x та обчислити значення функції

$$y=\begin{cases} f_{i+2}(\varphi), & \text{якщо } |x|<10, \\ f_{i+3}(\omega), & \text{якщо } |x|\geq 10, \end{cases}$$

де $\varphi=\operatorname{tg}(x+a)-\log_i|b+7|$, $\omega=c\sqrt[5]{x^2+de^{1,3}}$, i – номер варіанта. Скласти дві програми, використовуючи коротку команду розгалуження if.

Вхідні дані (x,a,b,c,d) ввести з клавіатури довільні. Результати обчислень вивести на екран і у файл.

№ п/п	Функція $f_n(x)$
1	9,2cos2x-|sinx/1,1|
2	12,4sin|x/2,1|-8,3cos1,2x
3	|cosx/2,7|+9,lsin(1,2x+1)
4	|sinx/3,12+cosx^2|-8,3sin3x
5	cos|2x|/1,12-cos(3x-2)+6,15
6	sinxcos^{2}xsin(x+1,4)/0,85+7,14
7	|sin(2x-1,5)+3sin4x|+2,38

8	$\cos x^2 \sin(2x-1)+4,29$
9	$\cos(x^{2,4}+1)-\lvert\sin 2x-5,76\rvert$
10	$\sin x-\cos x^3 \sin(x^2-4,2)+4,27$
11	$\lvert\sin 12x\cos\lvert 2x\rvert/3\rvert+4,21$
12	$\cos x^3/2,1+\cos x^2/1,1-8,3\sin(3x+1)$
13	$\sin x^2 \cos x^3-\sin x+5,2$
14	$2\sin x\sin(2x-1,5)\cos(2x+1,5)-6$
15	$\lvert\cos x^2-0,51\rvert\sin(3x-4)-4,44$
16	$\cos 2,lx\sin\lvert x\rvert/0,15-5,8$
17	$\lvert\cos 2x^3+2\sin(x/1,2-3,4)\rvert+10,51\cos\lvert 3x\rvert$
18	$\lvert\sin(x^2/1,5-2)\rvert+11,73\cos(1,6x-1)$
19	$13,4\cos\lvert x\rvert\sin(x^2-2,25)$
20	$\lvert\cos(x^2-3,8)\rvert/4,5-9,7\sin(x-3,1)$
21	$13,4\sin(-1,26x)\cos\lvert x/7,5\rvert$
22	$2\sin\lvert 2x\rvert\cos 2x-11,6\sin(x/0,4-1)$
23	$\sin\lvert x\rvert/0,1+9,4\sin(3x-2,5)$
24	$10,8\lvert\cos(x^2/1,13)\rvert\sin(x+1,4)$
25	$11,2\cos(2x-1)+\lvert\sin 1,5x\rvert/1,7$

Вихідний код та результати роботи програми продемонструвати викладачу та захистити її.

Контрольні запитання

1. Що таке потік керування?
2. Для чого призначені керуючі структури?
3. Які форми має умовний оператор if?
4. Які керуючі структури називаються вкладеними?
5. Як виконати перевірку стану потоку вводу-виводу?

Лабораторна робота №7
ТЕМА: ВИКОРИСТАННЯ ЛОГІЧНИХ ОПЕРАЦІЙ ДЛЯ ОРГАНІЗАЦІЇ РОЗГАЛУЖЕННЯ В ПРОГРАМІ

МЕТА РОБОТИ: навчитися використовувати власний логічний тип для перевірки введених даних та використовувати у програмах логічні функції і операції порівняння.

Теоретичні відомості

Щоб задати питання на мові С++, необхідно сформулювати твердження, яке є істинним або ні. У мові С++ твердження мають форму логічних (logical) виразів, що називаються також **бульовими** (Boolean). Такі вирази складаються із логічних значень і операцій. Ось декілька прикладів логічних виразів:

а) бульова змінна або константа;

б) два вирази, що з'єднані оператором порівняння;

в) два логічних вирази, що з'єднані логічним оператором.

Розглянемо ці приклади окремо.

У мові С++ величина 0 представляє значення **"хиба" (false)**, а будь-яка невід'ємна величина представляє значення **"істина" (true)**. Тому як правило використовують тип **int** для представлення логічних даних:

```
int dataOK;
    ...
dataOK=1; // зберігає істину (true) в dataOK
    ...
dataOK=0; // зберігає хибу (false) в dataOK
```

В той же час багато програмістів надають перевагу оголошувати власний логічний тип даних за допомогою typedef. Це оголошення дозволяє присвоїти нове ім'я існуючому типу даних:

```
typedef int Boolean;
```

Тут компілятор підставляє слово int замість кожного входження слова Boolean у програми. Щоб закінчити побудову власного логічного типу, треба оголосити дві іменовані константи: true і false.

```
typedef int Boolean;
const Boolean TRUE=1;
const Boolean FALSE=0;
Boolean dataOK;
...
dataOK=TRUE;
...
dataOK=FALSE;
```

В дійсності новий тип даних не створений. Boolean – це просто друге позначення для типу int, TRUE і FALSE – синоніми значень 1 і 0. Додаткові позначення дозволяють розділити логічні і цілі дані при розробці програми. По-друге, при читанні коду, стає зрозуміло зразу, що ідентифікатори Boolean, TRUE, FALSE відноситься до логічних величин.

Одним із способів присвоєння значень логічним змінним є застосування оператора присвоєння:

```
itemFound=TRUE;
```

Можна також присвоїти значення логічній змінній, прирівнявши її результату порівняння 2-ох виразів, що з'єднані операцією порівняння.

Операції порівняння :

= = – рівне

! = – нерівне

> – більше

< – менше

>= – більше або рівне

<= – менше або рівне

У мові С++ результатом порівняння є значення цілого типу, причому 1 означає TRUE, а 0 – FALSE. Наприклад, якщо x=5, а y=10, то кожний із виразів буде мати значення 1 (TRUE)

x = = 5	y>x	y>=x
x!=y	x<y	x<=y

Якщо x містить символи 'M', а y – 'R', то записані співвідношення будуть також справедливі, так як операція порівняння **<** застосовна до букв і означає бути попередньою у алфавіті.

Так як у випадку арифметичних операцій при порівнянні необхідно застосовувати явне перетворення типів. Для типу char, необхідно слідкувати, щоб порівнювати з таким самим типом.

Умовні оператори можуть використовуватися не тільки для порівняння змінних або констант, але і для порівняння значень арифметичних виразів:

x+3<=y*10	1 (true)	x-12, y-2
x+3<=y*10	0 (false)	x-20, y-2

Треба не плутати операцію присвоєння (=) з операцією порівняння (= =), бо вони виконують різні дії.

В С++ для позначення логічних операцій існують спеціальні символи: && (логічне множення і), | | (для логічного додавання "або"), і ! (для логічного заперечення "не"). За допомогою логічних операцій можна створювати значно складніші вирази. Наприклад, якщо потрібно визначити, чи перевищує підсумкова оцінка значення 90, чи проміжну оцінку, більшу за 70, тоді вираз буде такий:

finalScore>90 && midtermScore>70

Щоб результат операції "і" був рівний TRUE, необхідно, щоб обидва її операнди були істинними. Якщо будь-який із операндів має значення FALSE, то загальний результат також буде хибним.

Операція "або" (| |) використовує також два значення. Якщо одне із них або обидва істинні, то загальний результат істинний "TRUE". Якщо обидва хибні, то результат хибний "FALSE".

Логічне заперечення (!) надає протилежність значенню операнда. Якщо grade= ='A' рівне FALSE, то !(grade= ='A') рівне TRUE. Твердження !(hours>40) є еквівалентом hours<=40.

Інші пари еквівалентних виразів:

Вираз	Еквівалент
!(a= =b)	a!=b
!(a= =b | | a= =c)	a!=b && a!=c
!(a= =b && c>d)	a!=b | | c<=d

Треба мати на увазі, що для того, щоб отримати вираз із лівої колонки, потрібно взяти відповідний вираз із правої колонки і записати перед ним знак операції ! і

замінити всі логічні або умовні операції на протилежні (= = замість !=, | | замість && і т.д.)

Нехай маємо дві логічні змінні x і y.

Значення x	Значення y	Значення операції над x і y
Операції логічного множення && x і y		
true	true	true
true	false	false
false	false	false
false	true	false
Операції логічного додавання | | x і y		
true	true	true
true	false	true
false	true	true
false	false	false
Оператор логічного заперечення !		
true	false	
false	true	

У мові C++ використовується швидке або умовне обчислення логічних виразів.

Швидке обчислення – обчислення логічного виразу зліва направо, при якому знаходження підвиразів припиняється, як тільки кінцеве значення виразу стає відомим.

Приклад:

i= =1 && j>2

Якщо i=95, тоді перший підвираз хибний і немає змісту обчислювати другий підвираз. Відповідно комп'ютер припиняє обчислення цього виразу і отримує його кінцевий результат – false.

Для операцій "або" оцінка підвиразів зліва направо зупиняється, як тільки встановлено підвираз, що має значення true.

Тепер увівши логічні операції покажемо їх пріоритет у сукупності із арифметичними виразами:

найвищий пріоритет

```
!
/        %
+        -
<        <=      >=      >
= =      !=
&&
||
=
```

вирази записані в одному рядку мають однаковий пріоритет. Якщо в одному виразі зустрічаються декілька операцій з однаковим пріоритетом, то вони виконуються в порядку слідування, тобто зліва направо.

Операції порівняння можуть застосовуватися до будь-якого із трьох основних типів даних: int, float, char.

Для чисел з плаваючою крапкою для порівняння використовується різниця між двома числами і різниця перевіряється чи вона не перевищує окрему деяку максимально допустиму похибку:

fabs(r-s)<0.00001

де fabs-функція стандартної бібліотеки, що обчислює модуль дійсної величини. r і s змінні типу float. Якщо різниця менша 0,00001, то значення двох чисел достатньо близьке, щоб вважати їх рівними.

Завдання для роботи

Написати програму, яка б обчислювала середнє арифметичне значення чотирьох екзаменаційних оцінок і друкувала ідентифікаційний номер студента (номер залікової книжки), середнє арифметичне значення і результат (чи зданий екзамен чи ні). Для успішного складання екзамену в цілому середнє арифметичне значення оцінок має бути не менше 3 балів. Якщо при цьому середня оцінка менше 4 балів, то програма має повідомити, що екзамен складено задовільно, якщо більше або рівне 4 але менше 5, то повідомити, що складено добре, якщо рівне 5, то повідомити, що складено відмінно. В іншому випадку повідомити, що екзамен не складено.

Вхідні дані: Ідентифікаційний номер студента (номер залікової типу long) і чотири екзаменаційні оцінки (типу int).

Вихідні дані: Підказка для вводу, ідентифікаційний номер студента, середня оцінка, результат про здачу екзамену чи ні, уточнення якщо екзамен зданий задовільно і повідомлення про помилку (якщо якась екзаменаційна оцінка від'ємна). Для перевірки правильності введених оцінок використати власний логічний тип даних.

Контрольні запитання

1. Як створити власний тип користувача?
2. Назвати основні операції порівняння.
3. Які логічні операції ви знаєте? Сформулювати їх таблиці істинності.
4. Сформулюйте пріоритет логічних операцій у сукупності із арифметичними.
5. Як порівняти два дійсних числа?

Лабораторна робота № 8
ТЕМА: ЦИКЛИ, ЩО КЕРУЮТЬСЯ ЛІЧИЛЬНИКОМ

МЕТА РОБОТИ: навчитися застосовувати цикли while при організації операцій, що повторюються.

Теоретичні відомості

Оператор while, як оператор if, перевіряє виконання деякої умови.

Приклад.

```
while(inputVal !=25)
cin>>inputVal;
```

Вираз while являє собою циклічну керуючу структуру. Оператор який повторюється кожний раз, коли виконується той чи інший цикл, називається тілом (body). У прикладі тіло циклу – це оператор, що зчитує значення inputVal. Конструкція while має наступний зміст: ''Якщо значення умови відмінне від нуля (true), то виконати тіло циклу''.

Тіло циклу виконується за декілька кроків.

1. Той момент, коли потік керування передається першому оператору всередині тіла циклу, називається **точкою входу** (loop entry) даного циклу.
2. Наступний прохід через цикл здійснюється кожний раз, коли виконується його тіло; такий прохід називається **ітерацією** (iteration).
3. Перед кожною ітерацією керування передається на перевірку умови (loop test) на початку циклу.
4. Коли остання ітерація завершена і керування перейшло до оператора, наступному безпосередньо після циклу, говорять, що програма вийшла із циклу. Умова, що приводить до закінчення циклу, називається умовою **завершення циклу** (termination condition). Завершення циклу **while** відбувається при рівності нулю (false) умови в точці його перевірки.

Основні типи циклічних процесів: цикли, що керуються лічильником (count-controlled), які повторюються вказане число раз, і цикли, що керуються подією (event-controlled), які повторюються до тих пір, поки в середині циклу не відбудеться певна подія.

Цикл, що керується лічильником при аналізі умови використовує змінну циклу. Перед входом у цикл, що керується лічильником, необхідно ініціалізувати змінну керування циклу, тобто присвоїти їй певне початкове значення, а потім це значення протестувати. Потім при кожній наступній ітерації змінна циклу отримує приріст.

```
loopCount=1;            // Ініціалізація
While (loopCount<=10)       // Перевірка умови.
{
…                       // Повторюючи дії.
loopCount++;            // Приріст лічильника.
}
```

Якщо програма виконується набагато довше очікуваного часу, то є імовірність, що в програмі присутній нескінченний цикл.

Завдання для роботи

Побудувати таблицю відповідностей між мірами. Початкове значення міри, крок зміни цього значення та кількість рядків у таблиці (10-15) задати самостійно у режимі діалогу. Оформити таблицю якнайкраще, використовуючи формати виведення.

1. 1 унція=28.353495 г=142 карати;
2. 1 драхм=1.77185 г=0.06249 унцій;
3. 1 карат = 0.2 г=2.9412 гран;
4. 1 гран=0.068 г=0.038378 драхм;
5. 1 пайп=54.18 пек=477.33 л;
6. 1 галон (брит.)=1.2 галон (США)=4.546 л;
7. 1 галон (США)=0.0347 сак=3.785 л;
8. 1 чарка=0.0568 л=0.00012 пайпа;
9. 1 квартет=291 л=5123.24 чарок;
10. 1 страйк=72.73 л=1280.46 чарок;
11. 1 челдрон=1.309 л=0.149 пека;
12. 1 сак=109 л=1.499 страйка;

Контрольні запитання

1. Назвати основні типи циклів.
2. Синтаксис циклу while.
3. Що таке тіло циклу?
4. Що таке ітерація?

Лабораторна робота № 9
ТЕМА: ЦИКЛИ, ЩО КЕРУЮТЬСЯ ПОДІЄЮ

МЕТА РОБОТИ: вивчити тип роботи циклів, що керуються міткою.

Теоретичні відомості

Цикл що керується подією, має 2-а види: цикл, що керується сигнальною міткою (sentinel-controlled); і цикл, що керується ознакою закінчення файлу (end-of-file-controlled).

Цикли часто застосовуються для зчитування і обробки великих масивів даних. При кожному наступному зверненні до тіла циклу читається і аналізується нова порція даних. Щоб повідомити програмі, що даних для обробки більше немає, як правило використовується спеціальна величина, яка називається сигнальною (sentinel) або кінцевою (troiler) міткою. Виконання циклу продовжується, поки зчитується значення даних не рівній цій мітці. Іншими словами, читання мітки – це подія, яка керує процесом виконання циклу. Наприклад, якщо програма зчитує календарні дати, можна було б використовувати 31 лютого в якості сигнальної мітки:

```
cin>>month>>day;                // Початкове зчитування дати.
while (!(month= =2 && day= = 31))
{
…                                          // Обробка дати.
cin>>month>>day;                // Зчитування наступної дати.
}
```

Цей фрагмент зчитує перший набір даних до входу в цикл і якщо дані не співпадають з міткою, то починається їх обробка. В кінці циклу читається наступний набір даних, після чого керування вертається до початку циклу і т. д. Треба бути уважним при організації циклів програми, щоб не зациклити програму.

Завдання для роботи

Написати програму, яка б обчислювала кількість операцій „!=” у файлі, що містить текст програми на С++. Для цього необхідно створити файл myfile.dat з таким вмістом:

```
abc!=
def!=
abc!
def========
abc!!!!!!!!=!=!=abc !=
```

Результатом виводу має бути ціле число кількості знаків „!=”.

Контрольні запитання

1. Коротко поясніть різницю між циклом і розгалуженням.
2. Назвати чотири частини циклу, що керується лічильником.
3. Чи потрібно застосовувати початкове зчитування в циклі, що керується ознакою закінчення файлу?
4. В чому різниця між лічбою і сумуванням в циклі?
5. В чому різниця між змінною керування циклу і лічильником подій?
6. Написати перший рядок циклу while, який виконується до тих пір, поки змінна done не стане рівна TRUE.

Лабораторна робота № 10
ТЕМА: ЦИКЛИ, ЩО КЕРУЮТЬСЯ ОЗНАКОЮ ЗАКІНЧЕННЯ ФАЙЛУ

МЕТА РОБОТИ: продовжити вивчення циклів, що керуються сигнальною міткою.

Теоретичні відомості

Розглянемо цикл, що керується ознакою закінчення файлу. Після того, як програма прочитує останній елемент даних із вхідного файлу, досягається закінчення файлу (end-of-file). В цей момент потік знаходиться в робочому стані.

Але якщо попробувати ввести ще одне значення, потік перейде у стан відмови. Цю обставину можна повернути у свою користь: якщо число елементів даних, що зчитується циклом, попередньо невідоме, то в якості "мітки" зручно використовувати збій вхідного потоку.

Припустимо, що потрібно зчитати файл цілих чисел, а inData – це ім’я файлового потоку в програмі.

```
inData>>intVal;                 // Зчитуємо перше число.
while (inDate)                  // До тих пір поки ввід йде успішно,
{
        cout<<intVal<<endl;     // Відображаємо введене значення
        inDate>>intVal;                 // і зчитуємо наступне число.
}
```

Через спробу прочитати дані за межами файлу потік inDate переходить у стан відмови. При черговій перевірці виявляється, що значення умови рівне нулю (false), тому відбувається вихід із циклу.

Для того, щоб цикл був корисним, його тіло повинно виконувати певні дії. Є три основні задачі, які виконують циклічні обчислювальні процеси: лічба, сумування і облік попереднього значення.

Лічба.

Наведемо приклад коду, який з пристрою вводу отримує окремі символи і перераховує їх до тих пір, поки серед них не зустрінеться крапка:

```
count=0;                        // Ініціалізація лічильника.
cin.get(inChar);                // Отримуємо перший  символ.
while (inChar !='.')
{
count ++ ;                      // Приріст лічильника.
cin.get(inChar);                // Отримуємо наступний символ.
}
```

Після завершення циклу, count містить значення, яке менше загального числа прочитаних символів на 1. Треба відмітити, що тіло циклу взагалі не виконується, якщо крапка перший символ. Лічильник ітерацій – це змінна-лічильник, значення якої отримує одиничний приріст при кожній ітерації циклу.

Сумування

Друге типове застосування циклів полягає у сумуванні ряду значень:

```
sum=0;                          // Ініціалізація суми.
count=1;
while (count<=10)
{
cin>>number;            // Ввести значення .
```

```
sum=sum+number;    // Додати значення до суми.
count ++;
}
```

Після виконання коду sum містить десять введених чисел, лічильник count дорівнює 11 , а останнє введене число зберігається у змінній number.

Інколи потрібно від слідкувати попереднє значення будь-якої змінної. Припустимо, що потрібно знайти число операцій "!=" у файлі, що містить текст програми на C ++. Це можна зробити, підрахувавши всі появи знаку оклику (!), за яким слідує знак дорівнює (=). Спосіб підрахунку полягає в тому, щоб зчитати вхідний файл посимвольно і відслідковувати значення двох останніх введених символів. При кожній ітерації поточне значення стає попереднім, а нове поточне значення зчитується із файлу. Коли досягається закінчення файлу, цикл закінчується.

Завдання для роботи

Необхідно написати програму, яка б обчислювала середній дохід мужчин і жінок.

Вхідні дані: файл inFile.dat, який містить дійсні числа (доходи), по одному числу в кожному рядку. Перед кожним числом записаний символ „M”, для доходів мужчин або „F” для доходів жінок. Цей код стоїть в першій позиції у кожному рядку вводу і відділяється від значення доходу пробілом.

Результати виводу: еходрук вхідних даних, кількість жінок і їх середній дохід, кількість мужчин і їх середній дохід.

Задача розбивається на три основні етапи. Спочатку необхідно обробити дані, перелічити і просумувавши величини доходів мужчин і жінок. Потім необхідно знайти відповідні середні значення. На завершення необхідно вивести отримані результати.

Контрольні запитання

1. Оператор while.
2. Етапи виконання циклу.
3. Цикли з while. Цикли які керуються лічильником та подією.
4. Циклічні підзадачі.
5. Як проектувати цикл?
6. Вкладена логіка.

Лабораторна робота № 11
ТЕМА: ФУНКЦІЇ, ЩО НЕ ПОВЕРТАЮТЬ ЗНАЧЕННЯ

МЕТА РОБОТИ: навчитися використовувати власні функції користувача, що не повертають значення для розв'язування задач.

