# 1.

1. Які існують види умовного оператора та чим вони відрізняються? Навести приклади застосування.

**Else,switch**

У мові с++ присутні 2 умовні оператори: else-if и switch;

Особливість else-if у тому, що він перевіряю деяку умову, яку ми задаємо у “()” і при виконанні цієї умови він завершую свою роботу. Але ж else-if не дуже підходить до розгалужених структур, для цього більш підходить оператор switch,особливістю якого є те,що він перевіряє умову та шукає відповідний кейс, який задовільняє умові та продовжує роботу після того, як знайшов відповідний кейс, доки не знайде ”break” або не дійде до останнього кейсу, який зазвичай зветься default;

2. Написати шаблонну функцію, яка повертає суму головної діагоналі. Написати програму, яка ілюструє роботу цієї функції.

#include <iostream>

using namespace std;

template < typename f >

f Suma(f matr[6][6])

{

f S = 0.0;

for (int i = 0; i < 6; i++)

S += matr[i][i];

return S;

}

int main()

{

double matr[6][6];

for (int i = 0; i < 6; i++)

{

for (int j = 0; j < 6; j++)

{

matr[i][j] = rand() %21 - 3.9;

cout << matr[i][j] << "\t";

}

cout << endl;

}

cout << Suma(matr) << endl;

system("pause");

return 0;

}

3. Обчислити  де *a*=-0,5; *b*=1,7; *t*=0,44.

#include <iostream>

#include <math.h>

using namespace std;

int main()

{

double a = -0.5;

double b = 1.7;

double t = 0.44;

double y = exp(-b \* t)\*sin(a\*t + b) - sqrt(abs(b\*t + a));

double s = b \* sin(a\*pow(t, 2)\*cos(2 \* t)) - 1;

cout << "a= " << a << "\tb= " << b << "\tt= " << t << endl << "y= " << y << "\ts= " << s << endl;

system("pause");

return 0;

}

4. Специфікатори функції fscanf? Приклад застосування.

Спецификаторы:

|  |  |
| --- | --- |
| %с | Считать один символ |
| %d | Считать десятичное число целого типа |
| %i | Считать десятичное число целого типа |
| %е | Считать число с плавающей запятой |
| %f | Считать число с плавающей запятой |
| %g | Считать число с плавающей запятой |
| %о | Считать восьмеричное число |
| %s | Считать строку |
| %х | Считать шестнадцатиричное число |
| %р | Считать указатель |
| %n | Принимает целое значение, равное количеству считанных до текущего момента символов |
| %u | Считывает беззнаковое целое |
| %[] | Просматривает набор символов |
| %% | Считывает символ % |

Пример:

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <iostream>

int main()

{

FILE \* pFile;

pFile = fopen("myfile.txt","w");

fprintf(pFile, "DIMKA");

fclose(pFile);

char str[100];

pFile = fopen("myfile.txt", "r");

fscanf(pFile, "%s",str);

std::cout << str<<std::endl;

system("pause");

fclose(pFile);

return 0;

}

# 2.

1 Напишіть програму сортування масиву по правилу зростання з використанням покажчиків.

#include <iostream>

using namespace std;

int main()

{

const int n = 7;

int mas[n];

for (int i = 0; i < n; i++)

{

mas[i] = rand() % 21 - 10;

cout << mas[i] << "\t";

}

int \*a = mas;

for (int i = 0; i < n - 1; i++)

for (int j = 0; j < n - i - 1; j++)

{

if (a[j] > a[j + 1])

{

int rab = a[j];

a[j] = a[j + 1];

a[j + 1] = rab;

}

}

cout << endl;

for (int i = 0; i < n; i++)

cout << mas[i] << "\t";

cout << endl;

system("pause");

return 0;

}

2 Як запрограмувати функції зі змінною кількістю параметрів? Навести приклад.

Для того, щоб запрограмувати ф-ію зі змінною кількістю параметрів потрібно або задати змінну, яка зберігає в собі обмежену кількість елементів та задати ії однією з перших у функцію, або ж можна задати ліміт останнім елементом у функції, “наступивши” на який функція перестане шукати елементи далі. Для того, щоб внести в функцію змінну кількість параметрів необхідно спочатку задати в функцію відомі значеня, а після них потрібно поставити ...  
всередині функції зміни наступні:  
ми повинні задати покажчик на перший з невідомих аргументів і при необхідності інкрементувати його.

#include <iostream>

using namespace std;

void f(int num, int first,...)

{

int \*p = &first;

while(num--)

{

cout << \*p << endl;

p++;

}

}

int main()

{

f(3, 5, 7, 8);

system("pause");

return 0;

}

3 Створити функцію знаходження добутку ненульових елементів, розташованих нижче головної діагоналі у квадратній матриці.

#include <iostream>

using namespace std;

#define n 6

int Dobutok(int matr[n][n])

{

int Dob = 1;

for (int i = 0; i < n; i++)

for (int j = 0; j < i; j++)

{

if (matr[i][j] != 0)

Dob \*= matr[i][j];

}

return Dob;

}

int main()

{

int matr[n][n];

for (int i = 0; i < n; i++)

{

for (int j = 0; j < n; j++)

{

matr[i][j] = rand()%21-10;

cout << matr[i][j] << "\t";

}

cout << endl;

}

cout << "Dobutok el. nizhe gl. diagonali = " << Dobutok(matr) << endl;

system("pause");

return 0;

}

4 Методи роботи з «прапорцями» потокових помилок. Навести приклад.

В базовом классе *ios* определено поле *state*, которое представляет собой состояние потока в виде совокупности битов:

enum io\_state {

googbit = 0x00, // Нет ошибок

eofbit = 0x01, // Достигнут конец файла

failbit = 0x02, // Ошибка форматирования или преобразования

badbit = 0x04, // Серьезная ошибка, после которой пользоваться потоком

// нельзя

hardbit = 0x08 // Неисправность оборудования

};

Состоянием потока можно управлять с помощью следующих методов и операций:

int rdstate ( ) – возвращает текущее состояние потока;

int eof ( ) – возвращает ненулевое значение, если установлен флаг eofbit;

int fail ( ) – возвращает ненулевое значение, если установлен один из флагов failbit, badbit или hardbit;

int good ( ) – возвращает ненулевое значение, если сброшены все флаги ошибок;

void clear (int = 0) – параметр принимается в качестве состояния ошибки, при отсутствии параметра состояние ошибки устанавливается 0;

operator void \*( ) – возвращает нулевой указатель, если установлен хотя бы один бит ошибки;

operator ! ( ) – возвращает ненулевой указатель, если установлен хотя бы один бит ошибки.

*Пример*. Наиболее часто используемые операции с флагами состояния потока:

// Проверить, установлен ли флаг flag

if (steam\_obj.rdstate ( ) & ios :: flag)

// Сбросить флаг flag

stream\_obj.clear (rdstate ( ) & ~ios :: flag)

// Установить флаг flag

stream\_obj.clear (rdstate ( ) | ios :: flag)

// Установить флаг flag и сбросить все остальные

stream\_obj.clear (ios :: flag)

// Сбросить все флаги

stream\_obj.clear ( )

Операция *void \*( )* неявно вызывается всякий раз, когда поток сравнивается с нулем. Это позволяет записывать циклы вида:

while (stream\_obj) {

// Все в порядке, можно производить ввод/вывод

}

# 3.

1. Які типи даних вам відомі? Навести приклади застосування.

Будь-яка константа, змінна, значення чи функція в **С++** характеризується своїм типом. Тип цих об'єктів визначає безліч припустимих значень, форму внутрішнього уявлення, а так само безліч припустимих операцій. Тому всі дані, що використовуються в програмі, до їхньої появи обов'язково повинні бути оголошені (описані).

Існують такі типи даних:

|  |  |  |
| --- | --- | --- |
| **Тип** | **Розмір, байт** | **Значення** |
| **Bool** | 1 | true (1) або false(0) |
| **unsigned short int** | 2 | від 0 до 65 535 |
| **short int** | 2 | від –32 768 до 32 767 |
| **unsigned long int** | 4 | від 0 до 4 294 967 295 |
| **long int** | 4 | від –2 147 483 648 до 2 147 483 647 |
| **int** (16 розрядів) | 2 | від –32 768 до 32 767 |
| **int** (32 розряда) | 4 | від –2 147 483 648 до 2 147 483 647 |
| **unsigned int** (16 розрядів) | 2 | від 0 до 65 535 |
| **unsigned int** (32 розряда) | 4 | від 0 до 4 294 967 295 |
| **Char** | 1 | від 0 до 256 |
| **Float** | 4 | від 1.2е-38 до 3.4е38 |
| **Double** | 8 | від 2.2е-308 до 1.8е308 |
| **long double** | змінний, не меньше 8 байт | від 2.2е-308 |
| **Void** | 2 або 4 | - |

Приклад застосування:

#include <iostream>

int main()

{

const double Pi = 3.1415;

float Rad = -865658.767; double Square;

const int s = 6;

const char str[] = "qwert";

std::cout <<str<<std::endl;

std::cout << s << std::endl;

std::cout << Rad << std::endl;

Square = Pi \* Pi;

std::cout << "Square = " << Square << std::endl; // 9.86902

std::cin.get();// затримка экрана

return 0;

}

2. Задана структура з ім’ям student, яка містить два поля: П.І.Б.; бал.

Необхідно реалізувати функцію сортування масиву структур за зростанням балу. Написати програму, яка ілюструє роботу цієї функції.

#include<iostream>

using namespace std;

struct student

{

char PIB[100];

int bal;

};

void sorttMas(student\*mas, int n)

{

for (int i = 0; i < n - 1; ++i)

for (int j = 0; j < n - 1 - i; ++j)

{

if (mas[j].bal > mas[j + 1].bal)

{ student c = mas[j];

mas[j] = mas[j + 1];

mas[j + 1] = c;

}}}

int main()

{ int n;

char str[100];

do

{ cout << "Please input n>0: ";

cin.getline(str, 99);

n = atoi(str);

} while (n <= 0);

student \* mas = new student[n];

for (int i = 0; i < n; i++)

{

cout << "\nInput PIB:\t";

cin.getline(mas[i].PIB, 99);

do

{

cout << "Input bal:\t";

cin.getline(str, 99);

mas[i].bal = atoi(str);

} while (mas[i].bal <= 0);

}

cout << endl;

for (int i = 0; i < n; ++i)

cout << mas[i].PIB << "\t" << mas[i].bal << endl;

cout << endl;

sorttMas(mas, n);

for (int i = 0; i < n; ++i)

{

cout << mas[i].PIB << "\t" << mas[i].bal << endl;

}

delete[]mas;

system("pause");

return 0;

}

3. Обчислити ;  де *a*=16,5; *b*=3,4; *x*=0,61.

#include<iostream>

using namespace std;

int main()

{

double a = 16.5, b = 3.4, x = 0.61, S, c;

c = (b\*pow(x,2) -a) / (exp(x\*a)-1); // -0.000648161

S = (((pow(x,3)\*pow(tan(pow((x+b),2)),2))+a) / (sqrt(x+b))) \* c; //-0.00535185

cout <<"c= "<< c <<"\tS= "<< S << endl; // c=-0.000648161 S=-0.00535185

system("pause");

return 0;

}

Відповідь: c = -0.000648161, S = -0.00535185 .

4. Функції введення/виведення С-рядків. Навести приклади буферного введення рядків.

Потік є програмною надбудовою над файлом, яка надає програмісту додатковий сервіс. Схема роботи з потоком така ж, як і з файлом: відкрити потік, виконати зчитування і/або запис, закрити потік.

Відкриває потік разом із асоційованим файлом функція

FILE\* fopen(const char \*filename, const char \*mode )

Зчитування рядка із потоку виконується функцією char \*fgets(char \*s, int n, FILE \*stream).

Зауваження. Функція gets вводить рядки зі стандартного вхідного потоку stdin. При введенні з клавіатури символ '\n' не потрапляє до рядка. Якщо вводити за допомогою функції fgets(buf, 100, F), то спеціального виведення fputs("\n", F) не знадобиться.

Перший параметр вказує на буфер, який приймає рядок, другий – задає розмір буфера. Функція повертає покажчик на буфер або 0 у разі помилки.

Зчитування припиняється, коли досягнуто кінець рядка або прочитано n-1 символів із файлу. Рядок в буфері замикається символами ‘\n’ і ‘\0’. Покажчик файлу переміщується за символи CR-LF.

Запис рядка в потік виконується функцією

char \*fputs(char \*s, FILE \*stream)

Функція повертає покажчик на останній записаний символ або EOF у разі помилки. Термінальний символ рядка не копіюється.

**Введення і виведення записів**

Зчитування записів з потоку виконується функцією

size\_t fread(void \*ptr, size\_t size, size\_t n, FILE \*stream)

ptr - покажчик на буфер в пам’яті, який приймає записи

size – розмір запису в байтах

n – кількість записів для зчитування

Функція повертає кількість прочитаних записів, яка може бути нульовою.

Зауваження: Тип size\_t визначений у файлі stdio.h як unsigned int.

Виведення записів у потік виконується функцією size\_t fwrite(const void \*ptr, size\_t size, size\_t n, FILE \*stream)

ptr - покажчик на буфер в пам’яті, в якому зберігаються записи

size – розмір запису в байтах

n – кількість записів для виведення

Функція повертає кількість виведених записів, яка може бути нульовою.

**Форматоване виведення**

Розглянуті вище функції виводять інформацію в потік без або майже без перетворення. Функція Функция int fprintf(FILE \*stream, const char \*format [, argument, ...]) перетворює дані для виведення [, argument, ...] в послідовність символів, керуючись рядком формату format. У разі успіху функція повертає кількість виведених в потік байтів, у разі помилки – константу EOF. Квадратні дужки говорять про необов'язковість аргументу.

