**Лабораторна робота №11. Написання програм з використанням процедур і функцій**

**На період** роботи в дистанційній формі навчання можна використовувати **компілятор C++ Shell, який доступний за адресом** [**http://cpp.sh**](http://cpp.sh) (інструкція надана після тексту ЛР№3). При наявності на домашньому комп’ютері іншого компілятору С++ завдання можна виконати на ньому. Результати надсилати на електронну адресу викладача [**t.i.lumpova@gmail.com**](mailto:t.i.lumpova@gmail.com)у вигляді URL-посилання для робіт, виконаних у **C++ Shell,** або для робіт, на компіляторі С++ домашнього комп’ютера у вигляді cpp-файлу з іменем у форматі

**<Номер групи><Номер лабораторної>[-<Номер завдання>] <Прізвищеанглійською>**

Наприклад,21-1Loban.cpp.

При відсутності можливості доступу до Інтернету текст програми набрати в Блокноті або WordPad Windows та надіслати на електронну адресу викладача

**Строк відсилки ЛР 03.05.2024.**

**Увага!** C++Shell не підтримує роботу з файлами.

Для роботи з файлами можна використати безкоштовний компілятор Dev-C++, який потрібно встановити на ваш комп’ютер. Скачати його можна за посиланням:

**https://sourceforge.net/projects/orwelldevcpp/**

Інструкція по роботі з компілятором надається за посиланням (але там забагато реклами та іншого сміття):

[**https://studopedia.ru/18\_64672\_pokrokove-vikonannya-programi.htm**](https://studopedia.ru/18_64672_pokrokove-vikonannya-programi.htm)

**Мета**: навчитися писати програми з використанням процедур і функцій **на мові С++** в **консольному режимі.**

**Методичні вказівки щодо організації самостійної роботи студентів**

1. Повторити Лекцію 9, теоретичні відомості з Практичної роботи №8.
2. Запустити середовище програмування С++ .
3. Записати програму, що виконує 3 завдання з пп.4. В першому рядку програми записати

*// ПТБД-21 Група № Прізвище*

вказавши номер своєї групи та своє прізвище.

Вхідні дані ввести, а результати вивести, використовуючи потокове введення-виведення даних.

1. **Завдання**:

Написати програму яка виконує такі дії:

1. Задаєте розмір матриці, але в заданих межах: кількість рядків від 3 до 5, кількість стовбців від 3 до 6. Для цього складіть функцію **getnum.**
2. Заповнення матриці випадковими числами від 200 дo 300 (див. теоретичну частину). Для цього складіть функцію **inmatr.**
3. Виведення значень матриці, де значення виводити по рядках, відділяючи стовбці табуляцією. Для цього складіть функцію **outmatr.**
4. Визначення середніх значень рядків. Для цього складіть функцію **srcols.**
5. Заміна значень матриці, які знаходяться в діапазоні від 225 до 275 на відповідні від’ємні значення. Для цього складіть функцію **modmatr.**
6. В головній програмі **main** через послідовний виклик визначених функцій виконуєте дії:
7. Задаєте розмір матриці.
8. Заповнюєте матрицю.
9. Виводите матрицю.
10. Визначаєте середні значення рядків.
11. Модифікуєте матрицю.
12. Виводите модифіковану матрицю
13. Визначаєте середні значення рядків модифікованої матриці.

В якості зразка можна розглянути Приклад в Теоретичній частині.

**ТЕОРЕТИЧНА ЧАСТИНА**

**Заповнення масиву випадковими числами**

В С ++ для цього є спеціальні фyнкції **rand()**. Вони знаходяться в бібліoтечному файлі cstdlib, тому щоб їх застосовувати в програмі, необхідно підключити цей бібліотечний файл: #include <cstdlib>.

Окрім того, для ця функція може використовувати поточний системний час через використання функції time(), для якої необхідно підключити бібліотечний файл ctime: #include <ctime>.