Теоретичні відомості

В С++ існують два види підпрограм: функції, що повертають значення і функції типу void, які не повертають значення модулю, що їх викликає. Розглянемо програму, що використовує void-функції. Нехай треба надрукувати за допомогою програми таке повідомлення:

```
**************
**************
Ласкаво просимо!
**************
**************
**************
```

Якщо записати два модулі першого рівня як функції типу void, то main буде виглядати зовсім просто:

```
int main()
{
        Print2lines();
        cout<<"Ласкаво просимо!"<<endl;
        Print4lines();
        return 0;
}
```

Тут є два функціональні виклики Print2(4)lines. Обидві функції не мають параметрів, тобто не містить списку параметрів в круглих дужках після імені функції. Запишемо наступний фрагмент:

```
void Print2lines()              //заголовок функції
{
        cout <<"**************"<<endl;
        cout <<"**************"<<endl;
}
```

Цей фрагмент називається описом функції. Опис функції – це програмний код від заголовка до кінця блоку, що є тілом функції. Заголовок функції починається із слова void, що вказує компілятору, що дана функція не повертає значення.

А тепер запишемо функцію main() і дві інші функції у вигляді закінченої програми.

```
//**************
//Програма  Welcome
//Дана програма виводить повідомлення "Ласкаво просимо!"
//**************
include <iostream.h>
```

```
void Print2lines(); //Прототипи функцій.
void Print4lines();

int main()
{
        Print2lines(); //Виклик функції
        cout<<”Ласкаво просимо!”<<endl;
        Print4lines(); //Виклик функції
return 0;
}
//***************
void Print2lines()  //Заголовок функції
//Ця функція друкує два рядки із зірочок.
{
        cout <<”***************”<<endl;
        cout <<”***************”<<endl;
}
//******************
void Print4lines()  //Заголовок функції
//Ця функція друкує чотири рядки із зірочок
{
        cout <<”***************”<<endl;
        cout <<”***************”<<endl;
        cout <<”***************”<<endl;
        cout <<”***************”<<endl;
}
```

Умові C++ описи функцій можуть з'являтися в будь-якому порядку. В програмі Welcome вирази перед функцією main() називаються прототипами функцій. Функція main() включає ідентифікатори Print2lines() і Print4lines(), але самі описи цих функцій з’являються пізніше. Тому потрібно вказати прототипи функцій, повідомивши попередньо компілятору, що Print2(4)lines() є іменами функцій, що не повертають значень і не містять жодних параметрів.

Параметри, що перечислюються при зверненні до функції, називаються фактичними параметрами (actual parameters). При описі функції в її заголовку використовуються формальні параметри (formal parametera). При виклику функції фактичні параметри ставляться у відповідність формальним параметрам згідно їх розміщення (зліва направо), а потім виконання передається першому із виконуючих операторів у тілі функції. Після виконання останньої інструкції, керування вертається у точку виклику функції.

В C++ кожний ідентифікатор перед використанням повинен бути оголошеним. У випадку функції, оголошення має фізично випереджати будь-яке звернення до неї.

В С++ оголошення функції, в якому опущено тіло, називається прототипом функції, а оголошення, яке включає тіло – описом функції.

Список формальних параметрів в прототипі функції має містити типи даних параметрів, при цьому їх імена означувати не обов'язково, а при описі імена всіх формальних параметрів необхідно вказати.

Так як тіло функції – це блок, то будь-яка функція (не тільки main) може містити оголошення змінних всередині тіла. Вони доступні лише всередині блоку, в якому оголошені і тому називаються локальними змінними. Змінні, що оголошені поза програмними функціями називаються глобальними.

Коли функція виконується вона використовує фактичні параметри, що передані їй при виклику. С++ підтримує два види формальних параметрів: параметри, що передаються за значенням (value parameters) і параметри, що передаються за посиланням (reference parameters). Параметр, що передається за значенням оголошується без амперсанда (&) після імені типу даних. В цьому випадку функція отримує копію значення фактичного параметра. При оголошенні параметра, що передається за посиланням, до імені типу даних додається амперсанд. В такому випадку функції передається адреса фактичного параметра.

```
void Example  (int& param1, //параметр , що передається за посиланням
        int param2,  // параметр що передається за значенням
        float param3) //          -//-
```

Для простих типів даних (int, char, float) параметр, що передається за значенням є прийнятий за замовчуванням.

Число фактичних параметрів у зверненні до функції повинно відповідати числу формальних параметрів у її заголовку. Крім того, кожен фактичний параметр повинен мати той же тип даних, що і формальний, що стоїть на відповідному місці у списку.

FormalParameterList(in a function prototype)

DataType& VariableName, DataType& VariableName…

Якщо параметри в парі мають різні типи даних, відбувається неявне зведення типів. Наприклад, якщо формальний параметр має тип int, а фактичний параметр тип float, то перед передачею у функцію вираз зводиться до значення типу int.

Різниця між параметрами, що передаються за значенням і локальними змінними полягає в тому, що у момент виклику функції значення локальних змінних не визначені, у той час як параметри, що передаються за значенням, автоматично ініціалізуються відповідними фактичними аргументами. Оскільки вміст параметрів, що передаються за значенням, не зберігається після завершення функції, вони не можуть застосовуватися для повернення інформації у викликаючу функцію. Для вирішення цієї проблеми використовуються параметри, що передаються за посиланням. Вони називаються так, бо викликаюча функція може проглядати і змінювати фактичний параметр викликаючої функції. Коли функція викликається з використанням параметра, що передається за посиланням, то їй передається місцезнаходження (тобто адреса в пам'яті) фактичного параметра, а не його значення. Треба бути уважним використовуючи формальний параметр, що передається за посиланням, тому що будь-яка його зміна діє на фактичний параметр у коді виклику.

В якості фактичного параметра, що передається за посиланням може використовуватися тільки змінна, тому що функція може присвоювати нове значення цьому фактичному параметру, в той час, як в якості параметра, що передається за значенням, може використовуватися довільний складний вираз.

Нехай є функція із заголовком:

```
void DoThis      (float val,     //параметр що передається за значенням
          int& count )//параметр що передається за посиланням
```

Тоді справедливі наступні звернення до неї:

```
DoThis (someFloat, someInt);
DoThis (9.83, int Counter);
DoThis (4.9*sqrt(y), myInt);
```

У випадку використання параметрів, що передаються за посиланням фактичний і формальний параметри не одного типу, то буде у програмі помилка.

Завдання для роботи

1. Створіть еквівалент калькулятора, що виконує чотири основних арифметичних операції. Програма повинна запрошувати ввід користувачем першого операнда, знаку операції і другого операнда, які передаються у якості параметрів функції, що не повертає значення. Для зберігання операндів слід використовувати змінні дійсного типу. Перевірити вибір операції можна за допомогою оператора if. У кінці програма повинна відображати результат на екрані. Результат роботи програми може виглядати таким чином:

   Введіть перший операнд, операцію і другий операнд: 10 / 3
   Результат дорівнює 3.333333
   Виконати ще одну операцію (y/n)? y
   Введіть перший операнд, операцію і другий операнд: 12 + 100
   Результат дорівнює 112
   Виконати ще одну операцію (y/n)? N

2. Створіть програму, що викликає функцію, яка приймає два аргументи. Перший аргумент символ, а другий аргумент типу int не рівний нулю. Функція виводить символ стільки разів, скільки рівне значення другого введеного параметра.

Контрольні запитання

1. Які типи функцій бувають у С++?
2. Як оголошується прототип функції?
3. Як здійснюється виклик функції?
4. Що таке опис функції, його синтаксис?
5. Які типи параметрів використовуються при виклику функції?
6. Які типи параметрів використовуються при описі функції?

Лабораторна робота № 12
ТЕМА: ФУНКЦІЇ, ЩО ПОВЕРТАЮТЬ ЗНАЧЕННЯ

МЕТА РОБОТИ: навчитися використовувати власні функції користувача, що повертають значення для розв'язування задач.

Теоретичні відомості

Модулі в С++ називають функціями і класами. Підпрограми мовою С++ створюються об'єднанням нових функцій, які підготовлені програмістом, з функціями, які вже є в стандартній бібліотеці С++, і об'єднанням нових класів, підготовлених програмістом, з класами, які вже є в різних бібліотеках класів. Аргументами функцій можуть бути константи, змінні або вирази.

Функція – це підпрограма, яка може оперувати даними і повертати значення. Кожна програма мовою С++ містить хоча б одну функцію – main(), яка в разі запуску програми викликається автоматично.

Функція дозволяє створювати модулі. Всі змінні оголошуються в описах функцій локальними змінними – вони відомі тільки для функції, в якій їх описано. Більшість функцій мають список параметрів, які забезпечують значення для зв'язування інформації між функціями. Параметри теж є локальними змінними.

Формат опису функції:

Тип_значення,_яке_повертається, ім'я_ функції (список_параметрів)

{оголошення і інструкції}

Є три способи повернення керування в точку, з якої було викликано функцію. Якщо функція не повинна повертати результат, керування повертається або з досягненням правої фігурної дужки, яка закриває функцію, або під час виконання інструкції return;. Якщо функція повинна повертати результат, то інструкція return вираз; повертає значення виразу інструкції, яка викликала цю функцію.

Мінімальною функцією є, наприклад, функція Dummy() { }, яка не робить ніяких дій.

Функції розбивають великі обчислювальні завдання на маленькі підпрограми і дозволяють використовувати в роботі ті, що вже зроблено іншими. Відповідні функції часто можуть приховувати в собі деталі виконаних у різних частинах програми операцій, які знати немає потреби, сприймаючи тим самим всю програму як ціле, і полегшує процедуру внесення змін. Мову С++ було розроблено з метою зробити функції ефективними і зручними для використання. Програми мовою С++ складаються з великої кількості маленьких функцій. Програма може міститись в одному або декількох вхідних файлах. Вхідні файли можуть компілюватися окремо і завантажуватися разом поряд із скомпільованими раніше функціями з бібліотек.

Розрізняють три способи оголошення функції: записати прототип функції у файл, а потім використати директиву #include, яка включить його в текст програми; записати прототип у той файл, де ця функція застосовується; функцію означати до її першого виклику.

Для своїх функцій програміст сам повинен потурбуватися про внесення в програму відповідних прототипів.

Програма – це набір визначень окремих функцій. Зв'язок між функціями здійснюється через аргументи і значення, що повертаються функціями або через зовнішні змінні. Функції можуть розміщуватися у вихідному файлі в будь-якому

порядку, а вся програма – у декількох файлах, але так, щоб жодна функція не розщеплювалася.

Різниця між функціями типу viod і функціями, що повертають значення полягає в тому, що виклик перших – це самостійна інструкція, а виклик других повинен бути частиною деякого виразу.

З точки зору розробника, функції. Що повертають значення, використовуються в тому випадку, якщо в результаті роботи функції необхідно отримати тільки одне значення, яке буде безпосередньо використовуватися у виразі. При оголошенні прототипу функції, що повертає значення вказується її тип, який відповідає значенню, що повинна повернути функція. З допомогою інструкції return в описі функції, вона може повертати тільки одне значення. Є дві форми інструкції return:

return;

яка допустима тільки у функціях типу void. Вона заставляє керування негайно покинути тіло функції і повертатися до оператора виклику і друга форма

return вираз;

яка допустима тільки у функції, що повертає значення.

Вимоги до синтаксису формальних і фактичних параметрів для функцій, що повертають значення, такий же як і для функцій типу void.

Інтерфейс функції, що повертає значення розробляється так само як і для функцій типу void. На практиці у списку формальних параметрів функції, що повертає значення, ніколи не використовуються параметри, що передаються за посиланням. Виключенням із цього правила є передача функції, що повертає значення змінної потоку вводу/виводу. В С++ змінна потоку може бути передана тільки за посиланням. Всередині функції. Що повертає значення, єдина операція, яка повинна виконуватися, – це перевірка стану потоку. Функція, що повертає значення, не повинна виконувати операції вводу/виводу. Такі операції будуть побічними ефектами.

Не існує жодних формальних правил для визначення того, коли використовувати функцію типу void, а коли функцію, що повертає значення, але є деякі керуючі принципи:

1. Якщо модуль повинен повертати більше одного значення або змінювати будь-які фактичні параметри. Не використовуйте функцію, що повертає значення.
2. Якщо модуль повинен виконувати ввід/вивід, то функція, що повертає значення не використовується.
3. Якщо є тільки одне значення, яке повертається із модуля і воно має тип Boolean, то для цього підходить функція, що повертає значення.
4. Якщо повертається тільки одне значення, яке повинне негайно використовуватися у виразі, то слід використовувати функцію, що повертає значення.
5. Якщо є сумнів у використанні типу функції, то використовуйте функцію типу void.
6. Якщо допускається використання обох типів функцій, то можна використовувати ту, яку більш зручно реалізувати.

Програма С++ складається з набору зовнішніх об’єктів, які є змінними або функціями. Термін «зовнішній» використовують як протиставлення терміну «внутрішній», яким описуються аргументи і автоматичні змінні всередині функцій. Зовнішні змінні визначено поза функцією і, отже, потенційно доступні для багатьох функцій. Самі функції завжди зовнішні, оскільки правила мови С++ не дозволяють визначати одні функції всередині інших. За замовчуванням зовнішні змінні є також і глобальними. Тому, якщо ім’я зовнішньої змінної яким-небудь чином описано, то будь-яка функція має доступ до цієї змінної.

Якщо ж зв’язок між функціями здійснюється за допомогою великої кількості змінних, зовнішні змінні виявляються більш зручними і ефективними, ніж використання довгих списків аргументів.

Причина використання зовнішніх змінних полягає в ініціалізації. Зокрема, зовнішні масиви можуть бути ініціалізовані, а автоматичні ні. Причину використання зовнішніх змінних зумовлено їх областю дії та часом існування. Автоматичні змінні є внутрішніми відносно функцій; вони виникають під час входу у функцію і зникають у разі виходу з неї. Зовнішні змінні, навпаки, існують постійно, а, отже, можуть зберігати свої значення в період від одного звернення до функції до іншого. Тому, якщо дві функції використовують деякі загальні дані, причому жодна з них не звертається до іншої, то найзручніше зберігати ці загальні дані у вигляді зовнішніх змінних, а не передавати їх у функцію і назад за допомогою аргументів.

Функції і зовнішні змінні, які входять до складу програми мовою С++, можуть компілюватися окремо. Крім того, програма може розміщуватися в декількох файлах, а раніше скомпільовані процедури завантажуватися з бібліотек. Тому виникає проблема встановлення описів змінних, щоб вони правильно сприймалися під час компіляції і забезпечувався правильний зв’язок частин програми під час завантаження.

Областю дії імені є та частина програми, у якій це ім’я визначено. Для автоматичної (локальної) змінної, описаної на початку функції, областю дії є та функція, у якій описано ім’я цієї змінної. При цьому змінні з різних функцій, але з одним іменем вважаються такими, що не стосуються одна до одної. Це справедливо також для аргументів функцій.

Область дії зовнішньої змінної поширюється від точки, в якій вона оголошена у вихідному файлі, до кінця цього файлу. Проте, якщо потрібно послатися на зовнішню змінну до її визначення, або якщо таку змінну визначено в іншому файлі, який відмінний від того, у якому її використовують, то потрібно застосувати опис extern.

Важливо розрізняти опис зовнішньої змінної і її визначення. Опис вказує властивості змінної (її ім’я, тип, розмір і т. ін.), а визначення викликає ще й виділення пам’яті. Якщо поза функцією з’являється рядок:

int sp;

то він визначає зовнішню змінну sp, викликає виділення пам’яті для неї і є описом для іншої частини цього файлу. Водночас рядок

extern int sp;

не створює змінної і їй не виділяється місце в пам’яті.

У всіх файлах, які належать до вихідної програми, має бути тільки одне визначення зовнішньої змінної, інші файли можуть містити опис extern для доступу до неї. Опис extern може бути і в тому файлі, де перебуває визначення. Будь-яка ініціалізація зовнішньої змінної виконується тільки у визначенні. У визначенні мають бути вказані розміри масивів, а в описі extern цього можна не робити.

Статичні змінні – це третій клас пам’яті в локальних і зовнішніх (extern) змінних. Статичні змінні можуть бути як внутрішніми, так і зовнішніми. Внутрішні статичні змінні, як і локальні, є локальними для деякої функції, але, на відміну від локальних, вони залишаються, а не з’являються, і зникають разом зі зверненням до функції. Це означає, що внутрішні статичні змінні забезпечують постійне, недосяжне ззовні зберігання всередині функції.

Зовнішні статичні змінні визначаються в іншій частині вихідного файлу, у якому вони описані.

Статична пам’ять, як внутрішня, так і зовнішня, визначається словом static, що стоїть перед описом. Змінна є зовнішньою, якщо вона описана за межами будь-якої функції, і внутрішньою, якщо її описано всередині функції.

Функції є зовнішніми об’єктами; їх імена відомі глобально. Можна оголосити функцію як static, і тоді її ім’я стає невідомим поза файлом, у якому його описано. У мові C++ static означає не тільки сталість, але й ступінь «приватності». Внутрішні статичні об’єкти визначено тільки всередині однієї функції, а зовнішні статичні об’єкти (змінні або функції) – тільки всередині того вихідного файлу, де вони з’являються, і їх імена не вступають у конфлікт із такими ж іменами з інших файлів.

Зовнішні статичні змінні і функції дозволяють організовувати дані та працюючі з ними внутрішні процедури так, що інші процедури і дані не можуть з ними конфліктувати навіть через недоречності.

Четвертий і останній клас пам’яті називають регістровим. Опис register вказує компіляторові, що регістрова змінна буде часто використовуватися. Коли можливо, змінні типу register розміщуються в машинних регістрах. Опис register виглядає як

```
register int x;
register char c;
```

де int можна пропустити. Опис register можна використати тільки для автоматичних змінних і формальних параметрів функцій так:

```
F(c,n)
register int c,n;
{  register int i;}
```

Зазначимо, що в регістрах можна розмістити небагато змінних кожної функції і тільки певних типів. У разі перевищення можливої кількості або використання недозволених типів слово register ігнорується. Крім того, неможливо вилучити адресу регістрової змінної.

Мова C++ не є мовою з блоковою структурою, і вона не дозволяє описувати одні функції всередині інших. Змінні ж можна визначати за методом блокового структурування. Описи змінних (включаючи ініціалізацію) можуть записуватись за фігурною дужкою { , що відкриває будь-який оператор, а не тільки за тією, з якої починається тіло функції. Змінні, які так описані, витісняють будь-які змінні із зовнішніх блоків, що мають такі ж імена, і залишаються визначеними до відповідної фігурної дужки }. Наприклад:

```
if (n>0)
{ int i;      /* оголошення нового і */
   for (i=0; i<n; i++)
 }
```

Областю дії змінної і є «дійсна» гілка if, де і ніяк не пов’язано ні з якими іншими і в програмі.

Блокова структура впливає на область дії зовнішніх змінних. Якщо дано описи:

```
int x;
F()
{  double x;  }
```

то опис x всередині функції F належить до внутрішнього змінної типу double, а поза F – до зовнішньої цілої змінної. Це стосується імен формальних параметрів:

```
int x;
F(x)
double x;
{  ...  }
```

Усередині функції F ім’я x належить до формального параметра, а не до зовнішньої змінної.

Якщо явної ініціалізації немає, то зовнішнім і статичним змінним присвоюється значення нуль, а локальні та регістрові змінні у цьому випадку мають невизначені значення. Прості змінні (не масиви або структури) можна ініціалізувати під час їх опису, додаючи за їх іменем знак рівності та константний вираз:

```
int x = 1;
char squote = '';
long day = 60*24;   /* хвилин за день */
```

Для зовнішніх і статичних змінних ініціалізація виконується тільки один раз, на етапі компіляції. Локальні та регістрові змінні ініціалізуються щоразу під час входження у функцію або блок. У випадку локальних і регістрових змінних ініціалізатор може бути будь-яким виразом, а не тільки константою.

Ініціалізації локальних змінних є скороченим записом операторів присвоєння. Краще використовувати явні присвоєння, оскільки ініціалізація в описах менш наочна.

Рекурсивна функція – це функція, яка викликає сама себе безпосередньо або побічно за допомогою іншої функції. Зауважимо, що з кожним рекурсивним викликом функцією себе утворюється новий набір усіх локальних змінних, який не залежить від попереднього набору.

Рекурсія не забезпечує економії пам’яті, оскільки доводиться створювати стек для обробки значень. Не приводить вона й до створення швидших програм. Але рекурсивні програми компактні й прості для розуміння та написання. Рекурсія особливо зручна для роботи з рекурсивно обумовленими структурами даних, наприклад з деревами.

Завдання до роботи

Написати програму, в якій функція main() викликає функцію означену користувачем, що отримує у якості аргументу значення температури у градусах за Цельсієм і повертає еквівалентне значення у градусах за Фаренгейтом. За запитом програми температуру у градусах за Цельсієм вводить користувач. Дані, що виводяться на екран, мають наступний вигляд:

Будь-ласка, введіть температуру у градусах за Цельсієм: 20
20 градусів за Цельсієм дорівнює 68 градусів за Фаренгейтом

Формула перерахунку має вигляд: $Фаренгейт = 1,8 \times Цельсій + 32,0$.

Створіть програму, яка повторно запрошує ввід пари чисел до тих пір, поки хоч би одне з чисел цієї пари виявиться рівним нулю. Для кожної пари програма за допомогою функції обчислює середнє гармонійне. Функція повинна повертати відповідь у функцію main(), яка відображує результат на екрані. Середнє гармонійне двох чисел – це зворотна величина середнього значення обернених величин цих чисел, яка може бути обчислена таким чином $середнє\ гармонійне = \frac{2.0xy}{x+y}$.

Створіть функцію, яка приймає як аргумент ціле число і повертає його факторіал. Нагадуємо, що 3 факторіал, записується як 3!, що дорівнює 3x2!, при цьому 0!, за означенням, рівний 1. У загальному випадку $n! = n(n-1)!$. Перевірте це в програмі, яка використовує цикл, щоб дозволити користувачеві вводити різні значення, для яких програма відображає факторіал.

Контрольні запитання

1. Як оголошуються функції, що повертають значення?
2. Які вимоги ставляться до формальних і фактичних параметрів функцій, що повертають значення?
3. Які наслідки буле мати не оголошення типу функції, що повертає значення?
4. Які використовуються принципі при виборі типу функції?
5. Які змінні є локальними і глобальними?
6. Які інші типи змінних вам відомі?