Рядок формату містить прості символи і специфікації формату. Прості символи копіюються у вихідний потік без змін, специфікації застосовуються для форматування інших аргументів фунції. Якщо аргументів менше, ніж специфікацій, наслідки непередбачувані. Якщо аргументів більше, ніж специфікацій, зайві аргументи ігноруються.

**Форматоване введення**

Для форматованого введення з потоку застосовують функцію int fscanf(FILE \*stream, const char \*format [, address, ...])

Функція повертає число полів введення відформатованих і розміщених в пам’яті. У разі помилки функція повертає EOF.

Функція fscanf розглядає вхідний потік, як послідовність полів введення.

Функція переглядає послідовність полів введення, форматує їх і розміщує за адресами – аргументами fscanf. Число адрес, специфікацій формату і полів введення повинно бути узгоджено.

Рядок формату складається з символів, які не відображаються (' ',\t,\n), символів, які відображаються (всі інші крім ’%’) і специфікаторів формату. Якщо fscanf зустрічає символ, який не відображається, у форматному рядку, вона буде зчитувати, але не зберігати всі такі символи вхідного потоку аж до першого символу, який відображається. Якщо fscanf зустрічає символ, який відображається, у форматному рядку, вона зчитає, але не збереже відповідний символ вхідного потоку.

Приклади:

**Приклад 3.4** Вивести в текстовий файл таблицю множення на 5.

⇓

FILE \*out = fopen("c:\\xxx.txt","wt");

for (int i = 2; i < 10; i++)

fprintf(out, "%1d x 5 = %2d\n", i, i \* 5);

fclose(out);

**Приклад 7.5** Ввести рядки з клавіатури і зберегти їх в текстовому файлі.

⇓

char buf[100];

FILE \*F;

F = fopen("111.txt", "w");

for (;;) {

gets(buf);

if (strlen(buf) > 40)

break;

fputs(buf, F);

fputs("\n", F);

}

fclose(F);

**Приклад 7.7**. Вводити цілі числа з клавіатури і зберігати їх в бінарному файлі.

⇓

FILE \*F;

int buf;

F = fopen("111.dat", "w");

for (;;) {

scanf("%d", &buf);

if (!buf)

break;

fwrite(&buf, sizeof(int), 1, F);

}

fclose(F);

Зауваження. Для введення цілих чисел з вхідного потоку необхідно використовувати тільки форматне введення (scanf або fscanf.).

# 4.

2. Написати шаблонну функцію, яка повертає номер строки, в якій сума її елементів максимальна. Написати програму, яка ілюструє роботу цієї функції.

#include<iostream>

#include<stdlib.h>

using namespace std;

template <typename f>

f sraka\_motyka(f\*\* matr, f n)

{

f sum = 0;

f\* masx = new f[n];

f max = 0;

f maxi = 0;

for (int i = 0; i < n; ++i)

{

for (int j = 0; j < n; ++j)

{

sum += matr[i][j];

}

masx[i] = sum;

sum = 0;

}

for (int i = 0; i < n; ++i)

if (masx[i] > max)

{

maxi = i;

max = masx[i];

}

return maxi;

}

int main()

{

int n;

cout << " please input n:\tn = ";

cin >> n;

cout << endl;

int \*\*matr = new int\*[n];

for (int i = 0; i < n; i++)

matr[i] = new int[n];

for (int i = 0; i < n; ++i)

{

for (int j = 0; j < n; ++j)

matr[i][j] = rand() % 51 - 20;

}

cout << " massiv: \n";

for (int i = 0; i < n; ++i)

{

for (int j = 0; j < n; ++j)

cout << matr[i][j] << "; ";

cout << endl;

}

cout << endl;

cout << endl << " Nomer stroky z max sumoy = ";

int in = sraka\_motyka(matr, n);

cout << in << endl << endl;

system("pause");

return 0;

}

3. Обчислити ;  де *x*=1,82; *y*=18,5; *z*=-3,4

#include<iostream>

using namespace std;

int main()

{

double x = 1.82, y = 18.5, z = -3.4, g, f;

f = (x-y)\*((y-(z/(y-x)))/(1+pow((y-x),2))); // -1.11732

g = fabs(pow(x,(y/x))-pow((y/x),(1.0/3.0))); // 437.966

cout <<"f="<< f <<"\tg="<< g << endl; // f=-1.11732 g=437.966

system("pause");

return 0;

}

1. Яким чином можна ввести інформацію? Навести приклади.

**Введення – виведення даних.** При введенні - виведенні потоку всі дані розглядаються як потік окремих байтів. Функції введення - виведення для потоку дозволяють обробляти дані різних розмірів і форматів: від простого символу до великих структур даних. Програміст може використовувати функції чи бібліотеки, розробляти власні і включати їх у бібліотеку. Для доступу до бібліотеки цих класів треба включити в програму відповідні файли заголовків, наприклад:

**#include <iostream>**

За замовчуванням стандартні введення, виведення і виведення повідомлень про помилки відносяться до консолі користувача (клавіатура й екран). Це означає, що кожного разу, коли програма очікує введення зі стандартного потоку, дані повинні надходити з клавіатури, а якщо програма виводить, то на екран.

Найчастіше в **С++** використовується потокове введення даних, операції якого включені до складу класів **istream** чи **iostream**. Воно може здійснюватися з визначеним у цих класах вхідним потоком **std::cin** чи вхідним потоком, який визначив користувач. Для читання символів з цього потоку вказується операція витягу з потоку, що позначається за допомогою символів  **“ >> “**. Це перевантажена операція, визначена для всіх простих типів і покажчика на **char.**

Формат запису оператора **cin** має вигляд:

**std::cin [ >> values ];**

Наприклад, для введення значень змінним х і у можна написати:

**std::cin >> x >> y;**

Для форматованого введення даних використовується функція **scanf** з заголовного файлу <**stdio.h**>.Функція **scanf** має змінне число параметрів, але як фактичні параметри вона використовує адреси змінних, а не їхні значення. При цьому перед відповідним параметром ставиться знак **&** — символ взяття адреси змінної. Наприклад, **&x** означає "адреса змінної **x**", а не значення, що ця змінна має в даний момент. Рядок форматів функції **scanf** указує, які дані очікуються на вході. Якщо функція зустрічає у форматному рядку знак **%,** за яким стоїть знак перетворення, то вона буде пропускати на вході символи доти, поки не зустріне який – небудь непорожній символ. Форма запису цієї функції має вигляд:

**scanf ("**рядок форматних кодів**",** список імен змінних **);**.

Приклад 1: Обчислити значення площі круга за значенням радіуса, який вводиться з клавіатури.

#include <iostream>

int main()

{

const double Pi = 3.1415;

float Rad; double Square;

std::cout<<"Input radius, radius =";

std::cin>>Rad;

Square = Pi\*Rad\*Rad; // Square = Pi\*pow(Rad,2);

std::cout<<"Square = "<<Square<< std::endl;

std::cin.get();// затримка экрана

std::cin.get();

return 0;

}

Приклад 2: Ввести рядки з клавіатури і зберегти їх в текстовому файлі.

char buf[100];

FILE \*F;

F = fopen("111.txt", "w");

for (;;) {

gets(buf);

if (strlen(buf) > 40)

break;

fputs(buf, F);

fputs("\n", F);

}

fclose(F);

4. Специфікатори функції fopen. Навести приклад застосування.

Потік є програмною надбудовою над файлом, яка надає програмісту додатковий сервіс. Схема роботи з потоком така ж, як і з файлом: відкрити потік, виконати зчитування і/або запис, закрити потік.

Відкриває потік разом із асоційованим файлом функція

FILE\* fopen(const char \*filename, const char \*mode )

Функція отримує ім’я filename файлу, асоційованого з потоком і повертає покажчик на потік, який ідентифікує його в наступних операціях.

В рядку режимів mode можуть знаходитися наступні символи:

r — відкрити тільки для зчитування

w — створити для запису. Існуючий файл буде перевідкритий новим файлом.

a — відкрити для дописування або створити для запису, якщо файлу немає.

+ — операції будуть виконуватися з уже існуючим файлом.

t — текстовий режим (обробка символів CR-LF).

b — двійковий режим (ніякої обробки).

За відсутності у рядку символів b або t, режим визначається глобальною змінною \_fmode, яка визначена в заголовному файлі fcntl.h.

FILE – це управляюча структура для потоку, об’явлена в в stdio.h. Вона не призначена для прямого використання.

Окрім спільного відкриття потоку і файлу (fopen),можна відкрити потік і асоціювати його з уже відкритим файлом (fdopen), відкрити файл і асоціювати його з уже відкритим потоком (freopen).

Закриває потік і вивантажує буфери функція

int fclose ( FILE \*stream )

Функція повертає 0 при успішному завершенні її роботи і константу EOF у разі помилки.

Функція int fcloseall() закриває усі відкриті потоки, окрім стандартних: stdin, stdout, stdprn, stderr і stdaux.

Введення і виведення символів

Зчитування символу із потоку виконується функцією int fgetc(FILE \*stream).

Запис символу в стандартний потік stdout виконується функцією int fputchar(int c).

Приклад 3 Скопіювати файл xxx.bin у файл yyy.bin.

⇓

FILE \*in = fopen ("c:\\xxx.bin", "rt");

FILE \*out = fopen ("c:\\yyy.bin", "wt");

if (!in) return;

while (!feof(in))

fputc(fgetc(in), out);

fcloseall();

Приклад 7.2. Скопіювати текстовий файл xxx.txt у файл yyy.txt по рядках.

⇓

const int n = 100;

char buf [n];

FILE \*in = fopen("c:\\xxx.txt","r");

FILE \*out = fopen("c:\\yyy.txt","w");

if (!in) return;

while (!feof(in)) {

fgets(buf, n, in);

fputs(buf, out);

}

fcloseall();

# 5.

1. Яким чином можна вивести інформацію? Навести приклади.

В С++, как и в С, нет встроенных в язык средств ввода-вывода.  
  
В С для этих целей используется библиотека stdio.h.  
В С++ разработана новая библиотека ввода-вывода iostream, использующая концепцию объектно-ориентированного программирования:

#include <iostream>

Библиотека iostream определяет три стандартных потока:

* cin  стандартный входной поток (stdin в С)
* cout  стандартный выходной поток (stdout в С)
* cerr  стандартный поток вывода сообщений об ошибках (stderr в С)

Для их использования в Microsoft Visual Studio необходимо прописать строку:

using namespace std;  
Для выполнения операций ввода-вывода переопределены две операции поразрядного сдвига:

* >>  получить из входного потока
* <<  поместить в выходной поток

Вывод информации

**cout<<значение;**  
  
Здесь значение преобразуется в последовательность символов и выводится в выходной поток:

cout << n;

Возможно многократное назначение потоков:  
**cout << 'значение1' << 'значение2' << ... << 'значение n';**

int n;  
char j;

cin >> n >> j;  
cout << "Значение n равно" << n << "j=" << j;

Функция **printf** позволяет пользователю выводить информацию в консольное окно. Для использования [функции](http://proginfo.ru/functions/) необходимо подключить файл с описаниями с помощью директивы #include <stdio.h>

Описание функции имеет вид: int printf(char \*управляющая строка, ...)

Эта функция возвращает целое значение - количество выведенных символов в случае успешного вывода и отрицательное значение в противном случае. Аргумент "\*управляющая строка" - это строка, которая выводится в консольное окно. В этой строке могут быть указаны [спецификаторы](http://proginfo.ru/format-specifiers/) для вывода значений переменных и управляющие символы для форматированного вывода.

Основные спецификаторы:

* %d - десятичное целое (**int**),
* %f - вещественное число (**float**),
* %c - символ (**char**).

**fprintf** - функция для записи в файл форматированного текста.

Функция имеет вид:

int fprintf(FILE\* файловый\_поток, char\* выводимая\_в\_файл\_форматированная\_строка, ...);

Файловый поток перед использованием функции fprintf необходимо связать с файлом на диске и [открыть](http://proginfo.ru/fopen/) для записи.

FILE \* f;  {описание файловой переменной}

f= fopen("file.txt", "w");   {файловая переменная f связана с файлом file.txt на диске, w - спецификатор доступа к файлу, файл открыт для записи}

fprintf(f,"Таблица умножения \n %d\*%d=%d",2,2,4);

fclose(f); {Закрыть файл и освободить память}

Спецификаторы формата

Для функций [printf](http://proginfo.ru/printf/), [scanf](http://proginfo.ru/scanf/) существуют спецификаторы формата. Это условные обозначения, позволяющие считать из входящего потока определенную конструкцию и сохранить ее. Существует довольно большое число спецификаторов, но используют обычно лишь несколько из них.

Популярны, например, следующие спецификаторы:

* %с - символ,
* %s - [строка](http://proginfo.ru/strings/),
* %d - десятичное целое число,
* %a - шестнадцатеричное целое число,
* %f - вещественное число.

4. Принцип роботи з бінарними файлами. Навести приклад.

Для работы с **бинарными файлами** в языке C++ используется стандартная библиотека для работы с файлами, но с небольшими изменениями.

Так, например, для чтения бинарного файла применяется параметр «rb», а не обычный «r». Аналогично и с модификаторами записи «w», добавления «a», чтения и записи «r+», «w+», «a+»: «wb», «ab», «rb+», «wb+», «ab+». Как и для обычных файлов наличие буквы **w** обозначает, что файл будет создан в случае отсутствия, а, если файл уже есть, то его содержимое будет удалено и заменено новым. А наличие буквы **a** значит, что файл будет создан в случае отсутствия, но содержимое файла не будет уничтожено.

Весь процесс идет по аналогии с обычным текстовым файлом, но есть некоторые нюансы. Работа идет именно с битами, так что возможна запись или считывание различных управляющих символов, например, “\n”, который заканчивает строку и начинает новую.