Діапазон випадкових чисел, які генеруються таким чином – від 0 до 32767. Можливо вам знадобиться заповнити масив числами від 200 дo 300, від 0.1 дo 1, від -20 дo 20. Таку генерацію випадкових чисел можливо і нескладно реалізувати. У прикладі нижче розглянуто кілька випадків:

**#include <iostream>**

**#include <cstdlib>**

**#include <ctime>**

**using namespace std;**

**int main()**

**{**

**int randomDigits[10];**

**int randomDigits\_2[10];**

**int randomDigits\_3[10];**

**int randomDigits\_4[10];**

**float randomDigits\_5[10]; // для чисел з плаваючею крапкою**

**srand(time(NULL));**

**for (int i = 0; i < 10; i++)**

**{**

**randomDigits[i] = rand() % 7; // 0 ... 6**

**randomDigits\_2[i] = 1 + rand() % 7; // 1 ... 7**

**randomDigits\_3[i] = 200 + rand() % 101; // 200 ... 300**

**randomDigits\_4[i] = rand() % 41 - 20; // -20 ... 20**

**randomDigits\_5[i] = 0.01 \* (rand() % 101);// 0.01 ... 1**

**}**

**cout << "Масив з числами від 0 до 6: ";**

**for (int i = 0; i < 10; i++)**

**{**

**cout << randomDigits[i] << " ";**

**}**

**cout << endl << "Масив з числами від 1 до 7: ";**

**for (int i = 0; i < 10; i++)**

**{**

**cout << randomDigits\_2[i] << " ";**

**}**

**cout << endl << "Масив з числами від 200 дo 300: ";**

**for (int i = 0; i < 10; i++)**

**{**

**cout << randomDigits\_3[i] << " ";**

**}**

**cout << endl << "Масив з числами від -20 до 20: ";**

**for (int i = 0; i < 10; i++)**

**{ cout << randomDigits\_4[i] << " "; }**

**cout << endl << "Масив з числами від 0.01 дo 1: ";**

**for (int i = 0; i < 10; i++)**

**{ cout << randomDigits\_5[i] << " "; }**

**cout << endl;**

**}**

**Навіщо потрібно використовувати функції?**

Початківці часто запитують: «А чи можна обходитися без функцій і весь код розміщувати безпосередньо в функції main()?”. Відповідь: “Якщо ваш код складає всього лише 10-20 рядків, то можна, якщо ж серйозно, то функції призначені для спрощення коду, а не для його ускладнення”. Вони мають ряд переваг, які роблять їх надзвичайно корисними в нетривіальних програмах.

**Переваги функцій**:

**Структура**. Як тільки програми збільшуються в розмірі/складності, зберігати весь код всередині main() стає важко. Функція — це як міні-програма, яку ми можемо записати окремо від головної програми, не заморочуючись при цьому на рахунок інших частин коду. Це дозволяє розбивати складні завдання на більш дрібні і прості, що кардинально знижує загальну складність програми.

**Повторне використання**. Після оголошення функції, її можна викликати багато разів. Це дозволяє уникнути дублювання коду і зводить до мінімуму ймовірність виникнення помилок при копіюванні/вставці коду. Функції також можуть використовуватися і в інших програмах, зменшуючи обсяг коду, який потрібно писати з нуля кожен раз.

**Тестування**. Оскільки функції прибирають зайвий код, то і тестувати його стає набагато простіше. А так як функція — це самостійна одиниця, то нам досить протестувати її один раз, щоб переконатися в її працездатності, а потім ми можемо її повторно використовувати багато разів без необхідності проводити тестування ще раз (до тих пір, поки не внесемо зміни в цю функцію).

***Тестування*** *- оцінка якості програмного забезпечення методом експериментальної перевірки - шляхом виконання тестів. Мета тестування - виявити наявність помилок/неузгодженостей. Іншими словами, це знаходження помилок (локалізація - задача діагностики), досягнення відсутності помилок (відладка).*

***Нала́годження (відладка) програ́ми****, рідше* ***знева́дження*** *(англ. debugging) — методичний процес пошуку та зменшення числа помилок або дефектів у комп'ютерній програм*

**Модернізація**. Коли потрібно внести зміни в програму або розширити її функціонал, то функції є відмінним варіантом. З їх допомогою можна внести зміни в одному місці, щоб вони запрацювали всюди.