Лабораторна робота № 13
ТЕМА: ПРОГРАМИ ІЗ ВИКОРИСТАННЯМ КІЛЬКОХ ФУНКЦІЙ КОРИСТУВАЧА

МЕТА РОБОТИ: навчитися використовувати власні функції користувача для розв'язування задач підвищеної складності.

Завдання до роботи

Задача. Компанія, в якій ви працюєте, недавно збільшила кількість літаків, купивши Beechcraft Starship-1. Пілот повинен знати загальну вагу завантаженого літака при підйомі, а також цент ваги. Це необхідно для керування. Якщо літак важить надто багато, то він не зможе піднятися. Якщо центр ваги виходить за обмеження, встановлені для літака, то ним буде неможливо керувати. Будь-яка ситуація може привести до аварії. Вас попросили написати програму, яка визначає вагу і центр ваги цього літака, базуючись на чисельності команди і пасажирів, а також на вазі багажу, туалету і палива.

Вхідні дані. Чисельність команди, число пасажирів, вага вмісту туалету, вага багажу, кількість палива в галонах.

Обговорення задачі. Як в більшості реальних задач, базисний розв'язок простий, але він ускладнений спеціальними умовами. Ви повинні використовувати функції, що повертають значення і функції типу void, щоб основна функція все одно виглядала просто.

Загальна вага – це сума ваги порожнього літака додати вагу кожного окремого елемента: члени команди, пасажири, багаж, вміст туалету і паливо. Будемо використовувати середню вагу людини 170 фунтів, для обрахунку загальної ваги людей. Вага багажу і вміст туалету відомий. Паливо важить 6,7 фунтів на галон. Таким чином, загальна вага буде наступною:

Загальна вага = порожній літак + (команда + пасажири) * 170 + багаж + туалет + паливо *6,7

Щоб обчислити центр ваги, кожна вага множиться на відстань від неї до носу літака, і результати – що називаються моментами – після цього додаються і діляться на загальну вагу.

Таким чином, формула буде мати наступний вигляд:

Центр ваги = (момент літака + момент команди + момент пасажирів + момент багажу + момент палива) / загальна вага

Інструкція до літака Starship-1 містить відстань від носу літака до крісел команди, туалету, багажного відділення і паливних баків.

В літаку є чотири ряди пасажирських місць, тому обчислення залежать від того, де сидять окремі пасажири. Ви повинні зробити припущення де сидять пасажири. Кожен ряд має два місця. Найбільш популярними вважаються місця у другому ряді, тому що вони розміщені близько до виходу і орієнтовані по ходу літака. Як тільки другий ряд заповнений, пасажири звично займають місця в першому ряді, розміщуючись лицем до попутника. Ряд 3 звично заповнюється наступним, не дивлячись на те, що він орієнтований задом на перед, тому що у четвертому ряді

незручні відкидні крісла. Наступна таблиця містить відстані від кожної одиниці ваги до носу літака.

№ п/п	Одиниця ваги	Відстань до носу літака (в дюймах)
1.	Місця команди	143
2.	Місця 1 ряду	219
3.	Місця 2 ряду	265
4.	Місця 3 ряду	295
5.	Місця 4 ряду	341
6.	Туалет	182
7.	Багаж	386

Відстань до палива міняється, оскільки в літаку є декілька баків, що розміщені в різних місцях. Як тільки паливо залите в літак, воно автоматично роз приділяється по бакам, так що центр ваги міняється після заправки. Існує чотири формули для обчислення відстані від носу літака до центру паливних баків в залежності від того, скільки палива залито. Наступна таблиця містить ці формули.

№ п/п	Кількість палива (G) (в галонах)	Формула обрахунку відстані (D)
1.	0-59	D=314.6 * G
2.	60-360	D=305.8+(-0.01233*(G-60))
3.	361-520	D=303.0+(0.12500*(G-36))
4.	521-565	D=323.0+(-0.04444*(G-521))

Вам треба написати функцію, яка повертає значення обчислення кожного із моментів. Назвемо ці функції відповідно: CrewMoment, PassengerMoment, CargoMoment і FuelMoment. Тоді центр ваги буде обчислюватися за формулою, що наведена вище із такими параметрами:

Центр ваги = (CrewMoment(crew)) + PassengerMoment(passengers) + CargoMoment(closet,baggage) + FuelMoment(fuel) + момент літака) / загальна вага

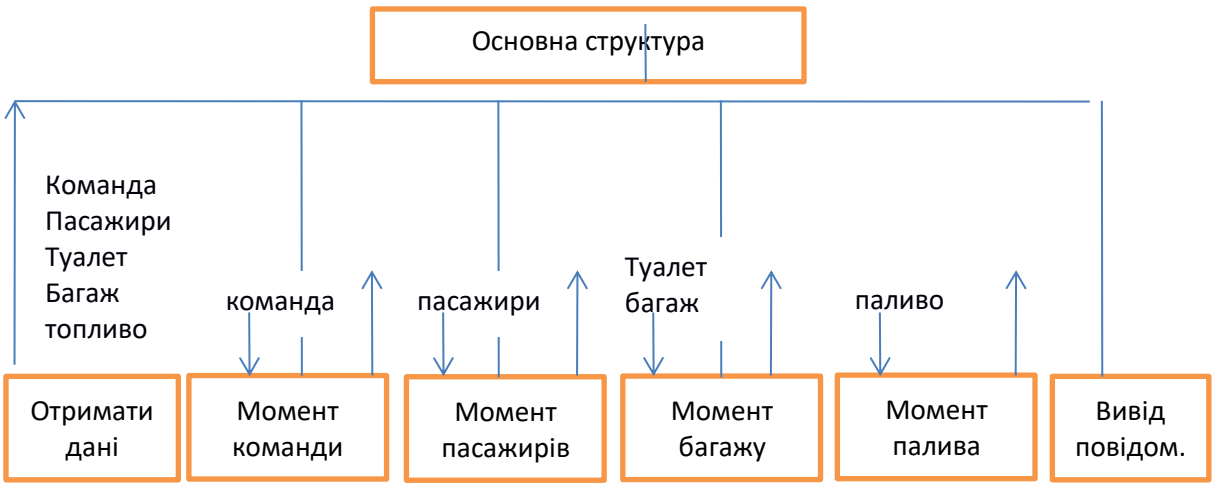
Вага порожнього літака складає 9987 фунтів, а центр ваги в цьому випадку розміщений на відстані 319 дюймів від носу літака. Таким чином, момент літака дорівнює 3153953 дюйм-фунтів.

Тепер у вас достатньо інформації для написання алгоритму розв’язку цієї задачі. Для отримання даних ви використайте функцію типу void. Функція main() повинна обчислювати загальну вагу і викликати відповідні функції, про які мова йшла вище при обчисленні центра ваги. Після цього вона повинна видати ці два результати. При виводі результатів має бути надруковане повідомлення-попередження для пілота, якщо отримані дані близькі до критичних. Це повідомлення повинне виводитися за допомогою окремої функції типу void PrintWarning(). Чисельність команди 1 або 2 особи, максимальна вага туалета 160 фунтів, максимальна вага багажу 525 фунтів.

Діаграма модульної структури

Наступна діаграма вводить нове позначення. Прямокутник, що відповідає кожній функції, що повертає значення, має справа стрілку, яка напрямлена вверх. Ця стрілка означає значення, що повертається функцією.

Результати роботи захистити і здати у якості звіту викладачу.


Основна структура
Команда
Пасажири
Туалет
Багаж
топливо
команда
пасажири
Туалет
багаж
паливо
Отримати дані
Момент команди
Момент пасажирів
Момент багажу
Момент палива
Вивід повідом.

Лабораторна робота № 14
ТЕМА: ВИКОРИСТАННЯ ОПЕРАТОРА SWITCH ДЛЯ ОРГАНІЗАЦІЇ РОЗГАЛУЖЕННЯ. ЗАСТОСУВАННЯ В ПРОГРАМАХ BREAK, CONTINUE, GOTO

МЕТА РОБОТИ: навчитися використовувати оператор switch для організації розгалужень в програмі із багато чисельними вітками та використовувати при цьому оператори зміни керування потоку.

Теоретичні відомості

Вираз **switch** – це керуюча структура для вибору з багаточисельними вітками, аналогічна вкладеним виразам if. Умова **switch** – це інструкція, значення якої визначає, по якій із віток піде керування. Вона не може мати тип **float**.

```
switch (letter)
{
        case ‘X’:       вираз 1;
        break;
        case ‘L’:
        case ‘M’:       вираз 2;
        break;
        default:        вираз 3;
}
вираз 4;
```

letter – це умова **switch**. Вона означає, що якщо **letter** дорівнює **‘X’**, то виконати **вираз 1**. Якщо **‘L’** або **’M’** – **вираз 2**. Якщо **letter** не дорівнює жодному значенню приведеному вище, то виконати **вираз 3**. Інструкція **break** перериває потік керування і переводить його на перший вираз, що слідує за структурою **switch**, тобто **вираз 4**. Якщо опустити **break**, то вирази будуть виконуватись всі послідовно.

Умова **switch** – це вираз цілого типу – **char, short, int, long** або **enum**. Необов’язкове слово перед початком кожної вітки – це або **case**, або **default**. Вираз умови **case** має бути **цілочисельним**. Його операндами мають бути літерні або іменовані константи. Як бачимо із прикладу одному виразу може передувати більше, ніж одна умова **case**. Однак, кожна умова може з’являтися тільки один раз у структурі **switch**, **default** може стояти також тільки один раз. У структуру **switch** можна вкласти коментарі, які будуть виводитись:

```
switch (grade)
{
        case ‘A’: cout << “вивід повідомлення”;
        break;
}
```

Інструкція **break** може також використовуватись в циклах. Вона викликає негайний вихід із самого внутрішнього виразу **switch**, **while**, **do-while** або **for**,у якому зустрічається. Якщо цикл вкладений в інший і в ньому зустрічається **break**, то вихід здійснюється тільки з внутрішнього, а не із зовнішнього циклу. Добрим правилом є використання **break** у циклах тільки як останній засіб. **break** треба використовувати для уникнення неприємної ситуації з багаточисельними **булевими прапорцями** і вкладеними умовами **if**.

Другим засобом для зміни напряму потоку керування в програмах на **C++** є інструкція **continue**. Її можна використовувати тільки в циклах; поява **continue**

перериває тільки поточну ітерацію (але не весь цикл в цілому). Поява цієї інструкції переміщує керування в самий кінець циклу, пропускаючи всі решту вирази в його тілі – на етап приготування до наступної ітерації.

```
for (dataCount=1; dataCount<=500; dataCount++)
{
        dataFile>>inputVal;
        if (inputVal<=0)
                continue;
        cout<<inputVal;
        ...
}
```

Якщо значення змінної **inputVal** менше або рівне **0**, то керування переходить у кінець циклу.

Використання оператора **goto** змінює природне виконання програми і переводить потік керування до мітки (позначки) на яку вказує **goto**.

Синтаксис застосування оператора goto має наступний вигляд:

goto <мітка>;

де мітка це ідентифікатор, який записується перед виразом, що має виконуватися через двокрапку:

```
m: cout<<”Введіть число”<<endl;
…
goto m;
…
```

Завдання до роботи

1. Напишіть програму на мові С++, яка вводить абревіатуру із двох букв одного із 50 штатів США і друкує повну назву цього штату. Якщо абревіатура введена неправильно, програма повинна вивести відповідне повідомлення про помилку і вивести повідомлення ввести абревіатуру знову. В таблиці приведені назви і абревіатури 50 штатів США. При розв’язуванні задачі використайте вкладені структури switch, в яких зовнішній рівень в якості умови перевіряє першу букву абревіатури.

Штат	Скорочення	Штат	Скорочення
Алабама	AL	Монтана	MT
Аляска	AK	Небраска	NE
Арізона	AZ	Невада	NV
Арканзас	AR	Нью-Хемпшир	NH
Каліфорнія	CA	Нью-Джерсі	NJ
Колорадо	CO	Нью-Мексіко	NM
Конектікут	CT	Нью-Йорк	NY
Делавер	DE	Північна Кароліна	NC
Флорида	FL	Північна Дакота	ND
Джорджія	GA	Огайо	OH
Гавайї	HI	Оклахома	OK
Ідахо	ID	Орегон	OR
Іллінойс	IL	Пенсільванія	PA
Індіана	IN	Род-Фйленд	RI

Штат	Скорочення	Штат	Скорочення
Айова	IA	Південна Кароліна	SC
Канзас	KS	Південна Дакота	SD
Кентуккі	KY	Теннессі	TN
Луїзіана	LA	Техас	TX
Мейн	ME	Ута	UT
Меріленд	MD	Вермонт	VT
Массачусетс	MA	Вірджинія	VA
Мічіган	MI	Вашингтон	WA
Мінесота	MN	Західна Вірджинія	WV
Міссісіпі	MS	Вісконсін	WI
Міссурі	MO	Вайомінг	WY

Нехай оплата робіт залежить від типу виконаної роботи чи виду підприємницької діяльності (А, Б, В) і нараховується за формулою

$$y = \begin{cases} 100\left|f_{i,2}(i)+50\right| & \text{для робіт типу А,} \\ 150\left|f_{i,3}(i)+100\right| & \text{для робіт типу Б,} \\ 200\left|f_{i,4}(i)+135\right| & \text{для робіт типу В,} \end{cases}$$

де і – номер варіанта. Для робіт типу А податок становить 10%, для Б – 15%, для В – 20%. Ввести тип робіт. Вивести нараховану суму, суми податку і суму до видачі. Розв'язати задачу двома способами, використовуючи:

1) команду вибору switch;
2) команду goto.

№ п/п	Функція $f_n(x)$
1	9,2cos2x-|sin(x)/1,1|
2	12,4sin(|x/2,1|)-8,3cos(1,2x)
3	|cos(x)/2,7|+9,1sin(1,2x+1)
4	|sin(x)/3,12+cosx^2|-8,3sin(3x)
5	cos|2x|/1,12-cos(3x-2)+6,15
6	sin(x)cos^{2}xsin(x+1,4)/0,85+7,14
7	|sin(2x-1,5)+3sin4x|+2,38
8	cosx^2sin(2x-1)+4,29
9	cos($x^{2,4}$+1)-|sin2x-5,76|
10	sinx-cosx^3sin(x^2-4,2)+4,27
11	|sin12xcos|2x|/3|+4,21
12	cosx^3/2,1+cosx^2/1,1-8,3sin(3x+1)

Виконати відповідні завдання, відкомпілювати програму, вивести результати і захистити у якості звіту.

Контрольні запитання

1. Синтаксис керуючої структури switch.
2. Який тип даних може бути умовою switch?
3. Принципи роботи операторів break, continue?
4. Оператор зміни потоку керування goto. Що таке мітка?

Лабораторна робота № 15
ТЕМА: ЦИКЛИ DO-WHILE

МЕТА РОБОТИ: навчитися використовувати у циклічних процесах цикл постперевірки do-while.

Теоретичні відомості

Вираз **do-while** – це циклічна керуюча структура, в якій умова перевіряється у кінці циклу. Це гарантує виконання тіла циклу принаймні один раз.

```
do
{
        вираз 1;
        вираз 2;
        .
        .
        .
        вираз n;
} while(умова);
```

Тут виконуються вирази між словами **do** і **while** до тих пір, поки “умова” є відмінною від нуля **(true)** в кінці циклу.

Порівняємо цикли **while** і **do-while**, які знаходять першу крапку у вхідному файлі даних:

```
dataFile>>inputChar;
while (inputChar!=’.’)
        dataFile>>inputChar;
```

```
do
        dataFile>inputChar;
while (inputChar!=’.’);
```

У першому випадку треба здійснити першочергове читання, щоб присвоїти змінній **inputChar** значення перед входом у цикл. Цього не потрібно робити у другому випадку, оскільки вираз вводу в тілі циклу виконується перед перевіркою умови циклу. Наведемо приклад порівняння віку:

```
cout <<”введіть вік:”;
cin>>are;
while (are<=0)
{
        cout<<”вік має бути позитивною величиною.”<<endl;
        cout<<”введіть вік:”;
        cin>>are;
}
```

А тепер за допомогою циклу do-while:

```
do
{
        cout<<”введіть вік:”;
        cin>>are;
        if (are<=0)
        cout<<”вік має бути позитивною величиною.”<<endl;
} while (are<=0);
```

Бачимо, що у другому випадку не треба двічі вставляти в текст вивід запрошення, але в ньому перевірка здійснюється двічі.

Цикл **do-while** можна використовувати для реалізації циклів, що керуються лічильником, якщо попередньо відомо, що тіло циклу завжди виконується принаймні один раз. Наведемо приклад обчислення суми цілих чисел від 1 до n.

```
sum=0;
counter=1;
while (counter<=n)
{
        sum=sum+counter;
        counter++;
}
```

```
sum=0;
counter=1;
do
{
        sum=sum+counter;
        counter++;
} while (couhter<=n);
```

Якщо n – позитивне число, то ці версії еквівалентні. Але якщо n дорівнює 0 або від'ємне, то два цикли дають різні результати. У першому випадку sum=0, так як тіло циклу не виконається жодного разу, а в другому випадку значення суми буде відмінне від нуля, оскільки тіло циклу раз виконується, після чого виконується перевірка умови.

Цикл **while** називається циклом **попередньої перевірки**. Цикл **do-while** називається циклом **постперевірки**.

Завдання до роботи

Використовуючи цикл do-while напишіть програму на C++, яка перетворює літери абетки у відповідні цифри на телефонному апараті. Програма повинна дозволяти користувачу вводити літери повторно, доки не зустрінеться літера Q або Z (літери Q і Z не використовуються в телефоні). При вводі будь-якого не абеткового символу повинно виводитися повідомлення про помилку. Між буквами і цифрами на телефонному апараті існує таке співвідношення:

ABC=2	JKL=5	TUV=8
DEF=3	MNO=6	WXY=9
GHI=4	PRS=7	

Приклад роботи такої програми:

```
Введіть літеру: P
Буква P відповідає цифрі 7 на телефоні.
Введіть літеру: D
Буква D відповідає цифрі 3 на телефоні.
Введіть літеру: 2
Введений неправильний символ. Введіть Q або Z для виходу.
Введіть літеру: Z
Вихід.
```

Обчислити значення змінної z свого варіанта:

1) $z=a+b$;
2) $z=ab$;
3) $z=tg(b)-a$;
4) $z=(a+b)^2$;
5) $z=5ab-4$;
6) $z=\sin(a)+b$;
7) $z=a^b$;
8) $z=a^2+3b$;
10) $z=ab-\pi$;
11) $z=a-2b$;
12) $z=b^a$;
13) $z=\cos(ab)$;
14) $z=|a-b|$;
15) $z=ctg(2a)-b$;
16) $z=\exp(3ab)$;
17) $z=4ba-b$;
19) $z=|l2a-b|$;
20) $z=2a-b$;
21) $z=tg(a+b)$;
22) $z=\ln|a+4b|$;
23) $z=3ab-b$;
24) $z=a+\exp(b)$;
25) $z=5a-2b$.

9) $z=(ab)^{1/4}$; 18) $z=2a-b$;

$$a=\sum_{k=i}^{i+5} f_{i+5}(k),\ b=\prod_{k=i}^{i+3} f_{i+6}(k)$$

де , i – номер варіанта, k – цілі числа. Функції вибрати з таблиці. Вивести значення i, a, b, z.

Таблиця. Функції.

№ п/п	Функція $f_n(x)$
1	9,2cos2x-|sinx/1,1|
2	12,4sin|x/2,1|-8,3cos1,2x
3	|cosx/2,7|+9,lsin(1,2x+1)
4	|sinx/3,12+cosx^2|-8,3sin3x
5	cos|2x|/1,12-cos(3x-2)+6,15
6	sinxcos^{2}xsin(x+1,4)/0,85+7,14
7	|sin(2x-1,5)+3sin4x|+2,38
8	cosx^{2}sin(2x-1)+4,29
9	cos($x^{2,4}$+1)-|sin2x-5,76|
10	sinx-cosx^{3}sin(x^2-4,2)+4,27
11	|sin12xcos|2x|/3|+4,21
12	cosx^3/2,1+cosx^2/1,1-8,3sin(3x+1)
13	sinx^{2}cosx^3-sinx+5,2
14	2sinxsin(2x-1,5)cos(2x+1,5)-6
15	|cosx^2-0,51|sin(3x-4)-4,44
16	cos2,lxsin|x|/0,15-5,8
17	|cos2x^3+2sin(x/1,2-3,4)|+10,51cos|3x|
18	|sin(x^2/1,5-2)|+11,73cos(1,6x-1)
19	13,4cos|x|sin(x^2-2,25)
20	|cos(x^2-3,8)|/4,5-9,7sin(x-3,1)
21	13,4sin(-1,26x)cos|x/7,5|
22	2sin|2x|cos2x-11,6sin(x/0,4-1)
23	sin|x|/0,1+9,4sin(3x-2,5)
24	10,8|cos(x^2/1,13)|sin(x+1,4)
25	11,2cos(2x-1)+|sin1,5x|/1,7

Контрольні запитання

1. Який синтаксис циклу do-while?
2. Яка різниця між циклами while і do-while?
3. Коли зручно застосовувати до циклічного процесу цикл do-while?

Лабораторна робота № 16
ТЕМА: ЦИКЛИ FOR

МЕТА РОБОТИ: навчитися використовувати у циклічних процесах цикл for.