В языке C++ иногда требуется считать из [бинарного файла](http://proginfo.ru/binary-file/)достаточно крупные объемы данных, при этом как-то классифицировав и разделив на переменные. Для этого прекрасно подходит одна из функций [библиотеки](http://proginfo.ru/header/) cstdio – **fread**.Эта же функция есть и в более старой библиотеке stdio.h.

#include <cstdio>

#include <cstdlib> // подключаем system("pause")

using namespace std;

int main()

{

FILE\* ptrFile = fopen( "file.txt" , "rb" );

char buffer[13];

fread(buffer, sizeof(char), 12, ptrFile);

fclose (ptrFile);

printf("%s",buffer);

system("pause");

}

Функция **fwrite** используется для записи [структур](http://proginfo.ru/struct/) в [бинарный файл](http://proginfo.ru/binary-file/). Эта функция определена в библиотеке cstdio, или более старой stdio.h.

Данная функция позволяет записывать любые структуры, главное указать размер таких элементов.

#include <stdio.h>

struct pair

{ char a;

char b;

};

int main ()

{

FILE \*ptrFile = fopen ("file.txt" , "wb" );

pair buffer[3] = {{'w' , 'a'},{'s' , 'c'},{'r' , 't'} };

fwrite(buffer , sizeof(char)\*2 ,3 , ptrFile );

fclose(ptrFile);

return 0;

}

3. Обчислити R=x2(x+1)/b-sin2(x-a); де a=0,7; b=0,05; x=0,43.



#include <iostream>

#include <math.h>

using namespace std;

int main()

{

double a = 0.7, b = 0.05, x = 0.43;

double R, S;

R = (pow(x, 2) \* (x + 1)) / (b - pow(sin(x - a), 2));

S = sqrt((x \* b) / a) + fabs(cos(pow(x + b, 3)));

cout << R << endl << S << endl;

system("pause");

return 0;

}

2. Створити функцію знаходження додатного мінімального елемента квадратної матриці.

#include <iostream>

using namespace std;

int MinMas(int \*\*matr, int x)

{

int min = matr[0][0];

for (int i = 0; i < x; i++)

{

for (int j = 0; j < x; j++)

{

if (min > matr[i][j])

{

if (matr[i][j] > 0)

{

min = matr[i][j];

}

}

}

}

return min;

}

int main()

{

int n = 5;

int \*\*mas2d = new int\*[n];

for (int i = 0; i < n; i++)

{

mas2d[i] = new int[n];

}

for (int i = 0; i < n; i++)

{

for (int j = 0; j < n; j++)

{

mas2d[i][j] = rand() % 21 - 10;

}

}

cout << "Origin mas:" << endl;

for (int i = 0; i < n; i++)

{

for (int j = 0; j < n; j++)

{

cout << mas2d[i][j] << " ";

}

cout << endl;

}

cout << endl;

int min = MinMas(mas2d, n);

cout << "Min = " << min << endl << endl;

system("pause");

return 0;

}

6.

1. Чим відрізняються змінні від констант? Навести приклади.

**Константа** — это величина, которая при выполнении программы остаётся неизменной.

**Переменная** — это ячейка памяти для временного хранения данных. Предполагается, что в процессе выполнения программы значения переменных могут изменяться.

Описание и инициализация переменных

**Змінна** — це об’єкт програми, що займає в загальному випадку декілька комірок пам’яті, та призначений зберігати дані. При оголошенні змінної для неї резервується де-яка область пам’яті, розмір якої залежить від конкретного типу змінної. Будь-яка константа, змінна, значення чи функція в **С++** характеризується своїм типом. Тип цих об'єктів визначає безліч припустимих значень, форму внутрішнього уявлення, а так само безліч припустимих операцій. Тому всі дані, що використовуються в програмі, до їхньої появи обов'язково повинні бути оголошені (описані).

Наприклад,

**bool dd = true;** — логічна змінна dd=1;

**bool dd1 = 1;** — логічна змінна dd1=true;

**int a = 1, b = 0;** — цілі змінні a = 1, b = 0;

**сhar sim =’A’;** — символьна змінна sim =’A’;

**float Age = 18.5;** — десяткова змінна Age = 18.5 з крапкою, що плаває;

**void MyFunction();** — функція не вертає ніякого значення, тощо.

Прежде чем использовать в программе какую-то переменную, надо дать ей **описание**, то есть сказать, какое имя имеет переменная и каков её тип. Вначале указывается тип переменной, а затем её имя. Например:

**int k; // это переменная целого типа int**

**double x; // это переменная вещественного типа удвоенной точности**

Если имеется несколько переменных одного типа, то допускается их описание через запятую в одном операторе, например:

**double a, b, c;**

После описания переменной её можно использовать. Переменная после описания имеет произвольное значение, то есть ей просто выделяется свободный участок памяти и на этом всё заканчивается. В переменной хранится та последовательность двоичных цифр, которая была там до выделения памяти под эту переменную.

Бьерн Страуструп рекомендует **инициализировать** **переменные**, то есть не просто выделять память под переменные, но и задавать им при этом необходимые значения. Например:

**double a=3, b=4, c=5;**

Инициализация переменных выполняется один раз на этапе компиляции, она не снижает скорость работы программы, но при этом уменьшает риск использования переменной, которая не получила ни какого значения.

Задание и использование констант

**Константи** відрізняються від змінних тим, що значення, що присвоєне константі первісно, не може бути змінено протягом виконання всієї програми. Оголошення констант виконується за наступною схемою:

сonst [тип] ідентифікатор = значення;

Наприклад:

**const double Pi = 3.1415;** // константа **Pi** принимает значение 3.1415;

**const a=1;** // по умолчанию тип **int**.

Все константы вне зависимости от типа данных можно подразделить на две категории: именованные константы и константы, которые не имеют собственного имени. Например:

**25** — константа целого типа;

**3.14** — вещественная константа;

**'A'** — символьная константа.

Все три приведённые здесь константы не имеют имени, они заданы своим внешним представлением и используются в программе непосредственно, например так:

**int k=25; // переменная k инициализирована константой — целым числом 25.**

В ряде случаев константе удобнее дать имя и использовать её далее по имени. Обычно это делается для математических или физических констант.

В языке C был единственный способ создания **именованных констант** — с помощью директивы препроцессора **#define**, например:

**#define PI 3.14**

**...............**

**double t;**

**t = PI \* 2; // здесь использована именованная константа PI, заданная выше**

В языке C++ появился ещё один способ — использование **константных переменных**, то есть переменных, которые нельзя изменять после инициализации. Рассмотрим на том же примере:

**const double PI=3.14; // здесь PI — константная переменная**

**double t;**

**t=PI \* 2;**

4. Специфікатори функції fscanf? Приклад застосування.

Синтаксис функции scanf ():

scanf ( "Строка формата", & аргумент1 [, & аргрумент2 [...]])

Синтаксис функции printf ():

printf ( "Строка формата" [, аргумент1 [, аргумент2 [...]]])

Первым параметром функции printf () является "строка формата", который задает форму вывода информации. Далее могут располагаться выражения арифметических типов или строки (в списке аргументов они отделяются запятыми). Функция printf () преобразует значения аргументов к виду, поданного в строке формата, "собирает" преобразованные значения в эту строку и выводит полученную последовательность символов в стандартный поток вывода. Строка формата состоит из объектов двух типов: обычных символов, из строки копируются в поток вывода, и спецификаций преобразования. Количество спецификаций в строке формата должно равняться количеству аргументов.

Отличие в синтаксисе функций printf () и scanf () заключается в особенностях списке аргументов. Функция printf () использует имена переменных, констант и выражения, в то время, как для функции scanf () указываются только указатели на переменные. Параметры этой функции обязательно должны быть указателями. Функция scanf () использует практически тот же набор символов спецификации, и функция printf ().

Спецификации преобразования для функции printf ()

% d десятичное целое

% i десятичное целое

% o восьмеричное целое без знака

% u десятичное целое без знака (unsigned)

% x шестнадцатеричное целое без знака

% f представление величин float и double с фиксированной точкой

% E или% Е экспоненциальный формат представления действительных величин

% g представления действительных величин как f или Е в зависимости от значений

% c один символ (char)

% s строка символов

% p указатель типа void \*

% n указатель типа int \*

% ld long (в десятичном виде)

% lo long (в восьмеричном виде)

% p вывода указателя в шестнадцатеричной форме

% lu unsigned long.

Можно расширить основное определение спецификации преобразования, поместив модификаторы между знаком% и символами, которые определяют тип преобразования.

3. Обчислити де x=1,25; y=0,93.



#include <iostream>

#include <math.h>

using namespace std;

int main()

{

double x = 1.25, y = 0.93;

double a, b, p;

a = (1 - y) \* ((pow(x + y, 2) / pow(x + 4, 3)) / (exp(-(x - 2)) + (pow(x, 3) + 4)));

b = (1 + cos(y - 2)) / (pow(x, 4 / 2) + pow(sin(y - 2), 2));

p = a / b;

cout << a << endl << b << endl << p << endl;

system("pause");

return 0;

}

2. Розробити функцію, яка для довільної матриці знаходить мінімуми в рядках, а серед них – максимум.

#include <iostream>

using namespace std;

int MaxSumRyadMas(int \*\*matr, int x, int y)

{

int\* mas = new int[x];

for (int i = 0; i < x; i++)

{

int min = matr[i][0];

for (int j = 0; j < y; j++)

{

if (min > matr[i][j])

{

min = matr[i][j];

}

}

mas[i] = min;

}

int max = mas[0];

for (int i = 0; i < x; i++)

{

if (max < mas[i])

{

max = mas[i];

}

}

return max;

}

int main()

{

int n = 5, m = 6;

int \*\*mas2d = new int\*[n];

for (int i = 0; i < n; i++)

{

mas2d[i] = new int[m];

}

for (int i = 0; i < n; i++)

{

for (int j = 0; j < m; j++)

{

mas2d[i][j] = rand() % 21 - 10;

}

}

cout << "Origin mas:" << endl;

for (int i = 0; i < n; i++)

{

for (int j = 0; j < m; j++)

{

cout << mas2d[i][j] << " ";

}

cout << endl;

}

cout << endl;

int maxs = MaxSumRyadMas(mas2d, n, m);

cout << "Max sredy min ryadok = " << maxs << endl << endl;

system("pause");

return 0;

}

# 7.

1. Розказати про структуру простішої програми мовою С++. Навести приклади.

Программа на С++ состоит из набора инструкций. Каждая инструкция (statement) выполняет определенное действие. В конце инструкции в языке C++ ставится точка с запятой (;). Данный знак указывает компилятору на завершение инструкции. Например:

std::cout << "Hello World!";

Данная строка выводит на консоль строку "Hello world!", является инструкцией и поэтому завершается точкой с запятой.

Набор инструкций может представлять блок кода. Блок кода заключается в фигурные скобки, а инструкции помещаются между открывающей и закрывающей фигурными скобками:

{

std::cout << "Hello World!";

std::cout << "Bye World!";

}

В этом блоке кода две инструкции, которые выводят на консоль определенную строку.

**Функция main**

Каждая программа на языке С++ должна иметь как минимум одну функцию - функцию main(). Именно с этой функции начинается выполнение приложения. Ее имя main фиксировано и для всех программ на Си всегда одинаково.

Функция также является блоком кода, поэтому ее тело обрамляется фигурными скобками, между которыми определяется набор инструкций.

#include <iostream> // подключаем заголовочный файл iostream

int main() // определяем функцию main

{ // начало функции

std::cout << "Hello World!"; // выводим строку на консоль

return 0; // выходим из функции

} // конец функции

Определение функции main начинается с возвращаемого типа. Функция main в любом случае должна возвращать число. Поэтому ее определение начинается с ключевого слова int.

Далее идет название функции, то есть main. После названия в скобках идет список параметров. В данном случае функция main не принимает никаких параметров, поэтому после названия указаны пустые скобки. Однако есть другие варианты определения функции main, которые подразумевают использование параметров. В частности, нередко может встречаться следующее определение функции main, использующей параметры:

int main (int argc, char \*argv[])

{

}

И после списка параметров идет блок кода, который и содержит в виде инструкций собственно те действия, выполняемые функцией main.

**Директивы препроцессора**

В примере выше на консоль выводится строка, но чтобы использовать вывод на консоль, необходимо в начале файла с исходным кодом подключать библиотеку iostream с помощью директивы include.

Директива include является директивой препроцессора. Каждая директива препроцессора размещается на одной строке. И в отличие от обычных инструкциий языка C++, которые завершаются точкой с запятой ; , признаком завершения препроцессорной директивы является перевод на новую строку. Кроме того, директива должна начинаться со знака решетки #. Непосредственно директива "include" определяет, какие файлы и библиотеки надо подвключить в данном месте в код программы.

**Комментарии**

Исходный код может содержать комментарии. Комментарии позволяют понять смыл программы, что делают те или иные ее части. При компиляции комментарии игнорируются и не оказывают никакого влияние на работу приложения и на его размер.

В языке C++ есть два типа комментариев: однострочный и многострочный. Однострочный комментарий размещается на одной строке после двойного слеша //:

#include <iostream> // подключаем библиотеку iostream

int main() // определяем функцию main

{ // начало функции

std::cout << "Hello World!"; // выводим строку на консоль

return 0; // выходим из функции

} // конец функции

Многострочный комментарий заключается между символами /\* текст комментария \*/. Он может размещаться на нескольких строках. Например:

#include <iostream>

/\*

Определение функции Main

Выводит на консоль строку Hello World!

\*/

int main()

{

std::cout << "Hello World!"; // вывод строки на консоль

return 0;

}

2. Створити функцію знаходження суми діагоналей квадратної матриці.