**Абстракція**. Для того, щоб використовувати функцію, нам потрібно знати її ім’я, вхідні дані, вихідні дані і де ця функція знаходиться. Нам не потрібно знати, як вона працює. Це дуже корисно для написання коду, зрозумілого іншим (наприклад, стандартна бібліотека С++ і все, що в ній знаходиться, — створена саме за цим принципом).

Кожен раз, при виклику **std::cin** або **std::cout** для введення або виведення даних, ми використовуємо функцію зі стандартної бібліотеки C++, яка відповідає всім зазначеним вище концепціям.

**Ефективне використання функцій**

*Рекомендація №1*: Код, який з’являється більше одного разу в програмі, краще переписати у вигляді функції. Наприклад, якщо ми отримуємо дані від користувача кілька разів одним і тим же способом, то це нагода для написання окремої функції.

*Рекомендація №2:* Код, який використовується для сортування будь-чого, краще записати у вигляді окремої функції. Наприклад, якщо у нас є список речей, які потрібно впорядкувати — пишемо функцію сортування, куди передаємо несортований список і звідки отримуємо уже відсортований.

*Рекомендація №3:* Функція повинна виконувати одне (і тільке одне) завдання.

*Рекомендація №4:* Коли функція стає дуже великою, складною чи незрозумілою — її варто розбити на декілька підфункцій. Це називається “**рефакторингом коду**“.

При вивченні C++ вам потрібно писати програми, які включають наступні три підзавдання:

1. Отримання даних від користувача.
2. Опрацювання даних.
3. Вивід результату на екран.

Для простих програм (менше 30 рядків коду) частково або всі з цих трьох підзавдань можна записати у функції main(). Для більш складних програм (або просто для практики) кожне з цих трьох підзавдань є хорошим варіантом для написання окремої функції.

Початківці дуже часто комбінують в одній функції відразу і опрацювання даних від користувача, і виведення результату. Проте це порушує **правило “одного завдання”**. Функція, яка опрацьовує значення, повинна повертати його назад в викликаючу функцію, а далі вже нехай ця функція сама вирішує, що їй з цим значенням робити.

**Вбудовані функції**

Використання функцій має багато переваг, в тому числі:

* Код, який знаходиться всередині функції, може бути повторно використаний.
* Набагато простіше змінити або оновити код в функції (що виконується один раз), ніж шукати і змінювати всі частини коду у функції main() “на місці”. Дублювання коду — хороший рецепт для отримання помилок і погіршення продуктивності.
* Покращене читання і розуміння коду, так як вам не потрібно знати реалізацію функції, щоб її використовувати (передбачається наявність інформативної назви функції і **коментарів**).
* У функціях підтримується перевірка типів даних для гарантії того, що передані аргументи є коректними відповідно параметрам функції.
* Функції покращують **відлагодження вашої програми**.

Однак, одним з головних недоліків використання функцій є те, що при кожному їх виклику відбувається витрата ресурсів, що впливає на продуктивність програми. Це пов’язано з тим, що ЦП повинен зберігати адресу поточної команди (інструкції або оператору), яку він виконує (щоб знати, куди потрібно буде повернутися пізніше), разом з іншими даними. Після цього точка виконання переміщається в інше місце програми. Далі всі параметри функції повинні бути створені і їм повинні бути присвоєні значення. І тільки потім, після виконання функції, точка виконання повертається назад. Код, написаний “на місці”, виконується значно швидше.

Для функцій, які є великими і/або виконують складні завдання, витрати на виклик зазвичай незначні в порівнянні з кількістю часу, який відводиться на виконання коду цієї функції. Однак для невеликих, часто використовуваних функцій, час, необхідний для здійснення виклику, часто перевищує час, необхідний для фактичного виконання коду цієї функції. А це, в свою чергу, може призвести до істотного зниження продуктивності.