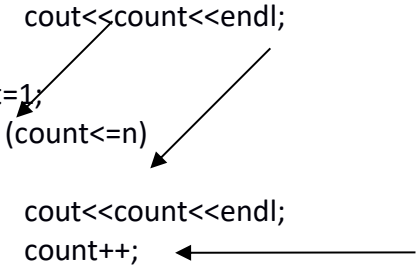
Теоретичні відомості

Вираз **for** полегшує написання циклів, що керуються лічильником. Наступний фрагмент друкує цілі числа від 1 до n:

```
for    (count=1; count<=n; count++)
          cout<<count<<endl;
```

for – це більш коротка форма циклу **while**. Фактично, компілятор перетворює вираз **for** у еквівалентний цикл **while** наступним чином:

```
for    (count = 1; count<=n; count++)
          cout<<count<<endl;
```



```
count=1;
while (count<=n)
{
          cout<<count<<endl;
          count++;
}
```

Цикли **for** можуть бути так само вкладеними, як і **while** чи **do-while**.

```
for (lastNum=1; lastNum<=7; lastNum++)
{
          for (numToPrint=1; numToPrint<=lastNum; numToPrint++)
          cout<<numToPrint;
cout<<endl;
}
```

Вивід буде таким:

```
1
12
123
1234
12345
123456
1234567
```

Завдання для роботи

Виконати табуляцію функції $y=f_{i+8}(x)$ на проміжку [0; i] з кроком h=0.1i, де i – номер варіанта використовуючи цикл for. Результати обчислень вивести у вигляді таблиці пар чисел x, y. Виконати завдання пошуку даних відповідно до вашого варіанта. Якщо шуканих даних немає, вивести про це повідомлення. Відповідну функцію взяти із таблиці до лабораторної роботи № 15.

1. Обчислити суму першого й останнього значень функції. Визначити кількість усіх значень.
2. Обчислити суму й добуток усіх значень функції y, для яких виконуються нерівності y<-3,2 або y>0.

3. Обчислити добуток та кількість усіх значень функції y, для яких виконуються нерівності y<-3 або y>0,4.
4. Обчислити добуток усіх від'ємних значень функції y та визначити кількість додатних.
5. Обчислити добуток значень аргумента (x), для яких досягаються мінімальне та максимальне значення функції y.
6. Скільки було від'ємних значень? Визначити максимальне значення.
7. Визначити суму додатних значень функції та кількість від'ємних.
8. Скільки від'ємних і додатних значень має функція y?
9. Обчислити модуль різниці максимального та першого значень y.
10. Обчислити добуток від'ємних значень функції y. У якій точці (x) функція набуває максимального значення?
11. Обчислити суму квадратів усіх додатних значень функцій y. Визначити, для якого x функція набуває мінімального значення.
12. Обчислити суму та кількість додатних значень функції y.
13. Обчислити суму всіх значень функції y, для яких виконуються нерівності y<1,2 або y>4. Визначити максимальне значення функції.
14. Обчислити добуток додатних значень і кількість від'ємних.
15. Обчислити добуток усіх значень функції y, для яких справджується нерівність 1<y<3,1. Визначити, для якого x функція набуває максимального значення.
16. Обчислити кількість і добуток усіх від'ємних значень y.
17. Обчислити суму квадратів і добуток усіх значень функції y, для яких справджується нерівність -2,41<y<5.
18. Обчислити модуль добутку максимального та мінімального значень.
19. Обчислити середнє арифметичне всіх від'ємних значень функції.
20. Обчислити суму кубів усіх додатних значень та їхню кількість.
21. Знайти середнє арифметичне тих значень функції y, для яких виконуються нерівності y<0 або y>1.
22. Знайти мінімальне значення функції, а також визначити значення аргумента, для якого воно досягається.
23. Обчислити суму та кількість тих значень функції y, для яких виконується нерівність 0<y<1.
24. Обчислити кількість і добуток тих значень функції y, для яких виконуються нерівності 1,3<y<5.
25. Яких значень функції більше: додатних чи від'ємних?

Контрольні запитання

1. Який синтаксис циклу for?
2. Коли використовується цикл for?
3. Провести порівняльний аналіз циклу for і while.

Лабораторна робота № 17-21
ТЕМА: ОДНОМІРНІ МАСИВИ

МЕТА РОБОТИ: навчитися використовувати одномірні масиви при розв'язуванні задач, використовувати їх в якості своєрідної бази даних, сортувати елементи одномірного масиву, шукати мінімальні і максимальні значення в масиві.

Теоретичні відомості

Значення простого типу – це окрема одиниця даних і вона не може бути розбита на складові частини. Складений тип даних – це набір одиниць даних, що об'єднані одним іменем. Не зважаючи на те, що вся група має одне ім'я, можна звертатися окремо до кожного компонента. Прості типи даних є окремими блоками складених типів. Складений тип збирає набір значень і групує їх у певному порядку. Метод отримання доступу до окремого елемента структурного типу залежить від розміщення компонентів.

Нехай треба прочитати список із 1000 значень і роздрукувати його у зворотному порядку.

Довжина коду цієї програми перевищила б 3000 рядків і в ній треба було б використати 1000 окремих змінних, якщо не використовувати складені типи. Набагато зручніше помістити цей номер у лічильник і використати цикл for від 0 до 999, а потім від 999 до 0.

```
for (number=0; number<1000; number++)
        cin>>value[number];
for (number=999; number>=0; number--)
        cout<<value[number]<<endl;
```

Цей фрагмент коду буде працювати коректно в C++, якщо оголосити змінну value як одномірний масив. Це набір змінних одного і того ж типу в яких перша частина імені однакова, а друга частина імені значення індексу у квадратних дужках.

Оголошення одномірного масиву подібне на оголошення простого типу. Однак для масиву необхідно вказати його розмір. Це кількість компонентів у квадратних дужках

```
int value[1000];
```

Елементи масиву можуть мати практично будь-який тип. Кількість елементів у масиві має бути більше нуля. Якщо розмір масиву дорівнює n, то діапазон кількості значень від 0 до n-1.

```
float angle[4];
int testScore[10];
```

Щоб отримати доступ до окремих компонентів масиву використовують інструкцію, що називається індексною. Вона розміщується у квадратних дужках. Ця інструкція визначає до якого компонента здійснюється доступ. Це може бути константа, ім'я змінної або складна комбінація із змінних, операторів і викликів функцій. Результат дії цієї інструкції є цілим числом, тобто одним із типів: char, short, int, long, або enum.

Для присвоєння значень елементам масиву використовується синтаксис, що наведений у прикладі

```
angle[0]=4.93;
angle[1]=-15.2;
```

З кожним елементом масиву можна працювати так само, як і з простою змінною. Можна присвоїти йому значення, як показано вище, записати в нього значення

```
cin>>angle[2];
```

вивести вміст цього елемента

```
cout<<angle[2];
```

використовувати його в арифметичному виразі

```
x=6.8*angle[2]+7.5;
```

С++ не здійснює перевірку значень індексів на вихід за межі масиву. Це має робити програміст. Індекси, що виходять за межі масиву мають значення менше нуля або більше розміру масиву мінус 1.

При обробці масивів часто використовують цикл for для переходу від одного елемента до іншого. Наведемо приклад циклу для обнулення масиву alpha, що складається зі 100 елементів.

```
for (i=0; i<100; i++)
        alpha[i]=0.0;
```

Масиви можна задати за оголошення. У цьому випадку треба вказати початкові значення елементів масиву, розділяючи їх комами

```
int age[5]={23, 10, 16, 37, 12};
```

Якщо значень менше, ніж розмір масиву, то значення, котрих не вистачає, будуть дорівнювати нулю. Більша кількість значень за розмір масиву викликає синтаксичну помилку.

Як відомо, у С++ прості змінні завжди передаються за значенням. Щоб передати просту змінну за посиланням, треба до типу даних дописати символ амперсанда (&) у списку формальних параметрів функції.

У С++ неможливо передати масиви за значенням. Масиви завжди передають за посиланням. Коли масив передається як параметр функції, то в цьому випадку повертається у функцію адреса першого елемента масиву в пам'яті комп'ютера. Після цього функція знає де розміщений фактичний масив і може звертатися до будь-якого з його елементів. Наведемо приклад функції, що обнулює одномірний масив типу float будь-якого розміру.

```
void ZeroOut (/*out*/float arr[],/*in*/int numElements)
{
        int i;
        for (i=0; i<numElements; i++)
                arr[i]=0.0;
}
```

Найбільш поширеною помилкою при роботі з масивами є передача у функцію елемента масиву, коли необхідно передавати весь масив.

```
void Function(float[], int); //прототип функції
...
int main()
{
        float array[50];
        ...
        Function(array, 50);
        ...
}
```

Виклик функції

Function(array[50], 50);

призведе до помилки. Однак, найнебезпечніше в цьому прикладі те, що елемента масиву за номером 50 взагалі не існує. Всі елементи масиву array лежать в діапазоні від 0 до 49.

Найчастіше виконуються три види обробки масивів: використання частини оголошеного масиву, використання двох або трьох масивів і використання значень індексів, що мають специфічне значення в рамках задачі.

Розмір масиву встановлюється на етапі компіляції. В міру запису даних у масив, ми збільшуємо лічильник заповнених елементів. Під час обробки він використовується для роботи з корисними елементами. Решта не будуть розглянуті. Наприклад, якщо в класі 25 студентів, то програма для аналізу оцінок за тест буде використовувати масив із 25 елементів. Однак деякі студенти будуть відсутні в день тестування, тому буде підрахована фактична кількість оцінок. Саме ця кількість, а не значення 25, буде використана під час подальшої обробки масиву. Фактична кількість значень у масиві часто називається його довжиною. Якщо довжина масиву менша за його оголошений розмір, то саме вона повинна передаватися у функцію в якості параметра. Наприклад:

```
void Print (/*in*/) const char grade[ ],//Масив для 25 студентів
        /*in*/ int length) //Фактична кількість оцінок у масиві
```

У багатьох задачах потрібно об'єднати декілька частин інформації. Можна створити масив типу int (або long) для номерів і масив типу char для оцінок студентів. Після цього до значень масивів можна звертатися паралельно. Кожний номер відповідає певній оцінці, бо їх позиції в масивах співпадають, тобто мають рівні індекси. В деяких задачах індекс є не тільки номером позиції. Його значення має семантичний зміст. Це означає, що крім чисел і назви можуть ставитися у відповідність.

Під час компіляції програмного коду для статично оголошених масивів виділяється пам'ять. Для ефективного використання пам'яті призначене динамічне оголошення масивів, а саме

<тип вказівника> *<назва>=new <тип змінної>[<кількість>];

Після виконання цієї команди буде виділена неперервна ділянка пам'яті обсягом sizeof(тип змінної)*<кількість>;

і назва масиву буде вказувати на початок цієї ділянки.

З динамічною змінною можна виконувати операції, що дозволені для даних відповідного базового типу.

Після опрацювання масиву звільнити пам'ять можна за допомогою команди

delete[]<назва вказівника на масив даних>;

Під час звільнення пам'яті розмір масиву вказувати не потрібно.

За допомогою динамічних змінних можна розв'язати завдання почергового опрацювання однією програмою деякої кількості великих масивів (якщо всі масиви неможливо одночасно ввести в пам'ять).

Завдання для роботи

1. Нехай прибуток фірми за k-й рік обчислюється за формулою $y_k=100f_{i+9}(k)$ умовних одиниць, де k=2000, 2001, ..., 2010; i – номер варіанта. Якщо $y_k>0$, то вважатимемо, що фірма певного року мала прибуток, а у випадку $y_k< 0$ – збитки. Вивести на екран таблицю: номер року, величина прибутку.

Розглянути фінансову діяльність фірми протягом десяти років. Виконати додатково завдання вашого варіанта, наведене нижче. Вивести повідомлення, якщо

шуканих даних немає, наприклад, якщо збитків чи прибутків не було взагалі. Відповідні функції до роботи взяти із лабораторної роботи № 15

1. Обчислити суму прибутків фірми. Визначити максимальний збиток (якщо збитки були).
2. Обчислити суму збитків. У якому році збиток був максимальний?
3. Обчислити суми прибутків і збитків фірми та їх різницю. Коли прибуток був максимальний?
4. Скільки років поспіль прибутків було менше, ніж 1000, але більше, ніж 500 у.о? Коли фірма зазнала найбільших збитків?
5. Обчислити суму збитків. У якому році прибуток був найбільший?
6. Обчислити суму прибутків у межах $0<y_k<710$ (в у.о.). У якому році фірма зазнала найбільших збитків?
7. Скільки років прибутки були в межах від 200 до 700 у.о? Які це були роки?
8. Обчислити суму всіх збитків. У якому році збиток був найбільший? Який це був збиток?
9. Обчислити суму тих збитків, для яких справджуються умови $y_k<-650$ або $y_k>-150$ (в у.о.). Визначити найбільший прибуток.
10. Визначити суми прибутків і збитків. Скільки років фірма була прибутковою?
11. Обчислити суму прибутків, що були у межах $230<y_k<8500$ (в у.о.). Скільки років фірма мала такі прибутки?
12. Обчислити суму збитків, що були у межах $-750<y_k<200$ (в у.о.). Коли дохід був мінімальний?
13. Обчислити суму прибутків і збитків за перші сім років роботи та їх різницю. Визначити максимальний прибуток за цей період.
14. Обчислити суми прибутків, що були в межах $y_k<170$ або $y_k>620$ (в у.о.). Скільки років фірма мала такі прибутки?
15. Обчислити суму збитків і визначити, скільки років фірма була збитковою? У якому році збиток був максимальний?
16. Визначити найбільший збиток. У якому році фірма мала найбільший прибуток?
17. У які роки фірма мала найбільші прибуток і збиток?
18. Обчислити суму збитків. Чи був хоч раз нульовий баланс?
19. Обчислити суми прибутків і збитків фірми та їх різницю. Визначити максимальний збиток фірми.
20. Обчислити суму збитків, для яких справджується умова $y_k<-590$ або $y_k>-330$ (в у.о.). Визначити найбільший прибуток і в якому році він був отриманий?
21. Обчислити суму збитків фірми. У якому році прибуток був найменший? Визначити його величину.
22. Обчислити середні арифметичні всіх прибутків і збитків.
23. Обчислити суми прибутків і збитків за перші п'ять років роботи. Скільки років протягом цього періоду фірма мала прибутки?
24. Обчислити суму прибутків, які були в межах $315<y_k<958$ (в у.о.). У якому році збитки були найбільші?

2. Утворити і вивести масив у з елементами $y_k=f_{i+10}(k)$, де і – номер варіанта, k=1, 2, ...,7. Виконати завдання вашого варіанта. У разі відсутності шуканих даних вивести про це повідомлення.

1. Чи третій додатний елемент є останнім у масиві?
2. Вивести номери двох найбільших елементів. Обчислити їх суму.
3. Чи є два елементи серед від'ємних із максимальним значенням?
4. Максимальний елемент поміняти місцями з четвертим, що задовольняє умову $y_k>1$.
5. Третій додатний елемент замінити максимальним.
6. Визначити номер п'ятого від'ємного елемента.
7. Обчислити добуток перших трьох додатних елементів та визначити їхні номери.
8. Обчислити суму другого додатного та третього елементів.
9. Другий від'ємний елемент поміняти місцями із третім додатним.
10. Утворити масив елементів, значення яких є між значеннями третього та максимального елементів заданого масиву.
11. Вивести добуток номерів двох найменших елементів серед додатних.
12. Визначити суму номерів другого та третього від'ємного елементів.
13. Перший додатний елемент поміняти місцями з максимальним.
14. Знайти суму третього та шостого додатних елементів.
15. Другий від'ємний елемент замінити мінімальним.
16. Скільки є елементів з мінімальним значенням серед додатних?
17. Усі додатні елементи масиву, крім максимального, занести в інший масив.
18. Обчислити суму перших чотирьох від'ємних елементів.
19. Вивести номер передостаннього додатного елемента.
20. Елементи масиву після другого від'ємного занести в інший масив.
21. Знайти добуток другого та четвертого елементів, більших, ніж 3.
22. Максимальний елемент поміняти місцями з другим нульовим.
23. Останній від'ємний елемент замінити найбільшим.
24. Обчислити добуток другого від’ємного та п'ятого елементів.

3. Масиви із різною кількістю елементів. У підрозділі Y є 15 співробітників, а в G – 20. Протягом місяця вони відпрацювали певну кількість днів, яка задана як випадкове число зі значенням від 0 до 31. Денна оплата праці d у.о. Податкова ставка 20%. Утворити масиви y, g, вивести значення їхніх елементів на екран та у файл. Виконати завдання пошуку даних для кожного підрозділу. Вивести повідомлення, якщо шуканих даних немає.

1. Скільки осіб працювали у кожному підрозділі більше 15 днів?
2. Хто найменше заробив у кожному підрозділі?
3. Кому нараховано більше, ніж 100 у.о., у кожному підрозділі?
4. Скільки людино-днів було відпрацьовано у кожному підрозділі?
5. Який середній заробіток у кожному підрозділі?
6. Скільки осіб отримали більше, ніж 50, і менше, ніж 120 у.о.?
7. Скільки осіб працювали менше, ніж 10 днів?
8. Яка сума податку була сплачена у кожному підрозділі?
9. Хто сплатив найбільший податок у кожному підрозділі?
10. У скількох осіб податок перевищив 20 у.о.?
11. Який середній податок був у кожному підрозділі?
12. У якому підрозділі більший середній заробіток?
13. Хто сплатив найменший податок у кожному підрозділі?
14. Скільки осіб працювали лише один день у кожному підрозділі?

15. У скількох осіб заробіток вищий за середній?
16. У якому підрозділі менший середній заробіток?
17. У скількох осіб заробіток відхиляється від середнього менше, ніж на 10%?
18. У якому підрозділі був зафіксований найбільший заробіток?
19. Скільки осіб працювали більше, ні ж 5, і менше, ніж 12 днів?
20. Який середній заробіток перших п'яти осіб?
21. У скількох осіб заробіток був менший за середній?
22. Який середній заробіток останніх чотирьох осіб?
23. У якому підрозділі було відпрацьовано більшу кількість людино-днів?
24. Хто заробив більше, ніж 100, і менше, ніж 200 у.о.?
25. Скільки осіб працювали два, три або чотири дні?

Приклад генерації випадкових чисел в діапазоні від 10 до 30:

```
#include <iostream>
#include <stdlib.h>
#include <time.h>
using namespace std;
int main()
{
	double m;
	srand(time(NULL));
	for(int i = 0; i < 10; i++)
	{
		m = 10 + rand() % 21;
		cout << m << endl;
	}
	system("pause");
	return 0;
}
```

Завдання підвищеної складності. Яка кількість відпрацьованих днів найчастіше була зафіксована у кожному під розділі?

4. Нехай елементи одномірного масиву обчислюються за допомогою функції $yk=f_{i+5}(x)$, де i – номер варіанта, x=-2...2 з кроком 0,1.

1. Впорядкувати масив за зростанням і результати впорядкування вивести в інший масив. Невпорядкований і впорядкований масиви вивести на екран.

2. Впорядкувати масив за спаданням і результати впорядкування вивести в інший масив. Невпорядкований і впорядкований масиви вивести на екран.

Контрольні запитання

1. Як оголошується одномірний масив?
2. Як ввести дані в одномірний масив?
3. Як вивести дані із одномірного масиву?
4. Як здійснюється нумерація елементів одномірного масиву?
5. Які типи даних можуть бути елементами масиву?
6. Який зміст має номер елемента масиву?

Лабораторна робота № 22-26
ТЕМА: ЗАСТОСУВАННЯ ОДНОМІРНИХ МАСИВІВ

МЕТА РОБОТИ: навчитися використовувати одномірні масиви для обробки рядків, використовувати структури та об’єднання для побудови програм.

Теоретичні відомості

Рядкова константа – це послідовність символів, поміщених у подвійні лапки. Наприклад, ``Привіт''. Рядкова константа зберігається як масив типу char, в якому кожний елемент призначений для зберігання одного символу. В кінці цього масиву є один ``зайвий'' елемент для збереження нульового символу `\0'. Рядок (string) є єдиним видом масиву в С++ для якого існує окрема константа. Рядок – набір символів, що інтерпретується як окремий елемент даних. Мовою С++ – це послідовність символів, що закінчується нулем і зберігається в масиві типу char. Виходячи із цього випливає, що можна створювати рядкові змінні. Для цього необхідно явно оголосити масив типу char і зберегти в ньому будь-які символи, завершуючи їх послідовність нульовим символом

```
char myStr [8];
```

Стандартна бібліотека С++ надає декілька функцій для роботи із рядками. Одна з них має ім'я strlen(). Вона визначає довжину будь-якого рядка без врахування останнього нульового символу `\0'. Наведемо приклад програми, яка буде визначати довжину рядка без функції strlen().

```
int StrLength (/*in*/ const char str[])
{
        int i=0; //Індексна змінна
        while(str[i]!='\0')
                i++;
        return i;
}
```

В цьому прикладі дії функції strlen() виконує функція StrLength(). Над рядком (як типом даних) можна виконати такі операції:

Створити та ініціалізувати рядок.

Вивести рядок.

Визначити довжину рядка.

Порівняти два рядки.

Копіювати один рядок в інший.