[#include](tg://search_hashtag/?hashtag=include) <iostream>  
[#include](tg://search_hashtag/?hashtag=include) <cstdlib>  
[#include](tg://search_hashtag/?hashtag=include) <time.h>  
  
using namespace std;  
  
int Sum(int\*\* mas,int n)  
{  
 int s=0;  
 for (int i=0;i<n;i++)  
 {  
 s+=mas[i][i]+mas[n-i-1][i];  
 }  
 return s;  
}  
  
void Fill(int\*\* mas,int n)  
{  
 for (int i=0;i<n;i++)  
 {  
 for(int j=0;j<n;j++)  
 {  
 mas[j][i]=rand()%10;  
 }  
 }  
}  
  
void Print(int\*\* mas,int n)  
{  
 for (int i=0;i<n;i++)  
 {  
 for(int j=0;j<n;j++)  
 {  
 cout<<mas[j][i]<<"\t";  
 }  
 cout<<endl;  
 }  
}  
  
int main()  
{  
 srand(time(0));  
 int n;  
 int\*\* arr;  
 cin>>n;  
 arr=new int\*[n];  
 for (int i=0;i<n;i++) arr[i]=new int[n];  
 Fill(arr,n);  
 Print(arr,n);  
 cout<<"Sum = "<<Sum(arr,n);  
 return 0;  
}

3. Обчислити R=x^3(x+1)/b^2 – sin^2(x(x+a)); S=Sqrt(xb/a)+cos^2(x+b)^3 де a=0,7; b=0,05; x=0,5.

[#include](tg://search_hashtag/?hashtag=include) <iostream>  
[#include](tg://search_hashtag/?hashtag=include) <math.h>  
using namespace std;  
int main()  
{  
 double a=0.7;  
 double b=0.05;  
 double x=0.5;  
 double S=sqrt(x\*b/a)+pow(cos(pow(x+b,3)),2);  
 double R=pow(x,3)\*(x+1)/(b\*b)-pow(sin(x\*(x+a)),2);  
 cout<<"R = "<<R<<"\tS = "<<S;  
 return 0;  
}

4. Для чого застосовуються директиви препроцесора? Навести приклади.

Директивой (командной строкой) препроцессора называется строка в исходном коде, имеющая следующий формат: #ключевое\_слово параметры:

* символ #;
* ноль или более символов пробелов и/или табуляции;
* одно из предопределённых ключевых слов;
* параметры, зависимые от ключевого слова.

Список ключевых слов:

* define — создание константы или макроса;
* undef — удаление константы или макроса;
* include — вставка содержимого указанного файла;
* if — проверка истинности выражения;
* ifdef — проверка существования константы или макроса;
* ifndef — проверка не существования константы или макроса;
* else — ветка условной компиляции при ложности выражения if;
* elif — проверка истинности другого выражения; краткая форма записи для комбинации else и if;
* endif — конец ветки условной компиляции;
* line — указание имени файла и номера текущей строки для компилятора;
* error — вывод сообщения и остановка компиляции;
* warning — вывод сообщения без остановки компиляции;
* pragma — указание действия, зависящего от реализации, для препроцессора или компилятора;

если ключевое слово не указано, директива игнорируется;

если указано несуществующее ключевое слово, выводится сообщение об ошибке и компиляция прерывается. (В некоторых компиляторах, Таких как g++, компиляция продолжается, просто показывая предупреждение)

Препроцессор С/С++ (англ. pre processor, предобработчик) — программа, подготавливающая код программы на языке C/C++ к компиляции.

Препроцессором выполняются следующие действия:

* замена соответствующих диграфов и триграфов на эквивалентные символы «#» и «\»;
* удаление экранированных символов перевода строки;
* замена строчных и блочных комментариев пустыми строками (с удалением окружающих пробелов и символов табуляции);
* вставка (включение) содержимого произвольного файла (#include);
* макроподстановки (#define);
* условная компиляция (#if, #ifdef, #elif, #else, #endif);
* вывод сообщений (#warning, #error).

Условная компиляция позволяет выбрать код для компиляции в зависимости от:

* модели процессора (платформы);
* разрядности адресов;
* размерности типов;
* наличия/отсутствия поддержки расширений языка;
* наличия/отсутствия библиотек и/или функций;
* особенностей поведения конкретных функций;
* и другого.

Примеры

1

#include <iostream>

#define TEXT "Марс" // определение константы

int main()

{

std::cout << TEXT;

return 0;

}

2

#ifndef PRODUCT\_H

#define PRODUCT\_H

class Product

{

// код класса...

};

#endif PRODUCT\_H

3

#define E 2.71828 // раннее определенный макрос

int sumE = E + E; // обращение к макросу

#undef E // теперь E - не макрос

# 8.

1. Що таке розадресування у мові С++? Навести приклади.

Эти операции используются для работы с переменными типа указатель.

Операция разадресации (\*) осуществляет косвенный доступ к адресуемой величине через указатель. Операнд должен быть указателем. Результатом операции является величина, на которую указывает операнд. Типом результата является тип величины, адресуемой указателем. Результат не определен, если указатель содержит недопустимый адрес.

int t, f=0, \* adress;

adress = &t /\* переменной adress, объявляемой как

указатель, присваивается адрес переменной t \*/

\* adress =f; /\* переменной находящейся по адресу, содержащемуся

в переменной adress, присваивается значение

переменной f, т.е. 0 , что эквивалентно

t=f; т.е. t=0; *\*/*

2. Створити функцію, яка у масиві обнуляє елементи, що знаходяться між максимумом та мінімумом.

[#include](tg://search_hashtag/?hashtag=include) <iostream>  
[#include](tg://search_hashtag/?hashtag=include) <cstdlib>  
[#include](tg://search_hashtag/?hashtag=include) <time.h>  
using namespace std;  
int MinPos(int\* arr,int n){  
 int m=arr[0];  
 int mpos=0;  
 for(int i=1;i<n;i++)  
 { if (m>arr[i])  
 { m=arr[i];  
 mpos=i;  
 }  
 }  
 return mpos;  
}  
int MaxPos(int\* arr,int n)  
{ int m=arr[0];  
 int mpos=0;  
 for(int i=1;i<n;i++)  
 { if (m<arr[i])  
 { m=arr[i];  
 mpos=i;  
 }  
 }  
 return mpos;  
}  
  
void Annul(int\* arr,int k,int l)  
{  
 for (int i=k+1;i<l;i++)  
 {  
 arr[i]=0;  
 }  
}  
  
void PrintArr(int\* arr,int n)  
{  
 for (int i=0;i<n;i++) cout<<arr[i]<<"\t";  
}  
  
void SetArr(int\* arr,int n)  
{  
 for (int i=0;i<n;i++) arr[i]=rand()%50+1;  
}  
  
int main()  
{  
 srand(time(0));  
 int\* arr;  
 int n;  
 cin>>n;  
 arr=new int[n];  
 SetArr(arr,n);  
 PrintArr(arr,n);  
 int a=MinPos(arr,n);  
 int b=MaxPos(arr,n);  
 if (a>b) Annul(arr,b,a);  
 else Annul(arr,a,b);  
 cout<<endl<<endl;  
 PrintArr(arr,n);  
 return 0;  
}

3. Обчислити де m=2; c=-1; x=1,2; b=0,7.



[#include](tg://search_hashtag/?hashtag=include) <iostream>  
[#include](tg://search_hashtag/?hashtag=include) <math.h>  
  
using namespace std;  
  
int main()  
{  
 double m=2.0;  
 double c=-1.0;  
 double x=1.2;  
 double b=0.7;  
 double Fi=sqrt(m\*tan(x)+fabs(c\*sin(x)));  
 double E=m\*cos(b\*x+sin(x))+pow(c,3);  
 cout<<"F = "<<Fi<<"\nE = "<<E;  
 return 0;  
}

4. Що таке макропідстановка? Навести приклади.

!ВНИМАНИЕ!

Макроподстановка расписана ужасно, ибо мне не удалось (влом) найти инфу в нормальном виде. Пожалуйста, самостоятельно воспользуйтесь гуглом

1) Управляющая строка следующего вида заставляет препроцессор заменять идентификатор на последовательность лексем везде далее по тексту программы:

#define идентификатор последовательность\_лексем

При этом символы пустого пространства в начале и в конце последовательности лексем выбрасываются. Повторная строка #define с тем же идентификатором считается ошибкой, если последовательности лексем не идентичны (несовпадения в символах пустого пространства не играют роли).

2) Строка следующего вида, где между первым идентификатором и открывающей круглой скобкой не должно быть символов пустого пространства, представляет собой макроопределение с параметрами, задаваемыми списком-идентификаторов.

#define идентификатор(список\_идентификаторов)последовательность\_лексем

Как и в первой форме, символы пустого пространства в начале и в конце последовательности лексем выбрасываются, и макрос может быть повторно определен только с идентичным по количеству и именам списком параметров и с той же последовательностью лексем.

Управляющая строка следующего вида приказывает препроцессору «забыть» определение, данное идентификатору:

#undef идентификатор

* Применение директивы #undef к не определенному ранее идентификатору не считается ошибкой.
* Если макроопределение было задано во второй форме, то любая следующая далее в тексте программы цепочка символов, состоящая из идентификатора макроса (возможно, с последующими символами пустого пространства), открывающей скобки, списка лексем, разделенных запятыми, и закрывающей скобки, представляет собой вызов макроса.
* Аргументами вызова макроса являются лексемы, разделенные запятыми, причем запятые, взятые в кавычки или вложенные круглые скобки, в разделении аргументов не участвуют.
* (!)Во время группировки аргументов раскрытие макросов в них не выполняется.
* Количество аргументов в вызове макроса должно соответствовать количеству параметров макроопределения.
* После выделения аргументов из текста символы пустого пространства, окружающие их, отбрасываются.
* Затем в замещающей последовательности лексем макроса каждый идентификатор-параметр, не взятый в кавычки, заменяется на соответствующий ему фактический аргумент из текста.
* (!)Если в замещающей последовательности перед параметром не стоит знак #, если и ни перед ним, ни после него нет знака ##, то лексемы аргумента проверяются на наличие в них макровызовов; если таковые есть, то до подстановки аргумента в нём выполняется раскрытие соответствующих макросов.

На процесс подстановки влияют два специальных знака операций.

* Во-первых, если перед параметром в замещающей строке лексем вплотную стоит знак #, то вокруг соответствующего аргумента ставятся строковые кавычки ("), а потом идентификатор параметра вместе со знаком # заменяется получившимся строковым литералом.
* Перед каждым символом " или \, встречающимся вокруг или внутри строковой или символьной константы, автоматически вставляется обратная косая черта.
* Во-вторых, если последовательность лексем в макроопределении любого вида содержит знак ##, то сразу после подстановки параметров он вместе с окружающими его символами пустого пространства отбрасывается, благодаря чему сцепляются соседние лексемы, образуя тем самым новую лексему.
* Результат не определён при генерировании таким образом недопустимых лексем языка или в случае, когда получающийся текст зависит от порядка применения операции ##.
* Кроме того, знак ## не может стоять ни в начале, ни в конце замещающей последовательности лексем.
* (!)В макросах обоих видов замещающая последовательность лексем повторно просматривается в поиске новых define-идентификаторов.
* (!)Однако если какой-либо идентификатор уже был заменен в текущем процессе раскрытия, повторное появление такого идентификатора не вызовет его замены; он останется нетронутым.
* (!)Даже если развернутая строка макровызова начинается со знака #, она не будет воспринята как директива препроцессора.

Восклицательным знаком (!) отмечены правила, отвечающие за рекурсивные вызов и определения.

Пример раскрытия макроопределения[править | править код]

#define cat( x, y ) х ## у

Вызов макроса «cat(var, 123)» будет заменён на «var123». Однако вызов «cat(cat(1, 2), 3)» не даст желаемого результата. Рассмотрим шаги работы препроцессора:

0: cat( cat( 1, 2 ), 3 )

1: cat( 1, 2 ) ## 3

2: cat( 1, 2 )3

Операция «##» помешала правильному раскрытию аргументов второго вызова «cat». В результате получилась следующая цепочка лексем:

cat ( 1 , 2 )3

где «)3» — результат сцепления последней лексемы первого аргумента с первой лексемой второго аргумента, не является допустимой лексемой.

Можно задать второй уровень макроопределения в таком виде:

#define xcat( x, y ) cat( x, y )

Вызов «xcat(xcat(1, 2), 3)» будет заменён на «123». Рассмотрим шаги работы препроцессора:

0: xcat( xcat( 1, 2 ), 3 )

1: сat( xcat( 1, 2 ), 3 )

2: cat( cat( 1, 2 ), 3 )

3: cat( 1 ## 2, 3 )

4: cat( 12, 3 )

5: 12 ## 3

6: 123

Всё прошло благополучно, потому что в раскрытии макроса «xcat» не участвовал оператор «##».

# 9.

1. Які дії можна виконувати над покажчиками? Навести приклади.

Над указателями можно выполнять унарные операции: инкремент и декремент. При выполнении операций ++ и — значение указателя увеличивается или уменьшается на длину типа, на который ссылается используемый указатель.

Пример:

int \*ptr, a[10];

ptr=&a[5];

ptr++; // равно адресу элемента а[6]

ptr--; //равно адресу элемента а[5]

В бинарных операциях сложения и вычитания могут участвовать указатель и величина типа int. При этом результатом операции будет указатель на исходный тип, а его значение будет на указанное число элементов больше или меньше исходного.

Пример:

int \*ptr1, \*ptr2, a[10];  
 int i=2;  
 ptr1=a+(i+4); */\* равно адресу элемента a[6] \*/*  
 ptr2=ptr1-i; */\* равно адресу элемента a[4] \*/*

В операции вычитания могут участвовать два указателя на один и тот же тип. Результат такой операции имеет тип int и равен числу элементов исходного типа между уменьшаемым и вычитаемым, причем если первый адрес младше, то результат имеет отрицательное значение.