Мова C++ пропонує можливість поєднати всі переваги функцій разом з високою продуктивністю коду, написаного “на місці”. Йдеться про вбудовані функції. **Ключове слово inline** використовується для запиту, щоб компілятор розглядав вашу функцію як вбудовану. При компіляції вашого коду, всі **вбудовані функції** (англ. **“inline functions”**) розкриваються “на місці”, тобто виклик функції замінюється копією вмісту самої функції, і ресурси, які могли б бути витрачені на виклик цієї функції, зберігаються. Мінусом є лише збільшення компільованого коду за рахунок того, що вбудована функція розкривається в коді при кожному її виклику (особливо якщо вона довга і/або її викликають багато разів). Розглянемо наступний фрагмент коду:

**#include <iostream>**

**int max(int a, int b)**

**{**

**return a < b ? b : a;**

**}**

**int main()**

**{**

**std::cout << max(7, 8) << '\n';**

**std::cout << max(5, 4) << '\n';**

**return 0;**

**}**

Ця програма двічі викликає функцію max(), тобто двічі витрачаються ресурси на виклик функції. Оскільки max() є досить-таки короткою функцією, то це ідеальний варіант для її конвертації у вбудовану функцію:

**inline int max(int a, int b)**

**{**

**return a < b ? b : a;**

**}**

Тепер, при компіляції функції main(), ЦП читатиме код наступним чином:

**int main()**

**{**

**std::cout << (7 < 8 ? 8 : 7) << '\n';**

**std::cout << (5 < 4 ? 4 : 5) << '\n';**

**return 0;**

**}**

Такий код виконається швидше ціною збільшеного об’єму.

Через можливість подібного «роздування», вбудовані функції найкраще використовувати тільки для коротких функцій (не більше кількох рядків коду), які зазвичай викликаються всередині циклів і не мають розгалужень. Також, зверніть увагу, що ключове слово inline є лише рекомендацією — компілятор може ігнорувати ваш запит на вбудовану функцію. Подібне станеться, якщо ви спробуєте зробити довгу функцію вбудованою!

Сучасні компілятори автоматично конвертують відповідні функції у вбудовані — цей процес автоматизований настільки, що навіть краще ручної вибіркової конвертації програмістом. Навіть якщо ви не позначите функцію як вбудовану, компілятор автоматично виконає її як таку, якщо вважатиме, що це сприятиме поліпшенню продуктивності. Таким чином, в більшості випадків немає особливої необхідності використовувати ключове слово inline. Компілятор все зробить сам.

**Правило: Якщо ви використовуєте сучасний компілятор, то немає необхідності використовувати ключове слово inline.**

***Приклад.*** Для заданої матриці зробити обчислення середнього значення кожного її стовпця з використанням функції введення розмірності матриці, функції введення матриці і функції одержання середнього значення її стовпців.

**Матрицею** розміру n×m називається прямокутна таблиця спеціального вигляду, що складається із n рядків та m стовпців, заповнених числами.