Щоб ініціалізувати рядкову змінну за оголошення, можна використати такий прийом

```
char message[8]={`w',`h',`o',`o',`p',`s',`!',`\0'};
```

тобто так само як і для масиву. Однак у С++ є більш зручний спосіб ініціалізації рядка:

```
char message[8]="Whoops!";
const char text[8]="Whoops!";
```

Цей короткий запис справедливий тільки для рядків. У рядків як і в масивів є можливість опускати розмір, якщо він ініціалізується за оголошення

```
char promptMsg[]="введіть додатне число";
```

Для виводу рядка використовується вже відомий оператор “<<”. Він виводить символи масиву до нульового символу, що завжди закінчує рядковий масив.

```
char msg[8]="Welcome";
...
```

```
cout<<msg;
```

При вводі даних у рядкову змінну оператор “>>” буде пропускати всі попередні недруковні символи, а саме пробіли, символи переходу на новий рядок. Після цього він читає всі послідовні символи в масив поки не зустрінеться кінцевий недруковний символ. В кінці він додає нульовий символ до рядка.

```
char firstName[31]; //Місце для 30 символів і `\0'
char lastName[31];
cin>>firstName>>lastName;
```

Різновид функції get може використовуватися для вводу рядкових даних. У якості параметрів функції виступають рядкова змінна і інструкція типу int

```
cin.get(myStr, charCount+1);
```

Функція get не пропускає ведучі недруковні символи. Вона продовжує ввід поки не зустрінеться символ переходу на новий рядок `\n'. Після цього у кінець рядка додається нульовий символ, а в рядковій змінній зберігається charCount символів.

```
char oneLine[81]; //місце для 80 символів і `\0'
...
cin.get(oneLine,81);
```

Функція ignore є корисною у поєднанні з get. Припустимо користувач вводить у програму довгий рядок, а для виконання програми потрібні тільки перших чотири символи. В цьому випадку функцію ignore разом із функцією get можна використати як показано у прикладі

```
char response[5]; //Місце для 4-ох символів і `\0'
cin.get(response,5); //Ввести принаймні 4 символи
cin.ignore(100, `\n');//Пропустити решту символів,
```

Рядки можна опрацьовувати посимвольно за допомогою вказівників або назви масиву.

Приклад опрацювання рядків за допомогою вказівників

```
char fraza[11];
char slovo[]={`U',`n',`i',`v',`e',`r',`s',`i',`t',`y',`\0'};
for (int n=0; n<11; n++)
        *(fraza+n)=*(slovo+n);
cout<<fraza;
```

В цьому прикладі змінній fraza присвоюється значення «University» і фраза виводиться на екран. Інакше це можна зробити як показано у прикладі опрацювання рядків за допомогою назви масиву

```
char fraza[11];
char slovo[]={`U',`n',`i',`v',`e',`r',`s',`i',`t',`y',`\0'};
for (int n=0; n<11; n++)
        fraza[n]=slovo[n];
cout<<fraza;
```

Посимвольно вводити чи виводити елементи рядка можна за допомогою команд циклу for або while.

```
for (int n=0; n<11; n++)
cin>>*(fraza + n);
```

У кінці рядка необхідно ввести нульовий символ, тобто

```
*(fraza+n+1)=`\0';
```

Для опрацювання масивів символів у мові С++ є стандартні функції, які описані у файлі заголовків string.h (див. лекцію).

Відомий вираз typedef дає можливість задати додаткове ім'я для вже існуючого типу даних. Виявляється, що його можна використовувати і для оголошення імені рядкового типу:

```
typedef char String20[21];
...
String20 firstName;
String20 lastName;
cin>>firstName>>lastName;
if (strcmp(lastName, "Jones")>0)
...
```

Структура – це спеціальний тип даних, який утворює користувач для опрацювання інформації про об'єкти з деякої предметної області. Така інформація може складатися з даних різних типів. Структура складається з набору полів – даних різних типів. Спосіб її оголошення наведений у прикладі

```
struct <назава типу структури>
{
        <тип поля 1> <назва поля 1>;
        ...
        <тип поля n><назва поля n>;
};
```

Опис структури обов'язково закінчується символом «;».

Під час оголошення структури можна задавати обсяг пам'яті, який будуть використовувати змінні певного поля. Для цього після назви поля ставиться знак двокрапки та зазначається ціле число або стала, які вимірюються у байтах. Такі поля називають бітовими. Визначити обсяг, який займає ціла структура, можна виразом

```
sizeof(<назва типу структури>);
```

Коли в програмі описана структура, то змінні або вказівники цього типу оголошуються як

```
<назва типу структури> <список змінних і вказівників>;
```

Наприклад, оголосимо змінні gr1 і gr2, вказівник p для структури grupa

```
grupa gr1, gr2;
grupa *p;
```

Оголосити змінні типу структура можна іншим способом.

```
struct grupa
{
        char name[20], sname[20];
        float seredniy_bal;
} gr1, gr2, *p;
```

При такому оголошенні змінних структури, назву структури можна не зазначати.

Структура може містити поля, які є також структурою.

Змінні типу структура можна ініціалізувати відразу за оголошення. Оголосимо та ініціалізуємо змінну gr.

```
struct grupa
{
        char name[20], sname[20];
        float seredniy_bal;
} gr={"Василюк", "Василь", 4.7};
```

У цьому випадку поле name змінної gr має значення ``Василюк'', поле sname – значення ``Василь'', а поле seredniy_bal – 4.7.

Доступ до конкретного поля змінної типу структура дає ім'я вигляду

<назва змінної>.<назва поля>;

grB.birthday.day=22;

Можна також створювати вказівники на структури. Доступ до полів вказівника на структуру здійснюється дещо інакше, ніж до полів відповідної змінної, а саме

<назва вказівника> -> <назва поля>;

```
grupa *p; //Вказівник p вказує
p=&grA; //на адресу змінної grA
//заповнюємо значення полів seredniy_bal
p->seredniy_bal=4.6;
p->birthday.year=1974;
```

Компонентом структури може також бути компонент, тип якого належить іншій структурі. Такі структури називаються ієрархічними.

Об'єднання у мові С++ нагадує опис структури. Однак їхнє призначення зовсім різне. Об'єднання призначені для зберігання (послідовного, не одночасного) в деякій ділянці оперативної пам'яті комп'ютера даних різних типів. Потреба в цьому виникає, наприклад, під час створення таблиці з даними різних типів. Приклад опису об'єднання наведений нижче

```
union <назва типу об'єднання>
{
        <тип поля 1> <назва поля 1>;
        ...
        <тип поля n> <назва поля n>;
};
```

Об'єднання використовуються для економії пам'яті, оскільки декілька значень можуть використовувати одну й ту ж область пам'яті в різні моменти часу. Усі правила опису структур діють і для опису об'єднань. При використанні об'єднань компілятором резервується місце в пам'яті для розміщення найбільшого елемента об'єднання. Найчастіше об'єднання включають у склад структур. Крім цього їх використовують для перетворення даних одного типу в інший. Доступ до елементів об'єднання виконується за допомогою операції крапка «.». На відміну від структур, змінна типу об'єднання може бути ініціалізована тільки значенням першого оголошеного члена.

Щоб зробити програму яка працює із файлами більш гнучкою, можна вказувати їй ім'я файлу даних у процесі роботи. Загальною методикою є вивід запрошення для вводу імені файла та отримання відповіді від користувача у вигляді рядкової змінної. Далі ця змінна передається в якості параметра функції open.

За допомогою функцій get і ignore, можна вводити ім'я файла в процесі виконання програми.

```
ifstream inFile; //вхідний файл для аналізу
char fileName[51];//максимальна довжина 50 символів
cout<<"введіть ім'я файла:";
cin.get(fileName,51);//Читаємо принаймні 50 символів
cin.ignore(100,'\n');//Пропускаємо решту символів
inFile.open(fileName);
if (!inFile)
{
        cout<<"не можу відкрити файл"<<endl;
        return 1;
}
```

...

Завдання для роботи

1. Написати програму, яка б визначала довжину довільної введеної фрази. Визначити кількість букв “а”, що зустрічаються в ній.
2. Написати програму, яка б дозволяла ввести фразу «Скоро прийдуть Великодні свята» з клавіатури, знайти довжину третього слова та вивести його на екран в прямому і зворотньому напрямках.
3. Даний рядок. Групу символів, що розділені пробілами (одним або кількома) та не містять пробілів всередині себе будемо називати словами. Вивести послідовність слів у зворотному порядку. Програма повинна запросити і ввести з клавіатури рядок і при необхідності додаткові дані. Програма повинна складатися не менш, ніж з двох функцій: у головної функції організується введення початкових даних і виведення результатів, інша функція виконує безпосередньо завдання відповідно з варіантом. Вихідний рядок і буфер для збереження результуючого рядка функція повинна отримати у списку аргументів (ніякого консольного вводу / виводу в ній бути не повинно). Увага: користуватися в цій роботі функціями з модуля <string.h> заборонено!

Для вирішення поставленого завдання зручно реалізувати три функції. Перша функція LenStr (char const Str []) знаходить довжину вхідного рядка.

Друга функція Inverse (char const StrIn [], char StrOut []) виконує необхідне перетворення. Тобто на основі вхідного рядка StrIn формує результуючий рядок StrOut.

Третя функція CopyWord (char const StrIn [], char StrOut [], int j, int k) копіює слово з рядка StrIn слово починається з позиції j в рядок StrOut починаючи з позиції k. Основна ідея алгоритму полягає в тому щоб, переміщаючись в циклі від кінця рядка до початку копіювати слова, що зустрічаються по порядку в результуючий рядок.

Можна придумати свій алгоритм.

4. Скласти програму для кодування деякого тексту до 50 символів, замінюючи кожну літеру на п'яту після неї літеру з алфавіту. Під алфавітом розуміти таблицю кодів ASCII. Зберегти результат виконання програми у файл. Під час написання програмного коду не використовувати файл заголовків string.h.
5. Використовуючи результат попередньої програми, скласти програму розкодування закодованого тексту.
6. Задачі на пошук необхідних значень зустрічаються досить часто у повсякденному житті, тому вміти розв'язувати такі задачі є актуальним завданням. Використовуючи цикл for заповнити одномірний масив дійсних значень, що обчислюються функцією $y[k]=f_{i+3}(k)$, де i - номер варіанту, а k=0, 1, 2, …, 10. Після заповнення масиву значень, використовуючи функцію із попередньої лабораторної роботи, впорядкувати масив за зростанням. Ввести елемент пошуку в одномірному масиві. Застосувати функцію пошуку, для відшукання введеного елемента у відсортованому масиві. Додатковим параметром функції пошуку використати логічну змінну, яка буде отримувати істину, якщо елемент знайдений і хибу - якщо ні. Функція пошуку повинна виводити позицію шуканого елемента у відсортованому масиві, якщо він знайдений і повідомлення про відсутність його у списку, якщо не знайдений. Проробити попереднє завдання, але заповнити масив цілими значеннями, застосувавши явне зведення до цілого типу int при розрахунку значень елементів масиву. Треба звернути увагу на те, що порівняння цілих значень і

дійсних відрізняється. При порівнянні дійсних значень, порівнюється різниця між шуканим значенням і списковим з деякою допустимою величиною точності в рамках даної задачі Δε. Відповідну функцію вибрати із таблиці, що представлена в роботі № 15.

7. Використовуючи цикл for заповнити одномірний масив дійсних значень, елементи якого обчислюються функцією $y[k]=f_{i+7}(k)$, де i - номер варіанту, а k=0, 1, 2, …, 20. Після заповнення масиву значеннями, використовуючи функцію із попередньої задачі, впорядкувати масив за зростанням. Ввести елемент вставки в одномірний масив. Застосувати функцію пошуку, для відшукання введеного елемента у відсортованому масиві. Функція вставки елемента в масив повинна виконувати вставку елемента в кінець масиву та застосовувати повторне сортування масиву, якщо введений елемент не знайдений у списку, або виводити повідомлення про його наявність у списку. Останні дії повинні залежати від вмісту логічної змінної функції пошуку, яка є додатковим параметром функції вставки. Проробити попереднє завдання, але заповнити масив цілими значеннями, застосувавши явне зведення до цілого типу int при розрахунку значень елементів масиву. Треба звернути увагу на те, що порівняння цілих значень і дійсних відрізняється. При порівнянні дійсних значень, порівнюється різниця між шуканим значенням і списковим з деякою допустимою величиною точності в рамках даної задачі Δε.
8. Придумати й описати структуру та скласти програму для створення масиву з шести-семи елементів цієї структури й опрацювання відповідних даних згідно з деяким сюжетом. У сюжеті задати і описати критерій пошуку деякої інформації. Прикладами структур можуть бути: дані про студентів (прізвище, ім'я, оцінки з певних предметів), адреси та телефони друзів, характеристики комп'ютерів, автомобілів, аудіотехніки, довідки про країни (назва, кількість населення, площа), інформація про бібліотечне наповнення (автор, назва, видавництво, рік видання, кількість сторінок, ціна) тощо. Приклад сюжету: створити і вивести на екран масив записів про автомобілі на складі (назва моделі, рік випуску, ціна, колір), а також знайти у масиві і вивести на екран назви моделей червоного кольору, які випускалися в 2001 році.

Контрольні запитання

1. Як оголошуються рядки в С++?
2. Що таке вказівники?
3. Як оголошується структура?
4. Способи присвоєння значень елементам структури?
5. Синтаксис оголошення об’єднання?
6. Яка різниця між структурами і об’єднаннями?
7. Які є способи обробки рядків?

Лабораторна робота № 27-31
ТЕМА: ПОБУДОВА ПРОГРАМ ІЗ ВИКОРИСТАННЯМ ДВОМІРНИХ МАСИВІВ

МЕТА РОБОТИ: навчитися опрацьовувати двомірні масиви та використовувати двомірні масиви для обробки матриць.

Теоретичні відомості

У багатьох задачах взаємозв'язок між елементами даних є більш складним, ніж простий список. Двомірний масив використовується для представлення таблиці, що містить рядки і колонки. При цьому всі елементи мають один і той же тип даних. Щоб звернутися до окремого елемента у двомірному масиві, треба вказати його позицію в рядку і колонці.

Двомірний масив оголошується так само, як і одномірний, з тією різницею, що розміри мають вказуватись для двох вимірів:

```
const int Num_Rows=100;
const int Num_Cols=9;
...
float alpha[Num_Rows][Num_Cols];
                1-й вимір   2-й вимір
```

У цьому випадку оголошується двомірний масив alpha, елементи якого мають тип float.

Для звернення до окремого елемента масиву alpha використовуються дві інструкції (по одній для кожного виміру)

```
alpha [0] [5]=36.4;
```

Дані в рядках чи колонках можна представити ще одним способом:

```
enum DayType{MONDAY,TUESDAY,…,SUNDAY};
int hiTemp[52][7];
```

У цьому прикладі оголошений двомірний масив, але в якості індексу колонки можна використовувати інструкцію типу DayType (назви днів). Наприклад, hiTemp[2][SUNDAY] це той самий елемент, що і hiTemp[2][6]. Можна використовувати назви в індексах і по рядках і по колонках.

```
enum Colors{RED,ORANGE,YELLOW,GREEN,BLUE,INDIGO,VIOLET};
enum Brands{FORD,TOYOTA,HYUNDAY,YAGUAR,CITROEN,BMW,FIAT,SAAB};
const int NUM_COLORS=7;
const int NUM_BRANDS=8;
float crashRating[NUM_COLORS][NUM_BRANDS];
...
crashRating[BLUE][YAGUAR]=0.83;
```

Обробка даниху двомірному масиві означає звернення до масиву одним з чотирьох способів: випадковим чином, до рядків, до колонок, до цілого масиву. Кожний із способів може включати обробку підмасивів.

Випадковий доступ – коли вводиться довільна, випадкова комбінація координат. Доступ до колонок або рядків – коли над колонками чи рядками виконується певна дія. Доступ до цілого масиву – коли хочемо знайти середнє значення всіх величин масиву. В цьому випадку сумування вже йде по рядках і колонках, а не окремо. Звідси випливають способи обробки масивів:

1. Додавання рядків.
2. Додавання колонок.

3. Ініціалізація всіх елементів таблиці значенням 0 (або іншим).
4. Друк таблиці.

У прикладі нижче означимо декілька величин, які необхідні для оголошення двомірного масиву.

```
const int NUM_ROWS=50;//Кількість рядків
const int NUM_COLS=50;//Кількість колонок
int table[NUM_ROWS][NUM_COLS]; //Двомірний масив
int rowLength; //Інтервал значень від 0 до rowLength–1
int row; //Індекс для рядка
int colLength; //Інтервал значень від 0 до colLength–1
int col; //Індекс для колонок
int total; //Змінна для додавання
```

Припустимо, що потрібно додати всі елементи рядка з номером 3 у масиві table і вивести результат.

```
total=0;
for (col=0; col<NUM_COLS; col++)
        total=total+table[3][col];
cout<<"Сума рядка:"<<total<<endl;
```

Якщо треба підсумувати елементи по двох рядках, то це можна зробити за допомогою вкладених циклів.

```
total=0;
for (row=2; row<4; row++)
{
        for (col=0; col<NUM_COLS; col++)
                total=total+table[row][col];
}
cout<<"сума рядків 2 і 3:"<<total<<endl;
```

Щоб організувати додавання за колонками, то в цьому випадку потрібно у попередньому фрагменті програми замінити місцями колонки і рядки.

```
total=0;
for (col=2; col<colLength; col++)
{
        for (row=0; row<rowLength; row++)
                total=total+table[row][col];
}
cout<<"сума колонок:"<<total<<endl;
```

Двомірний масив можна ініціалізувати за оголошення або з використанням виразів присвоювання. Якщо масив невеликий, то простіше ініціалізувати його за оголошення. Наприклад, щоб ініціалізувати таблицю на 2 рядки і 3 колонки числами:

```
int table[2][3]=
{
        {14,3,-5},
        {0,46,7}
};
```

Непрактично ініціалізувати таблицю за оголошення, якщо вона велика, наприклад, 100x100. Якщо її значення різні, то краще дані зберігати у файлі і вводити в процесі роботи програми. Якщо ж усі значення однакові, то, як правило, використовуються вкладені цикли for і вирази присвоювання. Наведемо фрагмент коду для обнулення таблиці із NUM_ROWS рядків і NUM_COLS колонок.

```
for (row=0; row<NUM_ROWS; row++)
{
        for (col=0; col<NUM_COLS; col++)
                table [row][col]=0;
}
```

Як відомо, при використанні одномірних масивів у якості формальних параметрів функції розмір масиву, як правило, не вказується

```
void SomeFune(/*inout*/ float list[ ], /*in*/ int size)
```

Якщо вказати розмір масиву у квадратних дужках, то він буде проігнорований. У функцію передається базова адреса фактичного параметра, тобто адреса в пам'яті першого елемента масиву. Функція працює з фактичним параметром будь-якого розміру. Оскільки вона не знає розміру фактичного масиву, то їй передається значення розміру у вигляді додаткового параметра (як у прикладі з функцією SomeFune, що вище) або використовується константа (якщо функція завжди працює з масивами одного і того ж розміру).

Коли в якості параметра передається двомірний масив, функція також отримує базову адресу фактичного масиву. Не можна опускати розмір зразу двох вимірів. Можна не вказувати розмір першого виміру (кількість рядків), але обов'язково повинна вказуватися кількість колонок. Тобто, оголошення формального параметра має завжди містити кількість колонок

```
void AnotherFune (/*inout*/ int arr[][4]);
```

Функція AnotherFune працює з двомірним масивом, що має будь-яку кількість рядків, але обов'язково чотири колонки. Щоб уникнути помилок, які пов'язані з не співпаданням розмірів формальних і фактичних масивів, корисно використовувати вираз typedef для встановлення типу двомірного масиву. Після цього можна оголосити і фактичні і формальні параметри цього типу.

```
const int NUM_ROWS=10;
const int NUM_COLS=20;
typedef int TableType[NUM_ROWS][NUM_COLS];
```

Запишемо у загальному вигляді функцію, яка ініціалізує всі елементи масиву вказаним значенням.

```
void Initialize(/*out*/TableType table, //Масив, який треба ініціалізувати
                /*in*/ int initVal //Значення для ініціалізації.
                //Дана функція ініціалізує кожний елемент
                //таблиці значенням initVal
{
        int row;
        int col;
        for (row=0; row<NUM_ROWS; row++)
                for (col=0; col<NUM_COLS; col++)
                        table[row][col]=initVal;
}
```

Після цього у коді програми можна оголосити й ініціалізувати один або більше масивів типу TableType за допомогою виклику функції Initialize.

```
TableType delta;
TableType gamma;
Initialize(delta,0);
Initialize(gamma,-1);
```

Двомірний масив можна уявити собі як масив, що складається з масивів. Це означає, що елементи двомірного масиву не обов'язково повинні бути неподільними. Наприклад, можна оголосити масив з імен студентів:

```
const int MAX_LENGTH=200;
typedef char String10[11];//Місце для 10 символів і символу `\0'
String10 student[MAX_LENGTH];//Масив із імен студентів
```

За такого оголошення елементами масиву student є одномірні масиви типу String10. Тобто масив student має два виміри. student[57] – це звернення до імені студента з номером 57. Якщо вказати два індекси student[57][0] – це звернення до першої літери імені студента з номером 57. Масив student можна оголосити і іншим способом

```
char student[MAX_LENGTH][11];
```

Два попередні способи оголошення двомірного масиву є рівноправними. Використання того чи іншого способу, буде залежати від вподобань самого програміста.

У мові С++ кількість вимірів масиву необмежена. Вимірів може бути стільки, скільки властивостей потрібно описати для кожного елемента масиву.

Завдання для роботи

I. Утворити масив з елементами $a_{kn}=nf_{i+11}(k)+\sin(k)f_{i+12}(n)$, де i – номер варіанта, k, n=1, 2, 3, 4. Вивести його на екран у вигляді таблиці (матриці). Виконати додатково завдання вашого варіанта.