Пример:

int \*ptr1, \*ptr2, a[10];  
 int i;  
 ptr1=a+4;  
 ptr2=a+9;  
 i=ptr1-ptr2; */\* равно 5 \*/*  
 i=ptr2-ptr1; */\* равно -5 \*/*

Значения двух указателей на одинаковые типы можно сравнивать в операциях ==, !=, <, <=",">, >= при этом значения указателей рассматриваются просто как целые числа, а результат сравнения равен 0 (ложь) или 1 (истина).

int \*ptr1, \*ptr2, a[10];  
 ptr1=a+5;  
 ptr2=a+7;  
 if (prt1>ptr2) a[3]=4;

В данном примере значение ptr1 меньше значения ptr2 и поэтому оператор a[3]=4 не будет выполнен.

2. Створити дві функції: перша сортує елементи матриці за зростанням для кожного стовпця; друга находить мінімальний елемент цієї матриці.

#include <iostream>

using namespace std;

int\*\* createMatr(int n, int m)

{

if (n < 1) return NULL;

int\*\* matr = new int\*[n];

for (int i = 0; i < n; ++i)

matr[i] = new int[m];

return matr;

}

void randMatr(int\*\* matr, int n, int m)

{

for (int i = 0; i < n; ++i)

for (int j = 0; j < m; ++j)

matr[i][j] = rand() % 21 - 10;

}

void outputMatr(int\*\* matr, int n, int m)

{

for (int i = 0; i < n; ++i, cout << endl)

for (int j = 0; j < m; ++j)

cout << matr[i][j] << "\t";

cout << endl;

}

void deleteMatr(int\*\*matr, int n)

{

for (int i = 0; i < n; ++i)

delete[] matr[i];

delete[] matr;

matr = NULL;

}

void SortStolb(int\*\* mas, int n, int m)

{

for (int j = 0; j < m; j++)

for (int i = 0; i < n - 1; i++)

for (int k = 0; k < n - i - 1; k++)

if (mas[k][j] > mas[k + 1][j])

{

int c = mas[k][j];

mas[k][j] = mas[k + 1][j];

mas[k + 1][j] = c;

}

}

int findMin(int\*\* mas, int n, int m)

{

int min = mas[0][0];

for (int i = 0; i < n; ++i)

for (int j = 0; j < m; ++j)

if (mas[i][j] < min)

{

min = mas[i][j];

}

return min;

}

int main()

{

const int n = 5, m = 6;

int\*\* mas2d = createMatr(n, m);

randMatr(mas2d, n, m);

cout << "Original matrix:" << endl;

outputMatr(mas2d, n, m);

SortStolb(mas2d, n, m);

cout << "Result matrix:" << endl;

outputMatr(mas2d, n, m);

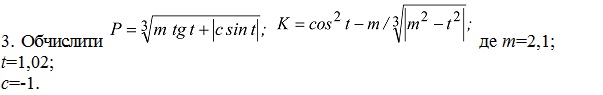
int min = findMin(mas2d, n, m);

cout << "Min = " << min << endl << endl;

deleteMatr(mas2d, n);

system("pause");

return 0;

}3. 

#include <iostream>

#include <math.h>

using namespace std;

int main()

{

double m = 2.1, t = 1.02, c = -1;

double P, K;

P = pow ((m \* tan(t)+(fabs (c \* sin(t)))), 1/3);

K = pow(cos(t), 2) - m/pow(fabs((pow(m,2) - pow(t,2))), 1/3);

cout <<"P="<< P << endl <<"K="<< K << endl;

system("pause");

return 0;

}

4. Що таке директива умовної компіляції? Навести приклади.

Директиви умовної компіляції

Директиви умовної компіляції #if, #ifdef і #ifndef застосовуються для того, щоб виключити компіляцію окремих частин програми. Це буває корисно при налагодженні або, наприклад, для підтримки декількох версій програми для різних платформ.

Формат директив #if:

#if константний\_вираз

[ #elif константний\_вираз

[ #elif константний\_вираз

...]

[ #e1se

…]

#endif

Кількість директив #elif - довільна. Блоки, що виключають код, можуть містити як опис, так і виконувані оператори. Приклад умовного включення різних версій заголовного файлу:

#if VERSION == 1

#define INCFILE "versl.h"

#elif VERSION == 2

#define INCFILE "vers2.h" /\* і так далі \*/

#else

#define INCFILE "versN.h"

#endif

#include INCFILE

Інше призначення директиви - тимчасово закоментувати фрагменти коду, наприклад:

#if 0

int i, j;

double x, у;

#endif

Оскільки допускається вкладеність директив, такий спосіб досить зручний.

Інший спосіб умовної компіляції заснований на застосуванні директив #ifdef, #ifndef, що означають "if defined" (якщо визначено) та "if not defined" (якщо не визначено). Загальний вид директиви #ifdef такий:

#ifdef ім'я\_макросу

послідовність операторів

#endif

Якщо ім'я макросу раніше визначено за допомогою директиви #define, компілюється відповідний блок коду.

Директива #ifndef має наступний загальний вигляд:

#ifndef ім`я\_макросу

послідовність операторів

#endif

Якщо ім'я макросу не було визначено за допомогою директиви #define, компілюється відповідний блок коду.

Директиви #ifdef і #ifndef можуть використовувати директиви #else і #elif.

Наприклад, наступний фрагмент програми виводить на екран рядки "Привіт, Влад" і "Марка немає вдома". Однак, якщо іменована константа VLAD не визначена, на екран виводиться повідомлення "Всім привіт", а потім - "Марка немає вдома".

#include <iostream>

#include <stdio.h>

#define VLAD 10

int main(void)

{

#ifdef VLAD

printf("Привіт, Влад\п");

#else

printf("Всім привіт\п");

#endif

#ifndef MARK

printf("Марка немає вдома\п");

#endif

return 0;

}

Директиви #ifdef і #ifndef можуть бути вкладеними, причому стандарт мови С допускає як мінімум вісім рівнів вкладення, а стандарт мови C ++ - до 256.

За допомогою цієї директиви зазвичай перевіряють повторюваність підключення деякого модулю (заголовочного файлу). Для цього необхідно, щоб у кожному заголовному файлі була наступна конструкція, де ім’я макропідстановки має бути унікальним. У нашому прикладі застосовується ім’я \_HEADER\_INCLUDED\_:

#ifndef \_HEADER\_INCLUDED\_

#define \_HEADER\_INCIUDED\_

Текст модулю

#endif

Наприклад, розглянемо 2 файли util.h та source.cpp

Текст файлу util.h:

#ifndef \_UTIL\_

#define \_UTIL\_

int sum(int a, int b) { return a + b; }

#endif

Текст файлу source.cpp:

#include "util.h"

#include "util.h"

int main()

{

int x = sum(5, 6);

return 0;

}

Повторне підключення файлу "util.h" не призводить до помилки, тому що в ньому наявна перевірка, яка дозволяє уникнути повторного створення функції sum.

Крім директиви #ifdef, є ще один спосіб перевірити, чи визначено ім'я макросу. Для цього можна застосовувати директиву #if в поєднанні зі статичним оператором defined, виконуваним на етапі компіляції. Оператор defined має наступний загальний вид:

defined ім`я\_макросу

Якщо в даний момент ім'я макросу є визначеним, вираз має справжнє значення. В іншому випадку він помилковий. Наприклад, щоб розпізнати, чи є іменована константа MYFILE визначеною, можна використовувати одну з наступних команд препроцесора:

#if defined MYFILE

чи

#ifdef MYFILE

Поставивши перед оператором defined символ !, можна задати протилежну умову. Наприклад, наступний фрагмент програми компілюється тільки в тому випадку, якщо іменована константа DEBUG не визначена.

#if !defined DEBUG

printf("Final version!\n");

#endif

Одна з причин, чому використовується оператор defined, полягає в тому, що він дозволяє перевірити, чи визначеним ім'я макросу за допомогою оператора #elif.

# 10.

1. Як задати змінний розмір масиву? Навести приклади.

Реализация массивов переменной длины мало отличается от реализации обычных статических массивов. Массив элементов типа T переменной длины обычно хранится в непрерывном блоке оперативной памяти размером sizeof(T)\*N, где N — число элементов, указываемое при описании (создании) массива. То есть описание массива переменной длины, фактически, просто маскирует динамическое выделение памяти. Разница может состоять в том, что (например, как в Си стандарта C99 и позже) массиву переменной длины выделяется память на стеке, как другим автоматическим переменным, в то время как типовой способ динамического выделения памяти (в Си — функция malloc) выделяет память в куче. Кроме того, для массива переменной длины компилятор, как правило, автоматически генерирует код освобождения памяти при выходе объявленного массива из области видимости.

В самом языке Си нет динамических массивов, но функции стандартной библиотеки malloc, free и realloc позволяют реализовать массив переменного размера:

int \*mas = (int\*)malloc(**sizeof**(int) \* n); // Создание массива из n элементов типа int  
 ...  
 mas = (int\*)realloc(mas, **sizeof**(int) \* m); // Изменение размера массива с n на m с сохранением содержимого  
 ...  
 free(mas); // Освобождение памяти после использования массива

Неудобство данного подхода состоит в необходимости вычислять размеры выделяемой памяти, применять явное преобразование типа и тщательно отслеживать время жизни массива (как и всегда при работе с динамически выделенной памятью в Си).

2. Розробити функцію, яка сортує тільки додатні елементи масиву на своїх місцях.

#include <iostream>

using namespace std;

void randMas(int\* mas, int n)

{

for (int i = 0; i < n; ++i)

mas[i] = rand() % 21 - 10;

}

void outputMas(int\* mas, int n)

{

for (int i = 0; i < n; ++i)

cout << mas[i] << " ";

cout << endl;

}

void sortMas(int\* mas, int n)

{

int\* masPol = new int[n];

int\* masInd = new int[n];

int kol = 0;

for (int i = 0; i < n; i++)

{

if (mas[i] > 0)

{

masPol[kol] = mas[i];

masInd[kol] = i;

kol++;

}

}

for (int i = 0; i < kol; i++)

{

for (int j = 0; j < kol; j++)

{

if (masPol[i] < masPol[j])

{

int c = masPol[i];

masPol[i] = masPol[j];

masPol[j] = c;

}

}

}

for (int i = 0; i < kol; i++)

{

mas[masInd[i]] = masPol[i];

}

}

int main()

{

const int n = 10;

int mas[n];

randMas(mas, n);

cout << "Original Massive:" << endl;

outputMas(mas, n);

cout << endl;

sortMas(mas, n);

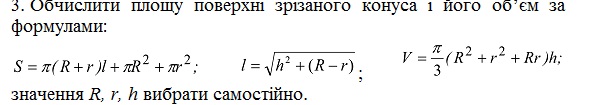
cout << "New Massive:" << endl;

outputMas(mas, n);

system("pause");

return 0;

}

3.

#include <iostream>

#include <math.h>

using namespace std;

int main()

{

double R, r, h;

double S, l, V;

const double PI = 3.14159265;

cin>>R>>r>>h;

S = PI \* (R+r)\*l+PI\*pow(R, 2)+PI\*pow(r, 2);

l = sqrt(pow(h,2)+(R-r));

V = PI/3 \*(pow(R, 2)+ pow(r, 2)+R\*r);

cout<<"S="<<S<<endl<<"l="<<l<<endl<<"V="<<V<<endl;

system("pause");

return 0;

}

4. Якою директивою підключається файли в модуль? Навести приклади.

В заголовному файлі модуля оголошується кожна функція, об'єкт і тип даних, які є частиною інтерфейсу виклику модуля - наприклад, в цьому випадку заголовки може містити тільки оголошення функції add. Кожен вихідний файл, який посилається на функцію add, повинен використовувати директиву #include для підключення заголовного файлу:

*/\* File triple.c \*/*  
 #include *"add.h"*  
   
 int triple(int x)  
 {  
 **return** add(x, add(x, x));  
 }

# 11.

1. Що таке рядок символів у мові С++? Навести приклади.

Найчастіше одномірні масиви використовуються для створення символьних рядків. У C ++

рядок визначається як символьний масив, який завершується нульовим символом

( '\ 0'). При визначенні довжини символьного масиву необхідно враховувати ознака її

завершення і задавати його довжину на одиницю більше довжини найбільшої рядки з тих,

які передбачається зберігати в цьому масиві.

Рядок - це символьний масив, який завершується нульовим символом.

Наприклад, оголошуючи масив str, призначений для зберігання 10-символьного рядка,

слід використовувати наступну інструкцію.

char str [11];

Заданий тут розмір (11) дозволяє зарезервувати місце для нульового символу в

наприкінці рядка.

У мові **С++** є кілька можливостей роботи з символьними даними. Класична робота зводиться до використання масиву символів. Для того, щоб скористуватися стандартними функціями **С++** для роботи з рядками, треба їх файли підключити в директиві **#include <string.h>**. Основні функції з цього пакету наведено нижче.

— **char\*strcat (char\*\_dest, const char\*\_src); —** функція реалізує з’єднання рядка dest з рядком src. Функція вертає покажчик на початок отриманого рядка (dest). Проміжний символ ‘\0’ рядка dest видаляється.

— **char\*strncat (char\*\_dest, const char\*\_src, size\_t\_maxlen); —** функція приєднує maxlen символів з рядка, на який вказує src, до рядка, на який вказує dest. Рядок dest повинен містити не менше maxlen вільних байтів. Якщо maxlen більше рядка src, виконується проста конкатенація.

— **char\*strchr (const char\*\_, int\_c);** **—** функція вертає покажчик на позицію першого входження символу “c” в рядок, на який вказує s. В рядок s включається і символ ‘\0’.

— **int strcmp (const char\*\_s1, const char\*\_s2); —** функція виконує порівняння двох рядків, на початок яких вказують s1 і s2. Функція вертає значення: менш нуля, якщо s1<s2; нуль, якщо s1==s2; більше нуля, якщо s1>s2.