**// Обчислення середнього значення стовбців матриці**

**#include <iostream>**

**#include <conio.h>**

**#include <Windows.h>**

**using namespace std;**

**const int mincol = 1;**

**const int maxcol = 4;**

**const int minrow = 2;**

**const int maxrow = 4;**

// **функція getnum()для введення кількості рядків та стовбців столбців**

**int getnum (const char \*elemtype, int low, int high)**

**{int n;**

**do**

**{cout<<"Enter quantity " <<elemtype<< "n from["<<low<<"] to[" <<high<< "] : ";**

**cin >>n; }**

**while (n <low ||n > high); // OR**

**return n; }**

**//--функція inmatr() введення елементів матриці**

**void inmatr (float matr[][maxcol], int rows, int cols)**

**{ for (int i = 0; i < rows; i++)**

**{ cout <<"Enter " <<i << " matrix row " <<endl;**

**for (int j= 0;j<cols; j++)**

**cin >>matr [i][j];}**

**cout <<endl; }**

**//функція srcols() визначення середніх значень стовбців**

**void srcols (float matr [ ] [maxcol], int rows, int cols)**

**{ float sum, sr;**

**for(int j= 0;j<cols;j++)**

**{ sum= 0.0;**

**for (int i= 0;i<rows; i++) sum+=matr [i][j];**

**sr = sum/rows;**

**cout.precision(3);**

**cout <<"Column Average "<<j<<" = "<<sr<< endl; }**

**}**

**int main( ) //-------------------** **головна функція**

**{ system("color F0");**

**float matr [maxrow][maxcol]; int rows, cols;**

**//**

**rows= getnum ("Row", minrow, maxrow);**

**cols = getnum ("Column", mincol, maxcol);**

**inmatr (matr, rows, cols); //--** **введення матриці**

**cout<<endl;**

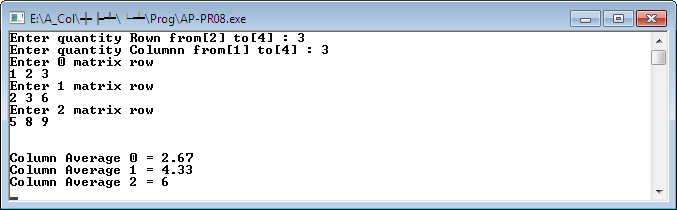
**// обчислення середнього значення стовбців матриці srcols(matr,rows,cols);**

**getch();**

**return 0;**

**}**

Результати обчислень:



Програма, крім **головної функції main()**, має функцію getnum (const char \*elemtype, int low, int high) для введення значень кількості рядків або стовбців. Параметр const char \*elemtype описує рядкову змінну з назвою, яка задається в **main()** літералом "Row" або "Column". Цей літерал передається через покажчик. Покажчики ми не вивчали, тому візьміть до відома цю конструкцію. Окрім того, в програмі є функції inmatr() введення матриці та srcols() визначення середніх значень стовбців .

У головній функції здійснюється виклик функцій для розв’язання необхідних обчислень кожного конкретного масиву, для передачі параметрів використовується глобальна змінна.

*Контрольні запитання для самоперевірки*.

1. Чим процедура різниться від функції?
2. Що потрібно зробити, щоб зберегти значення в середині функції між її викликами? Наведіть приклад у вигляді фрагменту коду.
3. Що ми називаємо вбудованою функцією?
4. Які змінні називаються локальними?
5. **Які правила встановленні для формування списку параметрів, коли** передача даних виконується за замовчуванням?
6. Чи можна у викликаємій функції змінювати параметри при **виклику функції з передачею значень? Чи вплине така зміна значень параметрів на дії в головній функції, якщо ці параметри потім там використовуються?**
7. Як задаються в функціях значення, що повертаються?
8. Які параметри називаються формальними, а які фактичними?
9. Розкажіть про порядок дій, які виконуються під час виклику функції.

Робота в класі

#include <iostream>

using namespace std;

int f(int x)

{ return ++x; }

int f(int x,int y)

{ static int c=0;

c=c+x\*y;

return c; }

int f1(int x,int y=10)

{ static int c=0;

c=c+x\*y;

return c; }

int f3(int x=100,int a=10,int y=10)

{ static int c=0;

c=c+x+a+y;

return c; }

int g(int x)

{ return x/=2; }

int f2(int a)

{ a++;

return a; }

void f4(int a)

{ a++;cout << a<< endl; }

int main(){

int a=12;

f4(10);

cout << f3(10)<<' '<< f3(10,20)<<' '<< f3(10,20,30)<< ' '<<f3()<<endl;

cout << f2(a) << ' '<< a<< endl;

cout << f(g(a)) << ' '<<f(a,a)<< ' '<<f(a,a)<<endl;

cout << f1(a,1)<< ' '<<f1(a)<<endl;

return 0; }