1. Визначити індекси мінімального елемента масиву. Обчислити добуток його від'ємних елементів.
2. Обчислити кількість елементів масиву, для яких виконується нерівність $1<a_{kn}<6$.
3. Обчислити добуток значень тих елементів, для яких справджуються нерівності $a_{kn}<-1$ або $a_{kn}>1$.
4. Обчислити кількість додатних елементів та їхній добуток.
5. Обчислити суму квадратів елементів, значення яких більші, ніж 1.
6. Обчислити добуток квадратів тих елементів масиву, для яких виконується нерівність $|a_{kn}|<3$.
7. Обчислити кількість тих елементів масиву, для яких виконується нерівність $a_{kn}>3$, та суму елементів менших, ніж 9.
8. Обчислити добуток від'ємних елементів. Визначити індекси максимального елемента.
9. Обчислити суму діагональних елементів масиву та кількість від'ємних елементів.
10. Обчислити добуток тих елементів масиву, для яких виконується нерівність $2<a_{kn}<10$.
11. Визначити індекси максимального елемента масиву. Обчислити добуток елементів над головною діагоналлю.
12. Обчислити добуток елементів перших двох рядків.
13. Обчислити суму елементів масиву над головною діагоналлю. Визначити індекси мінімального елемента.
14. Обчислити суму від'ємних елементів. Знайти максимальний.
15. Обчислити добуток мінімального і максимального елементів масиву.
16. Визначити індекси мінімального і максимального елементів масиву.
17. Елементи масиву, що дорівнюють нулю, замінити на 1. Знайти суму елементів під головною діагоналлю.
18. Визначити кількість від'ємних і суму додатних елементів.
19. Обчислити добуток тих елементів, для яких виконуються не рівності $a_{kn}<-5$ або $a_{kn}>3$. Визначити індекси мінімального елемента.

20. Визначити індекси максимального та мінімального елементів масиву. Обчислити їхній добуток.
21. Обчислити добуток елементів над головною діагоналлю матриці та визначити їхню кількість.
22. Обчислити середнє арифметичне додатних елементів масиву.
23. Обчислити суму тих елементів масиву, для яких виконується нерівність $1<a_{kn}<5$. Знайти максимальний елемент.
24. Обчислити суму діагональних елементів матриці та кількість елементів, значення яких менші, ніж 3.
25. Обчислити добуток елементів під головною діагоналлю на суму елементів на головною діагоналлю.

II. Задача про вибори. Нехай шість населених пунктів (k) позначені номерами від 1 до 6, а п'ять кандидатів (n) – номерами від 1 до 5. Кількість голосів, набраних кандидатами у кожному пункті, визначається формулою a_{kn}=random(10i+50), де і – номер варіанта, а сума голосів, поданих за кандидата, визначає його рейтинг. Вивести на екран таблицю результатів голосування, де у рядках є дані з населених пунктів, а у стовпцях – дані щодо конкретних кандидатів. Визначити і вивести значення величин з додаткового завдання. Створити одновимірний масив із шуканими даними.

1. Які підсумкові результати кожного кандидата? (Підказка: утворити одновимірний масив із сум значень усіх стовпців таблиці).
2. Які номери населених пунктів, де кількість поданих голосів перевищила 150 (Підказка: утворити одновимірний масив із цих номерів)?
3. Хто з кандидатів набрав максимальну, а хто – мінімальну кількість голосів у четвертому населеному пункті?
4. Яка кількість голосів була подана за першого і третього кандидатів у всіх населених пунктах?
5. В яких населених пунктах другий і четвертий кандидати набрали максимальну кількість голосів?
6. Скільки виборців взяли участь у голосуванні у кожному населеному пункті ?
7. Хто з кандидатів набрав максимальну кількість голосів у другому населеному пункті?
8. В яких населених пунктах кількість опитаних більша деякого заданого числа n?
9. За кого з кандидатів подано кількість голосів, меншу від деякого заданого числа n?
10. В яких населених пунктах перший кандидат набрав максимальну кількість голосів?
11. Хто з кандидатів набрав найбільше голосів у другому і третьому населених пунктах?
12. В якому населеному пункті перший кандидат набрав мінімальну кількість голосів, а в якому – максимальну?
13. Хто з кандидатів має максимальний рейтинг?
14. В якому населеному пункті проголосувало найбільше людей?
15. У кого з-поміж другого, четвертого і п'ятого кандидатів най вищий рейтинг?
16. Хто набрав максимальну, а хто – мінімальну кількість голосів у першому населеному пункті?
17. У яких населених пунктах перший і п'ятий кандидат набрали більше, ніж 100 голосів?
18. Які номери населених пунктів, де кількість учасників виборів не перевищила 450?

19. У кого з кандидатів рейтинг більший від деякого заданого числа n?
20. В яких містах кількість виборців менша від деякого заданого числа?
21. Які кандидати набрали мінімальну кількість голосів у кожному із населених пунктів?
22. Які кандидати набрали максимальну і мінімальну кількість голосів в другому і п'ятому населених пунктах?
23. У кого з кандидатів найменший рейтинг?
24. У скількох кандидатів рейтинг перевищує деяке задане число n?
25. В яких населених пунктах третій кандидат набрав максимальну кількість голосів?

III. У файл a.txt записати засобами текстового редактора значення елементів цілочисельної матриці A розміром nxn, а у файл b.txt – значення елементів матриці B розміром nxn, де n=4. Обчислити матрицю C згідно з варіантом, використавши функції користувача. Результати обчислень занести у файл c.txt.

1. Обчислити $C=(AA^T)^2-2B$
2. Обчислити $C=2AB-B^4$
3. Обчислити $C=A^2B-3B^2$
4. Обчислити $C=(B\,A-5AB)^2$
5. Обчислити $C=(A+9B)A^T$
6. Обчислити $C=3(A^TB-B)\,B$
7. Обчислити $C=2AB+BA^3$
8. Обчислити $C=A^3-B^2$
9. Обчислити $C=(AB-A^2B)^3$
10. Обчислити $C=5A^3-8B^2$
11. Обчислити $C=(A^TB-3B^2)\,B$
12. Обчислити $C=(AA^T)^3-2B$
13. Обчислити $C=4\,B^TA^2-B^3$
14. Обчислити $C=ABA-2B^2$
15. Обчислити $C=(2A+B)\,B^T$
16. Обчислити $C=(2B\,A-A^4)^2$
17. Обчислити $C=8\,BAB-A\,B^4$
18. Обчислити $C=(A^T-7B^2)\,A$
19. Обчислити $C=A^2A^T-4B$
20. Обчислити $C=(6A^T-B)\,B^3$
21. Обчислити $C=(A^T)^3-7B^2$
22. Обчислити $C=(A^TB^4-5B)\,B$
23. Обчислити $C=(4B\,A-B^2)^3$
24. Обчислити $C=B(BT)^2-4A$
25. Обчислити $C=3\,BAB-2AB^2$

Переоформити кожну задачу із використанням функцій користувача, які отримують у якості параметрів двомірний масив.

Оформити звіт з кожної задачі, захистити і здати викладачу.

Контрольні запитання

1. Які є способи оголошення двомірних масивів?
2. Яких типів можуть бути багатомірні масиви?
3. Які операції обробки двомірних масивів ви знаєте?
4. Які цикли використовуються для опрацювання даних багатомірних масивів?
5. Які особливості передачі двомірних масивів у якості параметра функції?
6. Як опрацьовуються багатомірні масиви?

Лабораторна робота № 32
ТЕМА: КЛАС ЯК ТИП ДАНИХ. КОНСТРУКТОР

МЕТА РОБОТИ: навчитися оголошувати клас та об’єкти класу. Реалізовувати звернення до закритих елементів класу. Навчитися використовувати об’єкти С++ у якості змінних типу.

Теоретичні відомості

Головним інструментом мови програмування С++, який призначений для створення абстрактних типів даних є клас. Класи в С++ є наслідком розвитку поняття структури.

Класи дають можливість програмісту моделювати об'єкти, які володіють атрибутами. Ці атрибути представлені у вигляді даних. Також об'єктам властива певна поведінка або дія, що представлена у вигляді функцій. Функції здатні виконувати маніпуляції над даними. При означенні типів, що містять дані і функції в С++ використовується ключове слово class. Відразу після означення класу його ім'я може бути використане для оголошення об'єктів цього класу. Нижче показано просте означення класу Time.

```
class Time{
        public:
                Time();
                void setTime(int, int, int);
                void printMilitary();
                void printStandard();
        private:
                int hour; //0-23
                int minute; //0-59
                int second; //0-59
};
```

Означення класу Time починається з слова class. Тіло класу обмежене лівою і правою фігурними дужками. Означення класу завершується крапкою з комою. В означенні класу Time, є три цілочисельні елементи: hour, minute і second. Решта компонентів класу є новими. Мітки public і private називаються специфікаторами доступу до елементів. Усі дані і функції, що оголошені після специфікатора public (і до наступного такого специфікатора доступу) доступні всюди, де програма має доступ до будь-якого об'єкта класу Time. Всі дані і функції, що оголошені після специфікатора private (і до наступного такого специфікатора доступу), доступні тільки для функцій класу. Специфікатори доступу до елементів класу завжди закінчуються двокрапкою (:) і можуть багато разів з'являтися в означенні класу. В означенні класу після специфікатора public містяться прототипи чотирьох функцій: Time, setTime, printMilitary і printStandard. Ці функції називають відкритими, публічними функціями, або інтерфейсом класу. Вони використовуються клієнтами (користувачами) класу для маніпулювання його даними. Функція із тим же ім'ям, що і сам клас, називається конструктором цього класу. Конструктор – це спеціальна функція класу, яка ініціалізує дані об'єкта класу. Конструктор класу викликається автоматично при створенні об'єкта. Три цілочисельні елементи після специфікатора private доступні тільки для функцій класу і для друзів класу. Таким чином, до даних класу можуть мати доступ тільки чотири функції, прототипи яких з'являються в означенні класу, а також друзі класу.

Зазвичай дані класу перелічуються в розділі private, а функції – в розділі public. Однак можуть існувати функції з доступом private і дані з доступом public. Відразу після означення класу він може бути використаний як тип в оголошеннях, подібних наступним:

```
Time sunset, //об'єкт типу Time
arrayOfTimes[5],//масив об'єктів типу Time
*poiterToTime, //вказівник на об'єкт типу Time
&dinnerTime=sunset;//посилання на об'єкт типу Time
```

Ім'я класу є новим ідентифікатором типу. Об'єктів класу може бути багато, як і змінних типу int. В міру необхідності можна створювати нові класи. Це дозволяє вводити нові типи даних. Завдяки таким можливостям C++ є розширюваною мовою.

У прикладі нижче програма створює один об'єкт класу Time з ім'ям t. При створенні об'єкта автоматично викликається конструктор Time, який явно ініціалізує кожен закритий елемент даних значенням 0. Для контролю ініціалізації час виводиться у військовому і стандартному форматах. За допомогою функції setTime відбувається повторна ініціалізація значеннями даних і вивід часу в обох форматах відповідними функціями. Далі функція setTime намагається присвоїти даним недопустимі значення. Щоб побачити результат цієї дії, час знову виводиться в обох форматах.

```
#include<iostream.h>
//Означення абстрактного типу даних (ADT)
class Time{
        public:
                Time(); //Конструктор за замовчуванням
                void setTime(int, int, int); //Ініціалізує елементи
                //hour, minute і second
                void printMilitary();//Виводить час у військовому форматі
                void printStandard();//Виводить час у стандартному форматі
        private:
                int hour; //0-23
                int minute; //0-59
                int second; //0-59
};
//Конструктор Time ініціалізує кожен
//елемент даних нулем
//Гарантує, що всі об'єкти Time спочатку знаходяться
// у коректному стані
Time::Time(){hour = minute = second =0;}
//Встановлює нове значення часу Time
//у військовому форматі
//Проводить перевірку правильності даних
//Встановлює недопустимі значення в нуль
void Time::setTime(int h, int m, int s)
{
        if(h>= 0 && h<24){hour=h;}
        else{hour=0;}
        if(m>= 0 && m<60){minute=m;}
        else{minute=0;}
        if(s>= 0 && s<60){second=s;}
        else{second=0;}
```

```
}
//Виводить Time у військовому форматі
void Time::printMilitary()
{
        if(hour<10){cout<<"0";}
        else{cout<<"";}
        cout<<hour<<":";
        if(minute<10){cout<<"0";}
        else{cout<<"";}
        cout<<minute<<":";
        if(second<10){cout<<"0";}
        else{cout<<"";}
        cout<<second;
}
//Виводить час у стандартному форматі
void Time::printStandard()
{
        if(hour==0 || hour==12){cout<<12;}
        else{cout<<hour%12;}
        cout<<":";
        if(minute<10){cout<<"0";}
        else{cout<<"";}
        cout<<minute<<":";
        if(second<10){cout<<"0";}
        else{cout<<"";}
        cout<<second;
        if(hour<12){cout<<" AM";}
        else{cout<<" PM";}
}
//Програма для тестування простого класу Time
int main()
{
        Time t;//Створює об'єкт t класу Time
        cout<<"The initial military time is ";
        t.printMilitary();
        cout<<"\nThe initial standard time is ";
        t.printStandard();
        t.setTime(13,27,6);
        cout<<"\n\nMilitary time after setTime is ";
        t.printMilitary();
        cout<<"\nStandard time after setTime is ";
        t.printStandard();
        t.setTime(99,99,99);//Присвоєння недопустимих
        //значень
        cout<<"\n\nAfter attempting invalid settings:\n"
        <<"Military time: ";
        t.printMilitary();
        cout<<"\nStandard time: ";
        t.printStandard();
```

```
    cout<<endl;
    return 0;
}
```

Результат:

```
The initial military time is 00:00:00
The initial standard time is 12:00:00 AM
Military time after setTime is 13:27:06
Standard time after setTime is 1:27:06 PM
After attempting invalid settings:
Military time: 00:00:00
Standard time: 12:00:00 AM
```

Даним hour, minute і second у програмі передує мітка private. Це закриті дані класу і є недосяжними за межами класу. Ідея полягає в тому, що реальне представлення даних всередині класу не повинно цікавити його клієнтів. Саме в цьому змісті говорять, що реалізація класу прихована від його користувачів. Таке приховування інформації сприяє модифікації програм і спрощує сприйняття класу користувачем.

У розглянутій програмі конструктор класу Time ініціалізує дані значенням 0. Це гарантує, що об'єкт після його створення знаходиться в коректному стані. Даним класу не можуть бути присвоєні недійсні значення, оскільки при створенні об'єкта Time, конструктор викликається автоматично. Всі наступні спроби зміни дані відстежуються функцією setTime.

Завдання для роботи

Задача. Оголосити клас **triangle**, який би знаходив периметр трикутника за його трьома сторонами.

1. Підключити необхідні файли заголовків.
2. Оголосити клас **triangle**. В якості відкритих елементів задайте конструктор класу, функцію присвоєння значень відповідним сторонам **setSide**, функцію виводу на екран результату програми **printP**, тобто периметру. Значення сторін повинні відноситися до типу **float**. У якості закритих елементів класу задайте три сторони трикутника **a, b, c**.
3. За допомогою конструктора класу спочатку встановіть значення сторін трикутника в «0».
4. Сформуйте опис функції, яка присвоює відповідні значення сторонам трикутника **setSide**.
5. Сформуйте опис функції **printP**, яка виконує обчислення і виводить значення периметра трикутника на екран.
6. Оголосіть головну функцію **main()**, в тілі якої оголосіть об’єкт класу **triangle** за допомогою якого Ви будете звертатися до елементів класу.
7. За допомогою об’кта класу задайте значення сторін трикутника.
8. За допомогою цього ж об’єкта класу викличте функцію виводу результатів роботи програми.
9. Результат роботи продемонструвати викладачу.

10. Модифікуйте Вашу програму таким чином, щоб Ви значення сторін трикутника вносили з клавіатури.
11. Модифіковану програму представте викладачу.
12. У якості додаткового завдання створіть клас для перерахунку дистанції у сантиметрах у метри. Ввід відстані у сантиметрах забезпечити із клавіатури.

Контрольні запитання

1. Яку функцію виконує ООП?
2. Що називають об’єктами ООП?
3. Що таке клас і яка його відмінність від структури?
4. Що називають елементами-даними класу?
5. Що називають елементами-функціями класу?
6. Що називають об’єктом класу?
7. Які операції використовуються для доступу до елементів класу.
8. Яке ключове слово використовується для означення класу?
9. Для чого призначені ключові слова public і private у означенні класу?
10. Що таке конструктор?
11. Чи доступні елементи-дані, що з’являються після private? Чи обов’язкове слово private у оголошенні класу?
12. Чи можуть бути ініційованими елементи-дані при їх оголошенні у тілі класу?
13. Що таке set-функції?

Лабораторна робота № 33
ТЕМА: ІНІЦІАЛІЗАЦІЯ КОНСТРУКТОРА ЗА ЗАМОВЧУВАННЯМ

МЕТА РОБОТИ: навчитися оголошувати ініціалізований конструктор за замовчуванням.

Теоретичні відомості

Дані класу не можуть бути ініціалізованими за їх оголошення в тілі класу. Ці дані повинні ініціалізуватися конструктором або їм можуть бути присвоєні значення спеціальними set–функціями. Функція із тим же ім'ям, що і клас, перед яким записаний символ тильди ~, називається деструктором класу. Деструктор проводить завершальне видалення кожного об'єкта класу перед тим, як виділена для нього пам'ять буде повернена системі. В попередньому прикладі деструктор не використовувався.

У відкритих (публічних) функціях реалізовані можливості, які клас надає своїм клієнтам. Відкриті функції класу називаються інтерфейсом або відкритим інтерфейсом класу. Клієнти повинні мати доступ до інтерфейсу класу, але вони не повинні мати доступу до його реалізації.

Функції класу можуть бути означені всередині класу, проте, добрим стилем є означення функцій поза означенням класу. Треба звернути увагу на використання двомісної операції дозволу області дії (::) в означеннях кожної функції, що йдуть за означенням класу у програмі вище. Ці функції повинні бути означені відразу після означення класу і оголошення його функцій.

Будь-яка функція класу може бути означена безпосередньо в його тілі замість включення туди її прототипу. Якщо функція означується після означення класу, то імені функції передують ім'я класу і двомісна операція дозволу області дії (::). Оскільки різні класи можуть мати однакові імена елементів, то операція дозволу області дії узгоджує ім'я елемента з іменем відповідного класу. Ця операція однозначно ідентифікує функції відповідного класу. Означена поза класом функція все одно знаходиться в області дії класу. Її ім'я відоме тільки іншим елементам даного класу, крім випадків, коли звернення до неї відбувається через об'єкт класу, посилання на об'єкт класу або вказівник на об'єкт класу. Функції розміром більше одного або двох рядків означуються поза тілом означення класу. Це допомагає відокремити інтерфейс класу від його реалізації. Якщо функція означена в класі, то вона автоматично стає вбудованою. Означені функції поза означенням класу можна зробити вбудованими шляхом явного використання ключового слова inline. Однак компілятор залишає за собою право не робити вбудованою будь-яку функцію.

Цікаво відзначити, що функції printMilitary і printStandard не мають параметрів, бо непрямим чином знають, які дані повинні виводити. Це робить виклики функцій коротшими у порівнянні з традиційними викликами в процедурному програмуванні. Ця перевага об'єктно-орієнтованого програмування є наслідком того, що інкапсуляція (об'єднання) даних і функцій всередині об'єкта дає функціям класу право звертатися до цих даних. Класи спрощують програмування, оскільки клієнт (користувач об'єкта класу) повинен мати справу тільки з інкапсульованими діями, тобто вбудованими в об'єкт. При розробці таких дій звичайно орієнтуються на клієнтів класу, а не на реалізацію. Клієнти не повинні мати справ з реалізацією класу.

Інтерфейси змінюються набагато рідше реалізацій. При зміні реалізації відповідним чином потрібно змінити залежний від неї код програми. Приховуючи реалізацію, усувається залежність інших частин програми від деталей створення класу.

Нові класи можуть утворюватися від інших класів. Крім цього класи можна включати як елементи інших класів. Такого роду повторне використання програмного забезпечення може значно підвищити продуктивність праці програміста. Створення похідних класів на основі тих, що існують, називається наслідуванням. Включення класів як елементів інших класів називається композицією.

Імена змінних і функцій, що оголошені в означенні класу, належать області дії класу. Функції, що не є елементами класу, оголошуються в області дії файлу. В області дії класу елементи класу безпосередньо доступні для функцій цього класу і допускають звернення за іменем. Поза областю дії класу на елементи класу можна посилатися за допомогою імені об'єкта класу, посилання на об'єкт або вказівника на об'єкт.

Функції всередині класу мають область дії функції. Якщо в функції означена змінна з тим же ім'ям, що і змінна з області дії класу, то остання буде прихованою змінною функції всередині області дії функції. Доступ до таких прихованих змінних можна отримати за допомогою операції дозволу області дії, помістивши перед цією операцією ім'я класу. Доступ до глобальних змінних можна отримати за допомогою одномісної операції дозволу області дії. Операції, що використовуються для доступу до елементів класу, ідентичні операціям для доступу до елементів структури. Для доступу до елементів об'єкта застосовується операція вибору елемента крапка (.) у поєднанні з ім'ям об'єкта або з посиланням на об'єкт. Операція вибору елемента стрілка (->) застосовується в поєднанні із вказівником на об'єкт.

У прикладі нижче для ілюстрації доступу до елементів класу за допомогою операцій вибору елементів використовується клас з ім'ям count. Він містить відкритий елемент даних x типу int і відкриту функцію print. Програма створює екземпляри трьох змінних типу count: counter, counterRef і counterPtr. Змінна counterRef оголошується як посилання на змінну counter, а змінна counterPtr оголошується як вказівник на counter. Важливо звернути увагу на те, що елемент даних x зроблений відкритим тільки для демонстрації звернення до відкритих елементів. Дані звичайно роблять закритими. У всіх наступних прикладах це буде збережено.

```
#include <iostream.h>
class count {
        public:
                int x;
                void print(){cout<<x<<endl;}
};
int main()
{
        count counter;//Створює об'єкт класу
        count *counterPtr=&counter;//Вказівник на counter
        count &counterRef=counter;//Посилання на counter
        cout<<"Assign 7 to x and print using the
        object’s name: ";
        counter.x =7;//Присвоює 7 елементу даних x
        counter.print();//Викликає функцію print
        cout<<"Assign 8 to x and print using a reference: ";
        counterRef.x=8;//Присвоює 8 елементу даних x
        counterRef.print();//Викликає функцію print
        cout<<"Assign 10 to x and print using a pointer: ";
        counterPtr->x=10; //Присвоює 10 елементу даних x
        counterPtr->print();//Викликає функцію print
```

```
        return 0;
    }
```
Результат:
```
Assign 7 to x and print using the object’s name: 7
Assign 8 to x and print using a reference: 8
Assign 10 to x and print using a pointer: 10
```
Одним з найбільш фундаментальних принципів системної розробки програмного забезпечення є відокремлення інтерфейсу від реалізації. Це спрощує модифікацію програм. Що стосується клієнтів класу, зміни в реалізації класу не зачіпають їх доти, доки не буде змінений інтерфейс.