— **int strnсmp (const char\*\_s1, const char\*\_s2, size\_t\_maxlen); —** функція, що аналогічна функції strcmp () і відрізняється тим, що виконується порівняння перших maxlen байтів.

— **int stricmp (const char \*\_s1, const char \*\_s2); —** функція виконує порівняння двох рядків, на що вказують s1 і s2. Перед порівнянням символи перетворюються на малі. Функція вертає значення: більше нуля, якщо s1>s2; рівно нулю, якщо s1==s2; менш нуля, якщо s1<s2.

— **int strlen (const char\*\_s); —** функція вертає довжину рядка в байтах, на який вказує s. Нуль-термінатор не враховується.

— **char\*stpcpy (char \*\_dest, const char \*\_src);** **—** функція копіює рядок, на який вказує src, в інше місце в пам'яті, на що вказує est. Функція вертає покажчик на кінець рядка, що скопіювався в dest.

— **char\*strncpy (char\*\_dest, const char\*\_src, size\_t\_maxlen); —** функція копіює maxlen байт з рядка, на який вказує src, в інше місце в пам'яті, на яке вказує dest. Нуль-термінатор теж копіюється. Якщо maxlen менше довжини рядка src, до рядка src не приєднується символ “\0”. І якщо більше, то рядок src переноситься повністю, а символи, що залишалися, заповнюються символом “\0”. Функція вертає покажчик dest.

— **char \*strlwr (char \*\_s);** **—**функція перетворює всі символи рядка, на початок якого вказує s, в малі літери. Функція повертає покажчик на початок цього рядка.

— **char \*strups (char \*\_s);** **—**функція перетворює всі символи рядка, на початок якої вказує s, в великі літери. Функція вертає покажчик на отриманий рядок.

— **char \*strset (char \*\_s, int\_ch); —** функція заповнює рядок, на початок якого вказує s, символом ch. Функція вертає покажчик на отриманий рядок.

— **char \*strtok (char \*\_s1, char \*\_s2); —** функція вертає наступну лексему із s1, яка відділена будь-яким символом з набора s2.

2. Розробити функцію, що знаходить максимальний і мінімальний елементи, їх номери у одновимірному масиві та міняє ці елементи місцями.

#include <iostream>

#include <cstdlib>

using namespace std;

void randMas(int mas[], int n)

{

for (int i = 0; i < n; ++i)

mas[i] = rand() % 100 - 21;

}

void outputMas(int mas[], int n)

{

for (int i = 0; i < n; ++i)

cout << mas[i] << " ";

cout << endl;

}

int maximum(int\* mas, int n)

{

int g = mas[0];

for (int i = 0; i < n; ++i)

{

if (mas[i] > g) g = mas[i];

}

return g;

}

int minimum(int\* mas, int n)

{

int m = mas[0];

for (int i = 0; i < n; ++i)

{

if (m > mas[i]) m = mas[i];

}

return m;

}

int IndexMin(int\* mas, int n)

{

int imin = 0;

for (int i = 0; i < n; ++i)

{

if (mas[i] < mas[imin]) imin = i;

}

return imin;

}

int IndexMax(int\* mas, int n)

{

int imax = 0;

for (int i = 0; i < n; ++i)

{

if (mas[i] > mas[imax]) imax = i;

}

return imax;

}

void Swap(int\* mas, int n) {

int min = 0, max = 0;

for (int i = 0; i < n; i++)

{

if (mas[i] < mas[min])

min = i;

if (mas[i] > mas[max])

max = i;

}

int tmp = mas[max];

mas[max] = mas[min];

mas[min] = tmp;

for (int i = 0; i < n; i++)

{

cout << mas[i] << " ";

}

}

int main()

{

const int n = 15;

int massive[n];

randMas(massive, n);

outputMas(massive, n);

int b = minimum(massive, n);

cout << "min = " << b << endl;

cout << "max = " << maximum(massive, n) << endl;

cout << "imin = " << IndexMin(massive, n) << endl;

cout << "imax = " << IndexMax(massive, n) << endl;

Swap(massive, n);

system("pause");

return 0;

}

3. Обчислити ;  де *a*=16,5; *b*=3,4; *x*=0,61.

#include<iostream>

#include<cmath>

using namespace std;

int main()

{

double a = 16.5;

double b = 3.4;

double x = 0.61;

double c = (b\*pow(x, 2) - a) / (exp(a\*x) - 1);

double S = ((pow(x, 3) \* pow(tan(pow((x + b), 2)), 2) + a) / (sqrt(x + b))) \* c;

cout << "S = " << S << endl;;

cout << "c = " << c << endl;;

system("pause");

return 0;

}

4)Як за допомогою умовної компіляції виключити повторне підключення файлу? Навести приклад.

#ifndef \_MACROS\_

#define \_MACROS\_

//Код

#endif

Данная функция позволяет избежать конфликт имен функции, то есть исключается повторное подключение файла.

Приклад:

#ifndef \_MY\_HEADER\_

#define \_MY\_HEADER\_

#define OUTPUT(n, mas,i) cout << "Input size:\n> ";\

cin >> n;\

mas = new int[n];\

for (i = 0; i < n; ++i)\

cin >> mas[i];

#define MINN(mas,n,imin) {for (i = 1; i < n; ++i) \

if (abs(mas[i]) < abs(mas[imin])) \

imin = i; \

cout << "Absolute minimum: \n< mas[" << imin << "] = " << mas[imin] << endl;

#define SABS(mas, n, i, abssum) i = 0; \

while (mas[i] && i < n) \

++i;\

while (i < n)\

abssum += mas[i], ++i;\

cout << "Absolute sum after first zero:\n< " << abssum <<endl;}

#define CHAN(t, mas, i, n) for(int i = 0; i < n/2; i++) \

{\

int t = mas[n / 2 + i];\

mas[n / 2 + i] = mas[i];\

mas[i] = t;\

}\

for(int i = 0; i < n; i++)\

cout << " "<< mas[i] << " " ; \

#endif

# 12.

1)Оператори continue, break, goto. Навести приклади.

BREAK

Оператор **break** имеет два назначения. Первое - это окончание работы оператора switch. Второе - это принудительное окончание цикла, минуя стандартную проверку условия. Данное назначение здесь и рассматривается. Когда оператор break встречается в теле цикла, цикл немедленно заканчивается и выполнение программы переходит на строку, следующую за циклом. Например:

#include<iostream>

#include<cmath>

using namespace std;

int main()

{

for (int i = 0; i < 1000000; ++i)

{

cout<< "i = " << i << endl;

if (i == 1895)

{

break;

}

}

system("pause");

return 0;

}

В данном случае, цикл прекратит работу, когда значение "i" будет равняться 1895.

CONTINUE

* Оператор **continue**используется только в циклах. В операторах for, while, do while**,**оператор continueвыполняет пропуск оставшейся части кода тела цикла и переходит к следующей итерации цикла.
* Разработаем программу с оператором continue. Программа должна работать циклически. Внутри цикла необходимо организовать ввод чисел. Если введено число от 0 до 10 включительно, то необходимо напечатать квадрат этого числа, иначе используя оператор continue пропустить оператор возведения в квадрат введенного числа. При введении отрицательного числа осуществить выход из цикла.

#include <iostream>

using namespace std;

int main()

{

    int in\_number; // переменная для хранения введённого числа

    do

    {

        cout << "Enter number: ";

        cin >> in\_number; // считываем введённое число в переменную

in\_number

        if ( in\_number > 10 || in\_number < 0 ) // если введённое число невходит в заданный интервал

            continue; // переход на следующую итерацию цикла do while

        cout << "square = " << in\_number \* in\_number << endl; // возводим в квадрат введённое число

    }

while ( in\_number >= 0 ); // пока введённое число больше либо равно нулю цикл будет работать

    system("pause");

    return 0;

}

**goto**

**Оператор goto** — это [**оператор управления порядком выполнения кода**](https://ravesli.com/urok-63-poryadok-vypolneniya-programm-operatory-upravleniya/), который заставляет ЦП сделать переход из одного участка кода на другой, осуществить так называемый прыжок. Другой участок кода идентифицируется с помощью **statement label**. Ниже приведен пример с использованием goto и statement label:

int main()

{

    double z;

tryAgain: // это statement label

    std::cout << "Enter a non-negative number";

    std::cin >> z;

    if (z < 0.0)

        goto tryAgain; // а это оператор goto

    std::cout << "The sqrt of " << z << " is " << sqrt(z) << std::endl;

    return 0;

}

В этой программе пользователю предлагается ввести неотрицательное число. Однако, если пользователь введет отрицательное число, то программа, используя оператор goto, сделает переход назад к строчке tryAgain. Затем пользователю снова нужно будет ввести число. Таким образом, мы можем постоянно запрашивать у пользователя ввод числа, пока он или она не введут корректные числа.

2. Створити функцію сортування другого рядка матриці і знаходження її максимального елемента.

#include<iostream>

#include <cstdlib>

#include<iomanip>

using namespace std;

void insertionSort(int arr[], int n)

{

int i, key, j;

for (i = 1; i < n; i++)

{

key = arr[i];

j = i - 1;

while (j >= 0 && arr[j] > key)

{

arr[j + 1] = arr[j];

j = j - 1;

}

arr[j + 1] = key;

}

}

int maxElem(int arr[], int n)

{

int max = arr[0];

for (int i = 0; i < n; ++i)

{

if (arr[i] > max) max = arr[i];

}

return max;

}

int main()

{

int n;

cout << "Vvedite razmer" << endl;

cin >> n;

int \*\*mas = new int\*[n];

for (int i = 0; i < n; i++)

mas[i] = new int[n];

for (int i = 0; i < n; i++)

{

for (int j = 0; j < n; j++)

{

mas[i][j] = rand() % 30-5;

}

}

for (int i = 0; i < n; i++)

{

for (int j = 0; j < n; j++)

{

cout << setw(4) << mas[i][j] << " ";

}

cout << endl << endl;

}

cout << endl;

insertionSort(mas[1], n);

for (int i = 0; i < n; i++)

{

for (int j = 0; j < n; j++)

{

cout << setw(4) << mas[i][j] << " ";

}

cout << endl << endl;

}

int s = maxElem(mas[1], n);

cout << "Maximum = " << s << endl;;

delete[]mas;

system("pause");

return 0;

}

3. Обчислити *y=sin3(x2+a)2* -  *z=x2/a+cos2(x+b)3*; де *a*=1,1; *b*=0,004; *x*=0,2.

#include<iostream>

#include<cmath>

using namespace std;

int main()

{

double a = 1.1;

double b = 0.004;

double x = 0.2;

double z = (pow(x, 2) / a) + pow(cos(pow((x + b), 3)), 2);

double y = pow(sin(pow((pow(x, 2) + a), 2)), 3) - pow((x / b), 1.0 / 3.0);

cout << "Z = " << z << endl;

cout << "Y = " << y << endl;

system("pause");

return 0;

}

4. Створити макропідстановку (макрос) для знаходження суми одновимірного масиву. Навести приклад застосування.

#include<iostream>

#include <stdio.h>

#include <stdlib.h>

#define SUMMARY(s,a,n) int s=0; \

for(int i=0; i<n; i++) \

s += a[i]; \

using namespace std;

int main()

{

const int n = 10;

int a[n];

for (int i = 0; i < n; ++i)

{

a[i] = rand() % 10;

cout << " " << a[i];

}

cout << endl;

SUMMARY(s, a, n)

cout << "Summa = "<< s << endl;

system("pause");

return 0;

}

# 13.

1. З тексту надрукувати всі слова, що починаються літерою "к".

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <string.h>

#include <iostream>

using namespace std;

int main() {

int count = 0;

char str[] = "kazda kobra, stone; drama...Kpril, joke... bambona!snow, . ?formula";

cout << str << endl;

char specs[] = " ,.;?!\n\t";

char temp[50] = "";

char \*token = strtok(str, specs);

int space = 0;

while (token != NULL) {

if (token[0] == 'k' || token[0] == 'K') {

space += strlen(token);

strcat(temp, token);

temp[space] = ' ';

space++;

}

token = strtok(NULL, specs);

}

cout << temp << endl;

system("pause");

return 0;

}

2. Для чого потрібен прототип функцій? Навести приклади.

Прототипом функции : это объявление функции, не содержащее тела функции, но указывающее имя функции, арность, типы аргументов и возвращаемый тип данных. В то время как определение функции описывает, что именно делает функция, прототип функции может восприниматься как описание её интерфейса.

Прототип и описание функции используются компилятором для того, чтобы вызов функции происходил правильным образом. Для этого компилятор сначала смотрит имя вызываемой функции и ищет в файле прототип или описание этой функции. Если найден прототип или описание, то проверяются аргументы, передаваемые функции в вызове, и использование возвращаемого значения.

 Прототип имеет следующий вид.

// прототип функции

void function(int arg1, double arg2);

Как видно в прототипе указываются по порядку тип возвращаемого значения (в данном примере void), название функции (в данном случае function) и список параметров в скобках. Объявление прототипа должно заканчиваться точкой с запятой.

3. Провести злиття відсортованих по зростанню масивів, розробивши відповідну функцію.

#include <iostream>

using namespace std;

void merge(int\* a, int\* b, int\* result, int n, int m)

{

int i = 0, j = 0, index = 0; //i, j - первый необработанный элемент a[], b[] соотв.

while (i < n && j < m)

{

if (a[i] < b[j])

{

result[index] = a[i];

i++;

index++;

}

else if (a[i] > b[j])

{

result[index] = b[j];

j++;

index++;

}

else

{

result[index] = a[i];

index++;

result[index] = a[i];

index++;

i++;

j++;

}

}

while (i < n)

{

result[index] = a[i];

index++;

i++;

}

while (j < m)

{

result[index] = b[j];

index++;

j++;

}

for (int i = 0; i < m + n; ++i)

cout << result[i] << " ";

cout << endl;

}

int main()

{

const int n = 5, m = 6;

int a[] = { 1,3,5,6,9 };

int b[] = { 2,4,6,6,11,15 };

int result[n + m];

merge(a, b, result, n, m);

system("pause");

return 0;

}

4. Створити макропідстановку (макрос) для знаходження мінімального елемента одновимірного масиву. Навести приклад застосування.

#include <iostream>

#include <time.h>

#define Min(mas,s) { min = mas[0]; for (int j = 1; j < s; j++)if(mas[j] < min) min = mas[j];}

using namespace std;

int main()

{

srand(time(NULL));

int size = 10;

int \*mas = new int[size];

for (int i = 0; i < size; i++) mas[i] = rand() % 10;

cout << "Nachalnije dannie\n";

for (int i = 0; i < size; i++) cout << "|" << mas[i];

cout << endl;

int min; Min(mas, size);

cout << "Min element = " << min << endl;

system("pause");

delete[]mas;

return 0;

}

# 14.

1. Скласти програму обчислення значення функцій:





#include <iostream>

#include <math.h>

using namespace std;

int main()

{

double x, y, hx = 0.1, a = 0.75, b = 1.19, c = -2.5;

for (x = 0; x <= 2; x += hx)

{

if (x < 0.5) y = (a\*x) + (b\*cos(x));

else

y = (b\*pow(x, 2)) + (c\*sin(2 \* x));

cout << "x=" << x << "\ty=" << y << endl;

}

cin.get();

return 0;

}

#include <iostream>

#include <math.h>

using namespace std;

int main()

{

float a, b, c, ha, hb, hc, p;

cin >> a >> b >> c;

p = (a + b + c) / 2;

ha = (2 / a)\*sqrt(p\*(p - a)\*(p - b)\*(p - c));

hb = (2 / b)\*sqrt(p\*(p - a)\*(p - b)\*(p - c));

hc = (2 / c)\*sqrt(p\*(p - a)\*(p - b)\*(p - c));

cout << "ha = " << ha << "\nhb = " << hb << "\nhc = " << hc << endl;

cin.get();

cin.get();

return 0;

}

2. Застосування функцій-покажчиків. Навести приклад.

Как и любой объект функция расположена в памяти и имеет адрес. Для хранения этого адреса необходимо применять указатель на фукнцию.

Синтаксис создания указателя на функцию.

тип\_результату (\* указатель) (список формальных аргументов);

При вызове функции через указатель допускается ее вызов как через разыменования так и без.

Тип (параметры и тип возвращаемого значения) указателя на функцию должен совпадать с типом функции. В отличие от фундаментальных типов данных, C++ неявно конвертирует функцию в указатель на функцию(поэтому не нужно использовать оператор адреса (&) для получения адреса функции).   
  
int My()

{

return 7;

}

int main()

{

int (\*fcnPtr)() = My;

return 0;

}

3. Знайти шість найменших елементів масиву, користуючись функцією сортування елементів.

#include <iostream>

using namespace std;

void randMas(int\* mas, int n)

{

for (int i = 0; i < n; ++i)

mas[i] = rand() % 21 - 10;

}

void outputMas(int\* mas, int n)

{

for (int i = 0; i < n; ++i)

cout << mas[i] << " ";

cout << endl;

}

void sortMas(int\* mas, int n)

{

for (int k = 0; k < n - 1; ++k)

for (int i = 0; i < n - 1 - k; ++i)

{

if (mas[i + 1] < mas[i])

{

int c = mas[i];

mas[i] = mas[i + 1];

mas[i + 1] = c;

}

}

}

int main()

{

const int size = 10;

int massive[size];

randMas(massive, size);

outputMas(massive, size);

sortMas(massive, size);

outputMas(massive, 6);

system("pause");

return 0;

}

4. Створити макропідстановку (макрос) для знаходження максимального елемента одновимірного масиву. Навести приклад застосування.

#define Max(mas,n) { max=mas[0];\

for (int j=1;j<n;j++)\

if(mas[j]>max)\

max=mas[j]; }

int main()

{

int \*mas = new int[N];

randmas(mas, N);

cout << "Nachalnije dannie\n";

outmas(mas, N);

cout << endl;

int max;

Max(mas, N);

cout << "Max element = " << max << endl;

delete[]mas;

system("pause");

return 0;

}

# 15.

1. Обчислити  де  a=1,5; b=15,5; x=-2,9.

#include <iostream>

using namespace std;

int main() {

double a = 1.5, b = 15.5, x = -2.9;

double y = pow(cos(pow((x), 3)), 2) - x / sqrt(pow(a, 2) + pow(b, 2));

double P = sqrt(pow(x, 2) + b) - pow(b, 2)\*pow(sin(x + a), 3) / x \* y;

cout << P;

system("pause");

}

2. Як правильно працювати з даними всередині функцій? Навести приклад.

В C++ існує 3 способи передачі параметрів у функцію:

1)Передача параметру за значенням (Call-By-Value). Це є проста передача копій змінних в функцію. У цьому випадку зміна значень параметрів в тілі функції не змінить значення, що передавались у функцію ззовні (при її виклику);

2)Передача параметру за адресою змінної. У цьому випадку функції в якості параметрів передаються не копії змінних, а копії адрес змінних, тобто покажчик на змінну. Використовуючи цей покажчик функція здійснює доступ до потрібних комірок пам’яті де розташована передана змінна і може змінювати її значення.

3)Передача параметру за посиланням (Call-By-Reference). Передається посилання (покажчик) на об’єкт (змінну), що дозволяє синтаксично використовувати це посилання як покажчик і як значення. Зміни, внесені в параметр, що переданий за посиланням, змінюють вихідну копію параметра викликаючої функції.

void MyFunction(int x, int\* y, int& c)

{

x = 8; // значення параметра змінюється тільки в межах тіла функції

\*y = 8; // значення параметра змінюється також за межами функції

c = 8; // значення параметра змінюється також за межами функції

return;

}

3. Створити функцію, яка перевіряє чи є задана матриця ортогональною, тобто скалярні добутки кожного рядка на інший дорівнюють 0.

#include <iostream>

#include <conio.h>

using namespace std;

#define M 3

#define N 3

int main()

{

setlocale(LC\_ALL, "RUS");

double s, matr[M][N];

int i1, i2, ok;

// ....Тут заполняем матрицу

ok = 1;

for (i1 = 0; i1 < M; i1++)

{

for (i2 = 0; i2 < M; i2++)

{

s = 0;

int i;

for (i = 0; i < N; i++)

s += matr[i1][i] \* matr[i2][i];

if (i1 != i2 && s != 0 || i1 == i2 && s != 1.0)

{

ok = 0;

break;

}

}

}

if (ok)

puts("Удовлетворяет");

else

puts("Не удовлетворяет");

system("pause");

}4. Навести приклад, як прочитати з файлу два дійсних числа.

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <cstdlib>

using namespace std;

int main() {

FILE \*f1 = fopen("Text.txt", "rt");

float a, b;

fscanf(f1, "%f%f", &a, &b);

fclose(f1);

cout << a << " - " << b;

system("pause");

}

# 16.

1. Основні відомі функції роботи з рядками у С++. Навести приклад.

— **char\*strcat (char\*\_dest, const char\*\_src); —** функція реалізує з’єднання рядка dest з рядком src. Функція вертає покажчик на початок отриманого рядка (dest). Проміжний символ ‘\0’ рядка dest видаляється.

— **char\*strncat (char\*\_dest, const char\*\_src, size\_t\_maxlen); —** функція приєднує maxlen символів з рядка, на який вказує src, до рядка, на який вказує dest. Рядок dest повинен містити не менше maxlen вільних байтів. Якщо maxlen більше рядка src, виконується проста конкатенація.

— **char\*strchr (const char\*\_, int\_c);** **—** функція вертає покажчик на позицію першого входження символу “c” в рядок, на який вказує s. В рядок s включається і символ ‘\0’.

— **int strcmp (const char\*\_s1, const char\*\_s2); —** функція виконує порівняння двох рядків, на початок яких вказують s1 і s2. Функція вертає значення: менш нуля, якщо s1<s2; нуль, якщо s1==s2; більше нуля, якщо s1>s2.

— **int strnсmp (const char\*\_s1, const char\*\_s2, size\_t\_maxlen); —** функція, що аналогічна функції strcmp () і відрізняється тим, що виконується порівняння перших maxlen байтів.

— **int stricmp (const char \*\_s1, const char \*\_s2); —** функція виконує порівняння двох рядків, на що вказують s1 і s2. Перед порівнянням символи перетворюються на малі. Функція вертає значення: більше нуля, якщо s1>s2; рівно нулю, якщо s1==s2; менш нуля, якщо s1<s2.

— **int strlen (const char\*\_s); —** функція вертає довжину рядка в байтах, на який вказує s. Нуль-термінатор не враховується.

— **char\*stpcpy (char \*\_dest, const char \*\_src);** **—** функція копіює рядок, на який вказує src, в інше місце в пам'яті, на що вказує est. Функція вертає покажчик на кінець рядка, що скопіювався в dest.

— **char\*strncpy (char\*\_dest, const char\*\_src, size\_t\_maxlen); —** функція копіює maxlen байт з рядка, на який вказує src, в інше місце в пам'яті, на яке вказує dest. Нуль-термінатор теж копіюється. Якщо maxlen менше довжини рядка src, до рядка src не приєднується символ “\0”. І якщо більше, то рядок src переноситься повністю, а символи, що залишалися, заповнюються символом “\0”. Функція вертає покажчик dest.

— **char \*strlwr (char \*\_s);** **—**функція перетворює всі символи рядка, на початок якого вказує s, в малі літери. Функція повертає покажчик на початок цього рядка.

— **char \*strups (char \*\_s);** **—**функція перетворює всі символи рядка, на початок якої вказує s, в великі літери. Функція вертає покажчик на отриманий рядок.

— **char \*strset (char \*\_s, int\_ch); —** функція заповнює рядок, на початок якого вказує s, символом ch. Функція вертає покажчик на отриманий рядок.

— **char \*strtok (char \*\_s1, char \*\_s2); —** функція вертає наступну лексему із s1, яка відділена будь-яким символом з набора s2.

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <cstring>

using namespace std;

int main()

{

int col = 0;

char str[] = "Repetitio est mater studiorum. Help.";

char seps[] = " ;,\t\n";

cout << str << endl;

char \*token = strtok(str, seps);

while (token != NULL)

{

col++;

token = strtok(NULL, seps);

}

cout << "\nKilkist sliv = " << col << endl;

system("pause");

return 0;

}

2. Створити дві функції:

- міняє місцями перший і останній рядки матриці;

- знаходить добуток елементів головної діагоналі квадратної матриці.

#include <iostream>

using namespace std;

int \*\*createMatr(int n, int m) {

int \*\*matr = new int\*[n];

for (int i = 0; i < n; i++)

matr[i] = new int[m];

return matr;

}

void generateMatr(int \*\*matr, int n, int m) {

for (int i = 0; i < n; i++)

for (int j = 0; j < m; j++)

matr[i][j] = rand() % 20 - 10;

}

void outputMatr(int \*\*matr, int n, int m) {

for (int i = 0; i < n; i++, cout << endl)

for (int j = 0; j < m; j++, cout << "\t")

cout << matr[i][j];

cout << endl << endl;

}

void changeRows(int \*\*matr, int n, int m) {

int \*first = new int[n];

first = matr[0];

matr[0] = matr[n - 1];

matr[n - 1] = first;

}

int mainDiag(int \*\*matr, int n, int m) {

if (n != m)

return 0;

int dobutok = 1;

for (int i = 0; i < n; i++)

dobutok \*= matr[i][i];

return dobutok;

}

int main() {

int n = 5, m = 5;

int \*\*matr = createMatr(n, m);

generateMatr(matr, n, m);

outputMatr(matr, n, m);

changeRows(matr, n, m);

outputMatr(matr, n, m);

int dobutok = mainDiag(matr, n, m);

cout << endl << endl << dobutok;

system("pause");

}

3. Обчислити K=ln(a+x3)+sin2(x/b); M=e-cx. де a=10,2; b=9,3; x=2,4;

#include <iostream>

using namespace std;

int main() {

double a = 10.0, b = 9.3, x = 2.4, c = 0.5;

double k = log(a + pow(x, 3)) + pow(sin(x / b),2);

double M = exp(-c \* x)\*(x+pow((x+a),1.0/3))/ (x - sqrt(abs(x + a)));

cout << k << endl;

system("pause");

}

\*4. Створити програму, яка підраховує кількість символів ‘,’ у файлі.

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <cstdio>

#include <iostream>

using namespace std;

int main() {

setlocale(LC\_ALL, "ru");

char symbol = ',';

FILE\* f = fopen("Text.txt", "r");

int count;

char ch;

count = 0;

while ((ch = fgetc(f)) != EOF)

if (ch == symbol)

count++;

printf("Нашлось %d символов '%c`\n", count, symbol);

system("pause");

return 0;

}

# 17.

1. Які існують оператори циклу в мові С++? Навести приклади.