В прикладі нижче показана програма прикладу із лабораторної роботи № 32, розбита на декілька файлів. При побудові програми на C++ означення класу поміщається у файл заголовків, а означення функцій цього класу поміщаються у файли вихідного коду із тим же базовим ім'ям.

Файл заголовків включається (за допомогою #include) у всі файли, що використовують цей клас. Файл із означенням функцій класу компілюється і компонується з файлом, що містить головну програму.

```
//TIME.H
//Оголошення класу Time.
//Функції означені в TIME.CPP
//Не допускає повторних включень файлу заголовків
#ifndef TIME_H
#define TIME_H
//Означення абстрактного типу даних Time
class Time {
    public:
        Time(); //конструктор за замовчуванням
        void setTime(int,int,int);//Ініціалізує елементи hour,minute і second
        void printMilitary();//Виводить час у військовому форматі
        void printStandard();//Виводить час у стандартному форматі
    private:
        int hour;//0-23
        int minute;//0-59
        int second;//0-59
    };
#endif
```
Програма в прикладі складається із файла заголовків time.h, в якому оголошений клас Time, файла time.cpp, в якому означені функції класу Time, і файла program.cpp, в якому означена функція main.

При створенні програм більшого розміру у файли заголовків поміщаються й інші означення та оголошення. Для імен символічних констант в директивах препроцесора використовують ім'я файла заголовків із символом підкреслення замість крапки.

```
//TIME.CPP
//Означення елементів-функцій класу Time.
#include<iostream.h>
#include"time1.h"
//Конструктор Time ініціалізував кожен елемент даних нулем
//Гарантує, що всі об'єкти Time спочатку знаходяться у коректному стані
Time::Time(){hour=minute=second=0;}
```

```
//Встановлює нове значення часу Time у військовому форматі
//Проводить перевірку правильності даних. Встановлює значення в нуль.
void Time::setTime(int h, int m, int s)
{
	if(h>= 0 && h<24){hour=h;}
	else{hour=0;}
	if(m>= 0 && m<60){minute=m;}
	else{minute=0;}
	if(s>= 0 && s<60){second=s;}
	else{second=0;}
}
//Виводить Time у військовому форматі
void Time::printMilitary()
{
	if(hour<10){cout<<"0";}
	else{cout<<"";}
	cout<<hour<<":";
	if(minute<10){cout<<"0";}
	else{cout<<"";}
	cout<<minute<<":";
	if(second<10){cout<<"0";}
	else{cout<<"";}
	cout<<second;
}
//Виводить час в стандартному форматі
void Time::printStandard()
{
	if(hour==0 || hour==12){cout<<12;}
	else{cout<<hour%12;}
	cout<<":";
	if(minute<10){cout<<"0";}
	else{cout<<"";}
	cout<<minute<<":";
	if(second<10){cout<<"0";}
	else{cout<<"";}
	cout<<second;
	if(hour<12){cout<<" AM";}
	else{cout<<" PM";}
}
//PROGRAM.CPP
//Програма-тестер для класу Time
//ЗАУВАЖЕННЯ: Компілювати з TIME1.CPP
#include<iostream.h>
#include "time1.h"
//Програма для тестування простого класу Time
int main()
{
	Time t;//створює екземпляр об'єкта t класу Time
	cout<<"The initial military time is ";
```

```
        t.printMilitary();
        cout<<"\nThe initial standard time is ";
        t.printStandard();
        t.setTime(13,27,6);
        cout<<"\n\nMilitary time after setTime is ";
        t.printMilitary();
        cout<<"\nStandard time after setTime is ";
        t.printStandard();
        t.setTime(99,99,99);//привласнення значень
        cout<<"\n\nAfter attempting invalid settings:\n"
        <<"Military time: ";
        t.printMilitary();
        cout<<"\nStandard time: ";
        t.printStandard();
        cout<<endl;
        return 0;
}
```

Результат:

```
The initial military time is 00:00:00
The initial standard time is 12:00:00 AM
Military time after setTime is 13:27:06
Standard time after setTime is 1:27:06 PM
After attempting invalid settings:
Military time: 00:00:00
Standard time: 12:00:00 AM
```

Не всі функції класу доцільно робити загальнодоступними, тобто частиною інтерфейсу класу. Деякі функції можна залишити закритими і вони будуть як допоміжні для інших функцій класу.

Функції доступу можуть читати або відображати дані. Іншим поширеним застосуванням функцій доступу є перевірка істинності або помилковості умов. Подібні функції часто називають предикативними функціями (сервісними).

У прикладі нижче демонструється поняття сервісної функції. Сервісна функція не є частиною інтерфейсу класу. Це закрита функція класу, яка підтримує роботу відкритих функцій класу. Сервісні функції не призначені для використання клієнтами класу.

```
// SALESP.H
//Означення класу Salesperson
//Функції класу означені в SALESP.CPP
#ifndef SALESP_H
#define SALESP_H
class Salesperson
{
        public:
                Salesperson(); //конструктор
                void setSales(); //користувач задає показники збуту
                void printAnnualSales();
        private:
                double sales[13]; //12 щомісячних показників збуту
                double totalAnnualSales();//сервісна функція
};
```

```
#endif
//SALESP.CPP
//Функції для класу Salesperson
#include<iostream.h>
#include<iomanip.h>
#include"salesp.h"
//Конструктор ініціалізує масив
Salesperson::Salesperson()
{
	for(int i=0; i<=12; i++)
		sales[i]=0.0;
}
//Функція для установки 12 щомісячних показників збуту
void Salesperson::setSales()
{
	for(int i=1; i<=12; i++)
	{
		cout<<"Enter sales amount for month "<<i<<": ";
		cin>>sales[i];
	}
}
//Закрита сервісна функція для обчислення річного збуту
double Salesperson::totalAnnualSales()
{
	double total = 0.0;
	for(int i=1; i<=12; i++)
		total+=sales[i];
	return total;
}
//Виводить підсумковий річний збут
void Salesperson::printAnnualSales()
{
	cout.setf(ios::fixed,ios::floatfield);
	cout.setf(ios::showpoint);
	cout<<setprecision(2)<<"\nThe total annual sales
	are: $ "<<totalAnnualSales()<<endl;
}
//PROGRAM.CPP
//Демонстрація сервісної функції
//Компілюється з SALESP.CPP
#include"salesp.h"
int main()
{
	Salesperson s; //створює об'єкт s класу
	s.setSales();
	s.printAnnualSales();
return 0;
}
```

Результат:

Enter sales amount for month 1: 5314.76
Enter sales amount for month 2: 4292.38
Enter sales amount for month 3: 4589.83
Enter sales amount for month 4: 5534.03
Enter sales amount for month 5: 4376.34
Enter sales amount for month 6: 5698.45
Enter sales amount for month 7: 4439.22
Enter sales amount for month 8: 5893.57
Enter sales amount for month 9: 4909.67
Enter sales amount for month 10: 5123.45
Enter sales amount for month 11: 4024.97
Enter sales amount for month 12: 5923.92
The total annual sales are: $ 60120.59

Клас Salesperson містить масив з 12 щомісячних показників збуту, які ініціалізуються нулями в конструкторі. Функція setSales привласнює показникам збуту значення, що задаються користувачем. Відкрита функція класу printAnnualSales виводить підсумковий збут за останні 12 місяців. Сервісна функція totalAnnualSales підсумовує 12 щомісячних показників збуту для функції printAnnualSales.

Після створення об'єктів класу, його елементи можуть бути ініціалізовані за допомогою конструкторів. Конструктор автоматично викликається кожен раз, коли створюється об'єкт класу. Дані класу не можуть бути ініціалізованими в означенні класу. Вони повинні бути ініціалізованими в конструкторі класу або їх значення можуть бути задані після створення об'єкта. Конструктор не може ні специфікувати тип значення, що повертається, ні повертати будь-яке значення. Конструктори можуть бути переозначені, щоб передбачити різні способи ініціалізації об'єктів класу.

При оголошенні об'єкта класу, праворуч від його імені і до крапки з комою в круглих дужках, можуть бути задані ініціалізатори. Ці ініціалізатори передаються як аргументи в конструктор класу.

Конструктор з time.cpp ініціалізував hour, minute і second нульовими значеннями. Однак, конструктори можуть містити аргументи за замовчуванням. У прикладі конструктор класу Time переозначується так, щоб він включав для кожної змінної аргумент із нульовим значенням за замовчуванням. Аргумент за замовчуванням гарантує, що об'єкт буде знаходиться в коректному стані навіть якщо не буде задано жодних значень при виклику конструктора. Конструктор, усі аргументи якого мають значення за замовчуванням, називається конструктором за замовчуванням. Такий конструктор може викликатися без аргументів. Конструктор використовує такий самий код перевірки допустимості даних, що і функція setTime. Це гарантує, що значення елемента hour знаходиться в діапазоні від 0 до 23, значення кожного з елементів minute і second знаходяться в діапазоні від 0 до 59. Якщо значення потрапляє за межі діапазону, воно отримує значення нуль.

Конструктор міг би безпосередньо викликати setTime, проте це привело б до накладних витрат, пов'язаних з додатковим викликом функції. У програмі нижче ініціалізовано п'ять об'єктів класу Time. Об'єкт t1 має всі аргументи, що приймаються за замовчуванням при виклику конструктора. t2 приймає один аргумент, t3 має два вказані аргументи, t4 із трьома вказаними аргументами і t5 із трьома недопустимими значеннями аргументів. Вміст елементів даних кожного об'єкта після його створення й ініціалізації відображається на екрані.

//TIME.H
//Оголошення класу Time.

```
//Функції класу означені в TIME.CPP.
//Не допускає повторних включень файлу заголовків.
#ifndef TIME2_H
#define TIME2_H
class Time
{
	public:
		Time(int hr=0,int min=0,int sec=0);//конструктор за замовчуванням
		void setTime(int, int, int);
		void printMilitary();
		void printStandard();
	private:
		int hour;
		int minute;
		int second;
};
#endif
//TIME.CPP
//Означення функцій класу Time.
#include"time.h"
#include<iostream.h>
//Конструктор для ініціалізації закритих даних.
//Значення за замовчуванням рівні 0
//(див.означення класу).
Time::Time(int hr, int min, int sec)
{
	if(hr>=0 && hr<24){hour=hr;}
	else{hour=0;}
	if(min>=0 && min<60){minute=min;}
	else{minute=0;}
	if(sec>= 0 && sec<60){second=sec;}
	else{second=0;}
}
//Встановлює значення hour, minute і second.
//Недопустимі значення встановлюються в 0.
void Time::setTime(int h, int m, int s)
{
	if(h>=0 && h<24){hour=h;}
	else{hour=0;}
	if(m>=0 && m<60){minute=m;}
	else{minute=0;}
	if(s>=0 && s<60){second=s;}
	else{second=0;}
}
//Відображає час у військовому форматі: HH:MM:SS
void Time::printMilitary()
{
	if(hour<10){cout<<"0";}
	else{cout<<"";}
```

```
        cout<<hour<<":";
        if(minute<10){cout<<"0";}
        else{cout<<"";}
        cout<<minute<<":";
        if(second<10){cout<<"0";}
        else{cout<<"";}
        cout<<second;
}
//Відображає час у стандартному форматі: HH:MM:SS
//AM (або PM)
void Time::printStandard()
{
        if(hour==0 || hour==12){cout<<12;}
        else{cout<<hour%12;}
        cout<<":";
        if(minute<10){cout<<"0";}
        else{cout<<"";}
        cout<<minute<<":";
        if(second<10){cout<<"0";}
        else{cout<<"";}
        cout<<second;
        if(hour<12){cout<<" AM";}
        else{cout<<" PM";}
}
//PROGRAM.CPP
//Демонстрація конструктора за замовчуванням
//для класу Time.
#include<iostream.h>
#include"time.h"
int main()
{
        Time t1,t2(2),t3(21,34),t4(12,25,42),t5(27,74,99);
        cout<<"Consrtucted with:\n"
        <<"all arguments defaulted: \n";
        t1.printMilitary();
        cout<<"\n";
        t1.printStandard();
        cout<<"\nhour specified; minute and second defaulted:\n";
        t2.printMilitary();
        cout<<"\n";
        t2.printStandard();
        cout<<"\nhour and minute specified; second defaulted:\n";
        t3.printMilitary();
        cout<<"\n";
        t3.printStandard();
        cout<<"\nhour, minute and second specified:\n";
        t4.printMilitary();
        cout<<"\n";
        t4.printStandard();
```

```
        cout <<"\nall invalid values specified:\n";
        t5.printMilitary();
        cout<<"\n";
        t5.printStandard();
        cout<<endl;
return 0;
}
```

Результат:

```
Constructed with:
all arguments defaulted:
00:00:00
12:00:00 AM
hour specified; minute and second defaulted:
02:00:00
2:00:00 AM
hour and minute specified; second defaulted:
21:34:00
9:34:00 PM
hour, minute and second specified:
12:25:42
12:25:42 PM
all invalid values specified:
00:00:00
12:00:00 AM
```

Якщо для класу не означений жодний конструктор, компілятор створює конструктор за замовчуванням. Такий конструктор не виконує жодної ініціалізації, тому після створення об'єкта не гарантується його коректний стан.

Деструктор – це спеціальна функція класу. Ім'я деструктора складається із символу тильди ~, за яким слідує ім'я класу.

Деструктор класу викликається автоматично, коли об'єкт класу виходить з області дії. Деструктор не приймає параметрів і не повертає значення. Клас може мати тільки один деструктор. Переозначення деструктора не допускається. Деструктори рідко використовуються із простими класами. Застосування деструкторів доцільне для класів, об'єкти яких містять динамічно виділену пам'ять, наприклад, для масивів і рядків.

В більшості випадків конструктори і деструктори викликаються автоматично. Порядок, в якому відбуваються звернення до цих функцій залежить від порядку, в якому об'єкти входять в область дії і виходять із неї.

Як правило, виклики деструкторів відбуваються в зворотному порядку до виклику конструкторів. Проте період зберігання об'єктів може впливати на порядок викликів деструкторів.

Для об'єктів, оголошених у глобальній області дії, конструктори викликаються на початку виконання програми. Відповідні деструктори викликаються при її завершенні.

Для локальних об'єктів конструктори викликаються за їх оголошення. Відповідні деструктори викликаються, коли об'єкти виходять із області дії (тобто відбувається вихід із блоку, в якому вони оголошені). Треба звернути увагу, що конструктори і деструктори для локальних об'єктів можуть викликатися багато раз у міру того, як об'єкти входять в область дії і виходять із неї.

Для статичних локальних об'єктів конструктори викликаються один раз за оголошення цих об'єктів. Відповідні деструктори викликаються при завершенні роботи програми. Програма нижче демонструє порядок, в якому викликаються конструктори і деструктори для об'єктів типу CreateAndDestroy, що знаходяться в різних областях дії. Програма оголошує об'єкт first в глобальній області дії. Його конструктор викликається на початку виконання програми, а деструктор – при завершенні роботи програми після знищення всіх інших об'єктів.

```
//CREATE.H
//Означення класу CreateAndDestroy
//Функції класу означені в CREATE.CPP
#ifndef CREATE_H
#define CREATE_H
class CreateAndDestroy
{
        public:
                CreateAndDestroy(int); // конструктор
                ~CreateAndDestroy(); //деструктор
        private:
                int data;
};
#endif
//CREATE.CPP
//Означення функцій класу CreateAndDestroy
#include<iostream.h>
#include"create.h"
CreateAndDestroy::CreateAndDestroy(int value)
{
        data=value;
        cout<<"Object "<<data<<" constructor ";
}
CreateAndDestroy::~CreateAndDestroy()
{
        cout<<"Object "<<data<<" destructor "<<endl;
}
//PROGRAM.CPP
//Демонстрація порядку, в якому викликаються конструктори і деструктори
#include<iostream.h>
#include"create.h"
void create(void); //прототип
CreateAndDestroy first(1); // глобальний об'єкт
int main()
{
        cout<<"(global created before main) \n";
        CreateAndDestroy second(2); //локальний об'єкт
        cout<<"(local automatic in main) \n";
        static CreateAndDestroy third(3);//локальний об'єкт
        cout<<"(local static in main) \n";
        create(); //викликає функцію, що створює об'єкти
        CreateAndDestroy fourth(4); //локальний об'єкт
```

```
        cout<<"(local automatic in main) \n";
        return 0;
}
//Функція, що створює об'єкти
void create(void)
{
        CreateAndDestroy fifth(5);
        cout<<"(local automatic in create) \n";
        static CreateAndDestroy sixth(6);
        cout<<"(local static in create) \n";
        CreateAndDestroy seventh(7);
        cout<<"(local automatic in create) \n";
}
```

Результат:

```
Object 1 constructor (global created before main)
Object 2 constructor (local automatic in main)
Object 3 constructor (local static in main)
Object 5 constructor (local automatic in create)
Object 6 constructor (local static in create)
Object 7 constructor (local automatic in create)
Object 7 destructor
Object 5 destructor
Object 4 constructor (local automatic in main)
Object 4 destructor
Object 2 destructor
Object 6 destructor
Object 3 destructor
Object 1 destructor
```

У функції main оголошені локальні об'єкти second і fourth, а об'єкт third оголошений статичним локальним об'єктом. Конструктори для кожного з цих об'єктів викликаються за оголошення кожного з них. Деструктори об'єктів fourth і second викликаються (у такому порядку), досягнувши закінчення функції main. Оскільки об'єкт third є статичним, то він існує з моменту свого оголошення і до завершення роботи програми. Деструктор об'єкта third викликається перед деструктором об'єкта first, але після руйнування всіх інших об'єктів. У функції create оголошені локальні об'єкти fifth і seventh, а об'єкт sixth оголошений статичним локальним об'єктом. Деструктори об'єктів seventh і fifth викликаються (у такому порядку), досягнувши закінчення функції create. Оскільки об'єкт sixth є статичним, то він існує з моменту свого оголошення і до завершення роботи програми. Деструктор об'єкта sixth викликається перед деструкторами об'єктів third і first, але після знищення всіх решти об'єктів.

Завдання для роботи

Задача. Клас який Ви створите може бути корисний практично для будь-якої програми. Лічильник – це засіб, який призначений для збереження кількісної міри певної величини яка змінюється. Лічильник може зберігати число звернень до файла, число разів, скільки користувач натиснув клавішу Enter, або кількість клієнтів у банку. Як правило при настанні деякої події лічильник збільшується на одиницю. Звернення до

лічильника відбувається як правило для того, щоб дізнатися поточне значення величини, для якої він був призначений. Отже необхідно створити клас лічильника.

Створювати клас із розбиттям на окремі файли!!!

1. Підключаємо необхідні файли заголовків.
2. Оголошуємо клас **Counter**. У якості відкритих елементів оголошуємо конструктор класу із присвоєнням за замовчуванням лічильнику значення «0», функцію інкрементування лічильника **inc_count**, функцію отримання значення лічильника **get_count**. У якості закритих елементів оголошуємо змінну лічильника **count** типу **unsigned int**, оскільки його значення не може бути від'ємним.
3. Оголосити конструктор для ініціалізації закритих даних.
4. Описати дії функції **inc_count**.
5. Описати дії функції **get_count**.
6. Оголосити головну функцію **main()**.
7. В тілі головної функції оголосити два об'єкти класу.
8. Вивести на екран значення лічильника за замовчуванням для обох об'єктів класу.
9. Виконати інкрементування лічильника для обох об'єктів класу.
10. Повторно виконати інкрементування лічильника для другого об'єкта класу.
11. Вивести на екран значення лічильника для обох об'єктів класу після інкрементування.
12. Результати роботи продемонструвати викладачу.

Контрольні запитання

1. Що таке інтерфейс класу?
2. Чи може бути означена елемент-функція в тілі класу, чи тільки поза ним?
3. Яка елемент-функція називається вбудованою?
4. Що таке наслідування?
5. Що таке композиція?
6. Що таке область дії класу і область дії файла?
7. Для чого використовується ключове слово inline?
8. Що відбувається коли змінна елемент-функції і області дії класу мають однакове ім'я?
9. Які функції називають функціями доступу і для чого вони призначені?
10. Які функції називають предикативними?
11. Які функції називають сервісними і для чого вони використовуються?
12. Чи може конструктор містити аргументи за замовчуванням?
13. Що відбувається коли не створено жодного конструктора класу?
14. Що таке деструктор і скільки деструкторів може мати клас? Коли доцільно використовувати деструктори?

Лабораторна робота № 34
ТЕМА: ОБ’ЄКТИ КЛАСУ В ЯКОСТІ АРГУМЕНТІВ ФУНКЦІЙ

МЕТА РОБОТИ: вивчити декілька нових аспектів створення нових класів, а саме перевантаження конструкторів, означення елементів класу поза межами класу, навчитися використовувати об’єкти класу в якості аргументів функцій.

Теоретичні відомості

Операція привласнення (=) використовується для привласнення деякому об'єкту іншого об'єкта того ж типу. Таке привласнення, звичайно, виконується за допомогою поелементного копіювання – кожен елемент одного об'єкта окремо копіюється в такий же елемент іншого об'єкта. Поелементне копіювання може викликати серйозні проблеми, якщо воно використовується з класом, елементи даних якого містяться в пам'яті, що динамічно виділяється.

Об'єкти можна передавати функціям як аргументи і можна повертати їх із функцій. За замовчуванням така передача параметрів (і їх повернення) відбувається за значенням – передається або повертається копія об'єкта.

```
//Демонстрація можливості привласнення об'єктів класу
//за допомогою поелементного копіювання
//*********************************************************
#include <iostream.h>
// Простий клас Date
class Date {
        public:
                Date (int = 1, int = 1, int = 2009); //конструктор за замовчуванням
                void print();
        private:
                int month;
                int day;
                int year;
};
//Конструктор класу Date без перевірки діапазону
Date::Date(int m, int d, int y)
{
        month = m; day = d; year = y;
}
//Виводить об'єкт Date у форматі mm-dd-yyyy
void Date::print()
{
        cout << month << '-' << day << '-' << year;
}
int main()
{
        Date d1 (7, 4, 2009), d2; //d2 приймає значення за замовчуванням 1/1/2009
        cout << "d1 = ";
        d1.print();
```

```
        cout << "\nd2 = ";
        d2.print();
        d2=d1; //привласнення за допомогою по елементного копіювання
                //за замовчуванням
        cout << "\n\nAfter default memberwise copy, d2 = ";
        d2.print();
        cout << endl;
return 0;
}
```

Результат:

```
d1 = 7-4-2009
d2 = 1-1-2009
After default memberwise copy, d2 = 7-4-2009
```

Передача об'єктів за значенням добра з погляду безпеки, оскільки функція, що викликається, не має доступу до початкового об'єкта. Проте виклик за значенням може знижувати продуктивність системи в разі створення копії об'єкта. Об'єкт може передаватися за посиланням шляхом передачі або вказівника на об'єкт, або посилання на нього. Передача за посиланням підвищує продуктивність, але слабша з погляду безпеки, оскільки функція, що викликається, має доступ до початкового об'єкта. Безпечною альтернативою є передача об'єкта за посиланням на константу.

При написанні об'єктно-орієнтованих програм, основна увага концентрується на реалізації корисних класів. Існує багатообіцяюча можливість створення каталогів класів, до яких будуть мати доступ великі групи програмістів. Вже є безліч бібліотек класів і по всьому світу розробляються нові бібліотеки. Прикладаються всі зусилля для того, щоб зробити ці бібліотеки широко доступними. В цьому випадку програмне забезпечення могло б створюватися на основі вже існуючих, точно означених, ретельно перевірених, добре документованих і широкодоступних компонентів. Такого роду повторне використання програмного коду може прискорити розробку могутнього і високоякісного програмного забезпечення.

Проте, перш ніж потенціал повторного використання програмного забезпечення можна буде реалізувати повністю, повинні бути вирішені суттєві проблеми. Потрібні схеми каталогізації, схеми патентування і механізми захисту, що гарантують достовірність вихідних копій класів. Потрібні схеми опису, що дозволяють розробникам нових систем визначати, чи відповідають їхнім потребам існуючі об'єкти. Не обійтися також і без механізмів перегляду, що дозволяють визначати, які класи доступні і наскільки точно вони відповідають вимогам розробника програмного забезпечення і т.д.

Завдання для роботи

Задача. Як відомо в англійській системі мір основними одиницями вимірювання довжини є фут і дюйм, причому 1 фут рівний 12 дюймам. Відстань, яка рівна 15 футам і 8 дюймам записується як 15’-8’’. Дефіс в даному записі не означає знак «відняти», а служить для розділення футів і дюймів. Припустимо, Вам необхідно створити інженерний проект з використанням англійської системи мір. Було б зручно представляти довжини у вигляді двох чисел, одне із яких би дорівнювало кількості футів, а друге – кількості дюймів. Клас, який би описував вище сказане повинен виконувати операцію додавання двох довжин представлених у англійській системі мір.

1. Засобами Microsoft Visual Studio створити консольний проект C++.

2. Додати до проекту новий файл заголовків distance.h. В цьому файлі оголосити клас Distance. В якості відкритих елементів класу оголосити: 1)конструктор за замовчуванням без аргументів із присвоєнням нульових значень змінним футів і дюймів, маючи на увазі, що перша змінна є цілого типу, а друга – дійсного (конструктор : змінна(значення), змінна(значення){}); 2) конструктор із двома аргументами, вказуючи тип і назву аргументу, а також присвоїти відповідним змінним футів і дюймів значень, що поступають аргументам (конструктор (тип аргумент, тип аргумент) : змінна_футів(аргумент), змінна_дюймів(аргумент){}); 3) оголосити прототип функції getdist(), яка виконує функції вводу даних користувачем; 4) оголосити прототип функції showdist, яка здійснює вивід футів і дюймів на екран; 5) оголосити прототип функції add_dist(аргумент1, аргумент2), яка виконує додавання довжин. У якості закритих елементів класу оголосити змінну feet цілого типу яка буде містити фути і змінну inches дійсного типу, яка буде містити дюйми.
3. Додати до проекту новий файл distance.cpp, який буде містити описи функцій, прототипи яких оголошені в класі.
4. В цьому файлі оголосити опис функції getdist(), який повинен містити повідомлення про необхідність вводу з клавіатури даних для футів і дюймів.
5. Оголосити опис функції showdist(), який має виводити дані у такому вигляді: фути’-дюйми”.
6. Оголосити опис функції add_dist(аргумент1, аргумент2), де в якості аргументів використати об’єкти класу, що будуть містити величини, які ми хочемо додати. В описі виконати додавання дюймів, причому до значень змінних необхідно звертатися за допомогою об’єкта класу і операції «.» (крапка) – змінна_дюйми=об’єкт1.змінна_дюйми+об’єкт2.змінна_дюйми. Початкове значення футів рівне «0». Якщо число дюймів більше 12.0, то від числа дюймів відняти 12.0, а число футів збільшити на одиницю. Після чого виконати додавання футів:
   змінна_фути=змінна_фути+об’єкт1.змінна_фути+об’єкт2.змінна_фути.
7. Додати до проекту файл тестування роботи класу main.cpp.
8. В цьому файлі оголосити два об’єкти класу dist1 і dist3 (дві довжини).
9. Оголосити ще один об’єкт класу dist2 з параметрами (параметри 11 і 6.25).
10. Виконати ввід даних футів і дюймів з клавіатури, для цього викличте об’єктом dist1 функцію getdist().
11. Виконайте додавання довжин dist1 і dist2 і результат помістіть в dist3. Для цього викличте функцію add_dist(аргумент1, аргумент2) за допомогою об’єкта dist3. В якості аргументів функції виступають dist1 і dist2.
12. Виведіть на екран першу, другу і третю довжини.

Контрольні запитання

1. Як виконати ініціалізацію змінних класу в момент їх створення?
2. Яким чином необхідно викликати конструктор при ініціалізації змінних класу в момент їх створення?
3. Як виконується опис функцій класу поза межами класу?
4. Який символ використовується для операції глобального дозволу?
5. Як елементи-функції мають доступ до членів інших об’єктів свого класу?
6. Як і коли об’єкти класу використовуються у якості аргументів елементів-функцій?

Лабораторна робота № 35
ТЕМА: ПЕРЕВАНТАЖЕННЯ ОПЕРАЦІЙ

МЕТА РОБОТИ: навчитися перевантажувати стандартні операції С++ для роботи із власними типами даних.

Теретичні відомості

Програмування в С++ є процесом чутливим до типу даних і орієнтоване на типи даних. Можна використовувати вбудовані типи і можна створювати нові. Вбудовані типи можуть використовуватися з широким набором операцій С++. Також можна використовувати операції з типами, що створені користувачем. Незважаючи на те, що С++ не дозволяє створювати нові операції, вона дозволяє переозначувати існуючі операції. Це дуже сильна сторона С++. Переозначення операцій сприяє розширюваності С++ і, поза сумнівом, є однією з найпривабливіших особливостей цієї мови програмування.

Операції переозначуються за допомогою означення функції за винятком того, що ім'ям функції стає ключове слово operator з подальшим символом операції, що переозначується. Наприклад, ім'я функції operator+ могло б використовуватися для переозначення адитивної операції.

Щоб використовувати операцію з об’єктами класу, вона повинна бути переозначеною. Тут може бути два винятки. Операція присвоєння «=» може використовуватися з будь-яким класом без переозначення. За замовчуванням операція присвоєння забезпечує поелементне копіювання елементів даних класу. Така поелементна копія представляє небезпеку для класів, елементи яких посилаються на динамічно виділені області пам'яті. Для таких класів, як правило, операція присвоєння переозначується. Операція адреси «&» також може використовуватися з об'єктами будь-якого класу без переозначення. Ця операція повертає адресу об'єкта в пам'яті. Операція адреси може бути також переозначена.

Переозначення найбільшою мірою підходить для математичних класів. Вони часто вимагають переозначення великого переліку операцій, щоб зберігалася узгодженість із способом обробки математичних об’єктів, які існують у реальному житті. Наприклад, було б дивно переозначувати тільки операцію сумування для класу комплексних чисел, оскільки інші арифметичні операції над комплексними числами проводяться не менш часто.

При виконанні переозначення операції необхідно забезпечувати ту ж лаконічність виразів, яку забезпечує С++ для вбудованих типів за рахунок багатого набору операцій. Переозначення операцій не є автоматичним процесом. Для виконання потрібної процедури потрібно написати функцію, яка виконає операцію переозначення. Іноді цю роль краще відіграють функції класу, а іноді дружні функції. Час від часу цю роль можуть відігравати функції, що не належать класу, чи відносяться до друзів.

У С++ є можливість вводу і виводу стандартних типів даних з використанням операцій видобування з потоку «>>» і передачі в потік «<<». Ці операції є переозначеними (у бібліотеці класів, що поставляються з компіляторами С++) і можуть обробляти будь-який стандартний тип даних, включаючи рядки та адреси пам’яті. Крім того, операції передачі і добування з потоку можуть бути переозначені, щоб виконувати ввід і вивід типів, означених користувачем. У прикладі нижче показано переозначення операцій передачі і добування з потоку. Ці операції мають можливість обробляти дані класу телефонного номера PhoneNumber, що означений користувачем. У цій програмі

припускається, що телефонний номер введений коректно. Перевірку коректності вводу номеру можна залишити як вправу для самостійної роботи.

Функція-операція добування з потоку (operator>>) приймає у якості аргументів посилання (input) на istream і посилання (num) на тип, що означений користувачем (PhoneNumber). Повертає ця функція посилання на istream. В прикладі функція-операція operator>> використовується для вводу телефонного номера у форматі (800) 555-1212 в об'єкт класу PhoneNumber. Коли компілятор зустрічає в main вираз

cin >> phone;

він генерує виклик функції

operator>>(cin, phone);

Коли виконується цей виклик, параметр input стає псевдонімом для cin, а параметр num стає псевдонімом для phone. Функція-операція використовує функцію getline класу istream для читання трьох частин телефонного номера в елементи areaCode, exchange і line об'єкта типу PhoneNumber. Символи дужок, пропуску і тире пропускаються викликом ignore (функції класу istream), яка відкидає задане число символів у вхідному потоці (один символ за замовчуванням). Функція operator>> повертає input (тобто cin) як посилання на istream. Це дозволяє конкатенацію операцій вводу об'єктів PhoneNumber з операціями вводу інших об'єктів PhoneNumber або об'єктів інших типів. Наприклад, два об'єкти PhoneNumber можуть вводитися таким чином

cin>>phone1>>phone2;

Спочатку виконався б вираз cin>>phone1 за допомогою виклику функції

operator>>(cin, phone1);

Цей виклик повернув би потім cin як значення виразу cin>>phone1. Частина виразу, яка залишилася, інтерпретувалася б як cin>>phone2.

```
//Переозначення операцій передачі в потік
//і добування з потоку
//*********************************************************
#include <iostream.h>
class PhoneNumber
{
        friend ostream &operator<<(ostream &, const PhoneNumber &);
        friend istream &operator>>(istream &, PhoneNumber &);
        private:
                char areaCode[4]; //три цифри і нуль
                char exchange[4]; //три цифри і нуль
                char line[5]; //чотири цифри і нуль
};
//Переозначення операції передачі в потік (не може бути функцією класу).
ostream &operator<<(ostream &output, const PhoneNumber &num)
{
        output<<"("<<num.areaCode<<") "<<
        num.exchange<<"-"<<num.line;
        return output; //дозволити cout<<a<<b<<c;
}
//Переозначення операції добування з потоку
istream &operator>>(istream &input, PhoneNumber &num)
{
        input.ignore(); //пропуск "("
        input.getline(num.areaCode, 4); //ввід areaCode
```

```
        input.clear();//для відновлення потоку у робочий стан
        input.ignore(2); //пропуск ")" і пропуску
        input.getline(num.exchange, 4); //ввід exchange
        input.clear();//для відновлення потоку у робочий стан
        input.ignore(); //пропустити тире "-"
        input.getline(num.line, 5); //ввести line
        return input; //дозволити cin>>a>>b>>c;
}
int main()
{
        PhoneNumber phone; //створити об'єкт phone
        cout<<"Enter a phone number in the "<<"form (123) 456-7890:"<<endl;
        //cin>>phone викликає функцію operator>> викликаючи operator>>(cin, phone)
        cin>>phone;
        //cout<<phone викликає функцію operator<<викликаючи operator<<(cout, phone)
        cout<<"The phone number entered was:"<<endl;
        cout<<phone<<endl;
        return 0;
}
```

Результат:

```
Enter p phone number in the form (123) 456-7890:
(321) 456-9874
The phone number entered was:
(321) 456-9874
```

Операція передачі в потік приймає як аргументи посилання (output) на osrteam і посилання (num) на тип, означений користувачем (PhoneNumber), і повертає посилання на ostream. Функція operator<< виводить об'єкти типу PhoneNumber. Коли компілятор зустрічає в main вираз cout<< phone, він генерує виклик функції

```
operator<<(cout, phone);
```

Функція operator<< виводить частини телефонного номера як рядки, оскільки вони зберігаються у форматі рядка.

Треба звернути увагу, що функції operator>> і operator<< оголошуються дружніми для класу PhoneNumber. Ці операції не можуть бути елементами класу, оскільки об'єкти класу PhoneNumber завжди з'являються в операції у якості правого операнда. Операнд класу повинен перебувати в лівій частині, щоб можна було переозначувати операцію функцією класу. Переозначені операції вводу і виводу повинні оголошуватися як друзі, якщо їм необхідно мати прямий доступ до елементів класу, не оголошених як public.

Нові можливості вводу-виводу типів користувача можуть бути запроваджені в C++ без модифікації оголошень і елементів даних private в будь-якому з класів ostream або istream. Це сприяє розширюваності мови програмування C++, що є одним з найпривабливіших її аспектів.

Одномісна операція для класу може бути переозначена як нестатична функція класу, що немає аргументів, або як функція, що не належить класу і має один аргумент. Цей аргумент повинен бути або об'єктом класу або посиланням на нього.

Ми переозначемо одномісну операцію «!» для перевірки вмісту рядка. При переозначенні одномісної операції «!» у якості нестатичної функції класу без аргументів, компілятор при зустрічі виразу «!s», генерує виклик s.operator!(). Операнд s є об'єктом класу, для якого викликається функція класу operator!:

```
class String
{
        public:
                int operator!() const;
                ...
```

Одномісна операція «!» може бути переозначена як функція, що не належить класу і має один аргумент двома різними способами. Перший спосіб це переозначення з аргументом, що є об’єктом класу, а другий – з аргументом, який посилається на об’єкт. Якщо s є об’єктом класу, то запис !s сприймається, як і виклик operator!(s).

```
class String
{
        friend int operator!(const String &);
        ...
};
```

При переозначенні одномісних операцій необхідно робити функції-операції елементами класу, а не дружніми функціями, що не належать класу. Потрібно уникати дружніх функцій і дружніх класів, якщо тільки це не є вкрай необхідним. Використання друзів порушує інкапсуляцію класу.

Двомісна операція може бути переозначена як нестатична функція класу з одним аргументом або як функція, що не належить класу і має два аргументи. Один з цих аргументів повинен бути або об’єктом класу, або посиланням на нього.

Переозначимо операцію «+=», для виконання конкатенації двох об’єктів рядкового типу. Якщо «y» і «z» є об’єктами класу, то двомісна операція «+=» переозначена як нестатична функція класу String з одним аргументом, для цих об'єктів (y+=z) сприймається як y.operator+=(z).

```
class String
{
        public:
                String &operator+=(const String &);
                ...
};
```

Двомісна операція «+=» може бути переозначена як функція з двома аргументами, що не належить класу. Один з аргументів повинен бути об’єктом класу або посиланням на об’єкт. Якщо «y» і «z» є об’єктами класу, тоді y+=z сприймається в програмі, як виклик operator+=(y,z).

```
class String
{
        friend int &operator+=(String &, const String &);
        ...
};
```

Завдання для роботи

Задача. В програмі лабораторної роботи № 34 ми мали два об’єкти класу Distance, які додавалися за допомогою елемента-функції add_dist():

```
dist3.add_dist(dist1, dist2);
```

Використовуючи перевантаження операції «+», переписати програмний код, який би не використовував попередню функцію, а звичну для нас операцію додавання:

```
dist3=dist1+dist2;
```

Тобто виконати перевантаження операції «+» для додавання змінних типу Distance.

1. Засобами Microsoft Visual Studio створити консольний проект C++.
2. Додати до проекту новий файл заголовків distance.h.
3. В цьому файлі оголосити клас Distance.
4. В якості відкритих елементів класу оголосити: 1)конструктор за замовчуванням без аргументів із присвоєнням нульових значень змінним футів і дюймів, маючи на увазі, що перша змінна є цілого типу, а друга – дійсного (конструктор : змінна(значення), змінна(значення){}); 2) конструктор із двома аргументами, вказуючи тип і назву аргументу, а також присвоїти відповідним змінним футів і дюймів значень, що поступають аргументам (конструктор (тип аргумент, тип аргумент) : змінна_футів(аргумент), змінна_дюймів(аргумент){}); 3) оголосити прототип функції getdist(), яка виконує функції вводу даних користувачем; 4) оголосити прототип функції showdist, яка здійснює вивід футів і дюймів на екран і є константною; 5) оголосити прототип функції, яка виконує перевантаження операції «+» для додавання змінних типу Distance. Пам'ятаємо, що для запису таких функцій спочатку пишуть тип, який буде повертатися даною функцією (тип визначається типом змінних, які будуть використовуватися даною функцією), після цього ключове слово operator, дальше саму операцію яка перевантажується (в нашому випадку «+»), і на останок список аргументів функції у круглих дужках. У якості аргументів задається знову ж таки тип змінних, для яких призначена дана функція. Крім того дану функцію необхідно зробити константною за допомогою ключового слова const, для того щоб вона не могла змінювати значення закритих елементів свого класу.
5. У якості закритих елементів класу оголосити змінну feet цілого типу, яка буде містити фути і змінну inches дійсного типу, яка буде містити дюйми.
6. Додати до проекту новий файл distance.cpp, який буде містити описи функцій, прототипи яких оголошені в класі.
7. В цьому файлі оголосити опис функції getdist(), який повинен містити повідомлення про необхідність вводу з клавіатури даних для футів і дюймів.
8. Оголосити опис функції showdist(), який має виводити дані у такому вигляді: фути'-дюйми".
9. Оголосити опис константної функції operator+ типу Distance, яка в якості аргументу містить оголошення об'єкта класу Distance d2.
10. В тілі функції перевантаження операції «+» здійснюємо додавання футів, а результат присвоюємо тимчасовій змінній f типу int, тобто int f = feet + d2.feet;
11. Аналогічно додаємо дюйми і результат присвоюємо тимчасовій змінній і типу float.
12. Якщо число дюймів більше 12.0, то від числа дюймів відніміть 12.0, а число футів збільшити на одиницю.
13. Після умови розгалуження записати вираз повернення даною функцією числа футів і дюймів.
14. Додати до проекту файл тестування роботи класу main.cpp.
15. В цьому файлі за допомогою конструктора за замовчуванням оголосити три об'єкти класу dist1, dist3 і dist4(три довжини).
16. Виконати ввід даних футів і дюймів з клавіатури, для цього зверніться об'єктом dist1 до функції getdist().
17. Оголосити ще один об'єкт класу dist2 з одночасною ініціалізацією його даними 11 і 6.25.

18. Додати дві змінних dist1 і dist2, а результат присвоїти змінній dist3.
19. Дальше додайте три змінні dist1, dist2, dist3, а результат присвойте змінній dist4.
20. Вивести на екран кожну дистанцію: dist1, dist2, dist3 і dist4.

Контрольні запитання

1. Що дає перевантаження операцій в C++?
   а) перетворює операції C++ для роботи з об’єктами;
   б) надає операціям C++ більше, ніж вони можуть опрацювати;
   в) дає нове значення існуючим в C++ операціям;
   г) створює нові операції C++.
2. Припустимо клас X не використовує перевантажені операції. Напишіть вираз, в якому віднімається об’єкт x1 класу X від іншого об’єкта x2 класу X, а результат поміщається в x3.
3. Припустимо, що клас X включає в себе процедуру перевантаження операції «-». Напишіть вираз, який буде виконувати дії приведені в пункті 2 для цього випадку.
4. Чи істинне наступне твердження: операція >=може бути перевантажена?
5. Скільки аргументів потрібно для означення переватаженої унарної операції?
6. Чи істинне наступне твердження: перевантажена операція завжди потребує на один аргумент менше, ніж кількість операндів?
7. Коли перевантажується операція арифметичного присвоєння, то результат:
   а) передається об’єкту справа від операції;
   б) передається об’єкту зліва від операції;
   в) передається об’єкту, що викликав операцію;
   г) повинен бути повернений.