В мові **С++** існують три оператори циклу: **while, do, for.**  
Оператор циклу із передумовою:  
**while (А) оператор;**  
⎯будь-який простий, складений чи порожній оператор, тут **А** — будьякий припустимий вираз. Виконується цей оператор таким чином: якщо  
результат виразу А не дорівнює нулю (“істина”), то виконується цикл, а якщо  
дорівнює нулю (хибний), то цикл не виконується і керування передається  
наступному за **while** оператору.  
Оператор циклу **for** має наступну форму запису:  
**for ( [вираз1;] [вираз2;] [вираз3] ) оператор;**  
де **вираз1** — вираз ініціалізації, що звичайно використовується для  
встановлення початкового значення; це вираз присвоювання (необов'язковий  
параметр); **вираз2** — вираз умови, що визначає, за якої умови цикл буде  
повторюватися (необов'язковий параметр); **вираз3** — вираз ітерації, що  
визначає крок зміни змінних, які керують циклом після кожного виконання  
(необов'язковий параметр).  
Цей оператор реалізується таким чином:  
— спочатку виконується вираз ініціалізації (ця нотація може бути  
виконана до оператора **for**);  
— обчислюється умовний вираз;  
— якщо результат умовного виразу “істина” (не дорівнює нулю), то  
виконується оператор циклу;  
— обчислюється вираз ітерації;  
— знову перевіряється умова;  
— як тільки умовний вираз приймає значення нуль (хибне), керування  
передається оператору, що розташовується за оператором циклу for. 33  
Оскільки перевірка умови виконується перед циклом, то цикл може не  
виконатися жодного разу, якщо умова відразу буде помилковою.  
Оператор циклу **do** звичайно використовується в тих випадках, коли тіло  
циклу повинне виконуватися хоча б один раз, і має наступну структуру  
запису:  
**do оператор**

**while (А);** де **А** ⎯умовний вираз.  
Виконується оператор **do** таким чином: спочатку здійснюється вхід у  
тіло циклу і виконується оператор (він може бути простий чи складний),  
після цього перевіряється умова і, якщо вона виконується, тобто “істина” (не  
дорівнює нулю), то цикл повторюється, а якщо “неправда” ⎯здійснюється  
вихід з циклу.  
**Приклад**   
Підсумувати всі парні числа від парного числа **a** до парного числа **b**  
(**a**<**b**).

#include <iostream>  
int main()  
{  
int sum=0, a, b, i;  
std::cout<<"Vvedi a, a=";  
std::cin>>a;  
std::cout<<"\nVvedi b, b=";  
std::cin>>b;  
for (i=a; i<b; i+=2)  
sum +=i;  
std::cout<<"Summa dorivnue Sum= "<<sum;  
std::cout<<"\n";  
std::cin.get();  
std::cin.get();  
return 0;  
}

2. Написати функцію, яка в залежності від заданої ознаки проводить сортування масиву по зростанню або по зменшенню.

#include <iostream>

#define ZNAK >

using namespace std;

void randMas(int\*mas, int n) {

for (int i = 0; i < n; ++i) {

mas[i] = rand() % 50 - 3;

}

}

void outMas(int\*mas, int n) {

for (int i = 0; i < n; ++i)

cout << mas[i] << "\t";

}

void sortMas(int\*mas, int n) {

for (int k = 0; k < n - 1; ++k) {

for (int i = 0; i < n - 1 - k; ++i) {

if (mas[i] ZNAK mas[i + 1])

{

int a = mas[i];

mas[i] = mas[i + 1];

mas[i + 1] = a;

}

}

}

}

int main()

{

int n;

cin >> n;

int\*mas = new int[n];

randMas(mas, n);

cout << "Origin mas:" << endl;

outMas(mas, n);

sortMas(mas, n);

cout <<"\nSort mas:"<< endl;

outMas(mas, n);

cout << endl;

delete[]mas;

system("pause");

return 0;

}

3. Обчислити   
де a=0,3; b=0,9; x=0,53.

#include<iostream>

#include<math.h>

using namespace std;

int main()

{

double a = 0.3, b = 0.9, x = 0.53;

double y = (pow(a, 2 \* x) + pow(b, -x)\*cos(a + b)\*x) / fabs(x - 1);

double r = sqrt(pow(x, 2) + b) - (pow(b, 2)\*pow(sin(x + a), 3)) / x;

cout << "y = " << y << "\nr = " << r << endl;

system("pause");

return 0;

}

4. Створити програму, яка підраховує загальну кількість символів у файлі.

#define \_CRT\_SECURE\_NO\_WARNINGS

#include<iostream>

using namespace std;

int main()

{

FILE \*fin(NULL);

char namef[40];

puts("Input file name\n");

fgets(namef, 99, stdin);

char \*p = strchr(namef, '\n');

if (p) \*p = '\0';

fin = fopen(namef, "r");

if (!fin)

{

printf("Can not open file\n");

return 1;

}

int num = 0;

while (!feof(fin))

{

char ch;

fscanf(fin,"%c", &ch);

num++;

}

fclose(fin);

printf("kolichestvo simvolov = %i ", (num-1));

system("pause");

return 0;

}

# 18.

1. Які типи даних вам відомі? Навести приклад.

Основні типи даних(*див. табл*) часто називають арифметичними, тому  
що їх можна використовувати в арифметичних операціях. Для опису основних  
типів мови C++ використовують такі службові слова:

* **int** (цілий);
* **char** (символьний);
* **bool** (логічний);
* **float** (дійсний);
* **double** (дійсний з подвійною точністю);
* **void** (порожній, не має значення).

Типи **int, char, bool** називають ***цілими***, а типи **float** та **double** — ***дійсними з***  
***плаваючою крапкою***. Код, що формує компілятор для обробки цілих величин, відрізняється від коду для величин з плаваючою крапкою.  
Для уточнення внутрішнього подання та діапазону значень стандартних  
типів **мова C++ використовує чотири специфікатори типу**:

* **short** (короткий);
* **long** (довгий);
* **signed** (знаковий);
* **unsigned** (беззнаковий).

Базові типи даних для ПК (платформа Intel)

|  |  |  |
| --- | --- | --- |
| Тип | Розмір, байт | Значення |
| **bool** | 1 | true або false |
| **unsigned short int** | 2 | від 0 до 65 535 |
| **short int** | 2 | від -32 768 до 32 767 |
| **unsigned long int** | 4 | від 0 до 4 294 967 295 |
| **long int** | 4 | від -2 147 483 648 до 2 147 483 647 |
| **int** (16 розрядів) | 2 | від -32 768 до 32 767 |
| **int** (32 розряди) | 4 | від -2 147 483 648 до 2 147 483 647 |
| **unsigned int** (16 розрядів) | 2 | від 0 до 65 535 |
| **unsigned int** (32 розряди) | 4 | від 0 до 4 294 967 295 |
| **char** | 1 | від 0 до 256 |
| **float** | 4 | від 1.2е-38 до 3.4е38 |
| **double** | 8 | від 2.2е-308 до 1.8е308 |
| **long double** | 10 | від 3.4е-4932 до 3.4е+4932 |

У *табл.* наведено діапазони значень та розміри основних типів даних  
(для 16-розрядного і 32-розрядного процесорів). Розмір однакового типу даних може відрізнятися на комп’ютерах різних платформ, а також може залежати від застосованої операційної системи. Тому при оголошенні тієї чи іншої змінної потрібно чітко уявляти, скільки байт вона буде займати в пам’яті ЕОМ, щоб запобігти проблемам, пов’язаним з переповненням і неправильною.

2. Розробити функцію, яка знаходить одинакові елементи у двох масивах, та використати її для обробки рядків матриці.

#include <iostream>

using namespace std;

template <typename myT>

void massGen(myT\* mass, int n)

{

for (int i = 0; i < n; ++i) {

mass[i] = rand() % 10;

}

}

template <typename myT>

void outputMas(myT\* mas, int n)

{

for (int i = 0; i < n; ++i) {

cout << mas[i] << " ";

}

cout << endl << endl;;

}

template <typename myT>

myT findLen(myT \*mas1, myT \*mas2, int n1, int n2) {

int length = 0;

for (int i = 0; i < n1; i++) {

for (int j = 0; j < n2; j++) {

if (mas1[i] == mas2[j]) {

length++;

}

}

}

return length;

}

template <typename myT>

void findNums(myT \*mas1, myT \*mas2, int n1, int n2, myT \*&newMas, myT& len) {

len = findLen(mas1, mas2, n1, n2);

newMas = new int[len];

int length = 0;

for (int i = 0; i < n1; i++) {

for (int j = 0; j < n2; j++) {

if (mas1[i] == mas2[j]) {

newMas[length++] = mas1[i];

}

}

}

}

template <typename myT>

myT createMatr(int n, int m) {

if (n < 1 || m < 1) {

return NULL;

}

int matr = new int \*[n];

for (int i = 0; i < n; i++) {

matr[i] = new int[m];

}

return matr;

}

template <typename myT>

void matrGen(myT &matr, int n, int m, myT \*mass) {

for (int i = 0; i < n; i++) {

for (int j = 0; j < m; j++) {

matr[i][j] = mass[j];

}

}

}

template <typename myT>

void outputMatr(myT matr, int n, int m)

{

for (int i = 0; i < n; i++, cout << endl) {

for (int j = 0; j < m; j++) {

cout << matr[i][j] << " ";

}

}

}

void deleteMatr(int &matr, int n) {

for (int i = 0; i < n; i++) {

delete matr[i];

}

delete matr;

}

int main()

{

int lenMass1, lenMass2, lenMass3 = 0;

cout << "Enter size of first Massive - ";

cin >> lenMass1;

int \*mas1 = new int[lenMass1];

massGen(mas1, lenMass1);

cout << "Output mass 1: \n";

outputMas(mas1, lenMass1);

cout << "Enter size of second Massive - ";

cin >> lenMass2;

int \*mas2 = new int[lenMass2];

massGen(mas2, lenMass2);

cout << "Output mass 2: \n";

outputMas(mas2, lenMass2);

int \*mas3;

findNums(mas1, mas2, lenMass1, lenMass2, mas3, lenMass3);

cout << "Output (mass 1 & mass 2): \n";

outputMas(mas3, lenMass3);

int matrCount = 0;

cout << "Enter the numver of rows - ";

cin >> matrCount;

int matr = createMatr<int>(matrCount, lenMass3);

matrGen(matr, matrCount, lenMass3, mas3);

cout << "\nOutput the matr:\n";

outputMatr(matr, matrCount, lenMass3);

deleteMatr(matr, matrCount);

delete mas1;

delete mas2;

delete mas3;

system("pause");

}

*3. Обчислити*  *де a=0,5; b=3,1; x=1,4.*

#include <iostream>

#include <math.h>

using namespace std;

int main()

{

double a = 0.5, b = 3.1, x = 1.4;

double l = sqrt(fabs((a\*pow(x, 2)\*sin(2 \* x)) + (exp(-2 \* x)\*(x + b))));

double w = (1.0 / pow(cos(pow(x, 3)), 2)) - (x / pow((pow(a, 2) + pow(b, 2)), 1.0 / 3));

cout << "l = " << l << "\nw = " << w << endl;

system("pause");

return 0;

}

4. Створити програму, яка підраховує загальну кількість слів у файлі. Слова у файлі розділені пробілами.

#define \_CRT\_SECURE\_NO\_WARNINGS

#include<iostream>

#include<stdio.h>

#include<conio.h>

using namespace std;

int main()

{

FILE \*fin(NULL);

char namef[40];

puts("Input file name\n");

fgets(namef, 99, stdin);

char \*p = strchr(namef, '\n');

if (p) \*p = '\0';

fin = fopen(namef, "r");

if (!fin)

{

printf("Can not open file\n");

return 1;

}

char symbol;

int ColWord = 1;

while ((symbol = fgetc(fin)) != EOF)

{

if ((symbol == '\n') || (symbol == ' ') || (symbol == '\0'))

ColWord++;

}

fclose(fin);

printf("ColWord = %i", ColWord);

system("pause");

return 0;

}

# 19.

1. Які існують оператори циклу в мові С++? Навести приклади.

В мові **С++** існують три оператори циклу: **while, do, for.**

Оператор циклу із передумовою:

**while (А) оператор;**

⎯ будь-який простий, складений чи порожній оператор, тут **А** —будь-який припустимий вираз. Виконується цей оператор таким чином: якщо результат виразу А не дорівнює нулю (“істина”), то виконується цикл, а якщо дорівнює нулю (хибний), то цикл не виконується і керування передається наступному за **while** оператору.

Оператор циклу **for**  має наступну форму запису:

**for ( [вираз1;] [вираз2;] [вираз3] ) оператор;**

де  **вираз1** — вираз ініціалізації, що звичайно використовується для встановлення початкового значення; це вираз присвоювання (необов'язковий параметр); **вираз2** — вираз умови, що визначає, за якої умови цикл буде повторюватися (необов'язковий параметр); **вираз3** — вираз ітерації, що визначає крок зміни змінних, які керують циклом після кожного виконання (необов'язковий параметр).

Цей оператор реалізується таким чином:

— спочатку виконується вираз ініціалізації (ця нотація може бути виконана до оператора **for**);

— обчислюється умовний вираз;

— якщо результат умовного виразу “істина” (не дорівнює нулю), то виконується оператор циклу;

— обчислюється вираз ітерації;

— знову перевіряється умова;

* як тільки умовний вираз приймає значення нуль (хибне), керування передається оператору, що розташовується за оператором циклу for.

Оскільки перевірка умови виконується перед циклом, то цикл може не виконатися жодного разу, якщо умова відразу буде помилковою.

Оператор циклу **do** звичайно використовується в тих випадках, коли тіло циклу повинне виконуватися хоча б один раз, і має наступну структуру запису:

**do оператор**

**while (А);**

де **А** ⎯ умовний вираз.

Виконується оператор **do** таким чином: спочатку здійснюється вхід у тіло циклу і виконується оператор (він може бути простий чи складний), після цього перевіряється умова і, якщо вона виконується, тобто “істина” (не дорівнює нулю), то цикл повторюється, а якщо “неправда” ⎯ здійснюється вихід з циклу.

2. Створити функцію, яка міняє місцями перший і останній стовпці квадратної матриці і знаходить суму елементів побічної діагоналі.

#include <iostream>

using namespace std;

template <typename T>

T \*\* createMatr(int n)

{

if (n < 1) return NULL;

int \*\*matr = new int\*[n];

for (int i = 0; i < n; ++i)

matr[i] = new int[n];

return matr;

}

template <typename T>

void randMatr(T\*\* matr, int n)

{

for (int i = 0; i < n; ++i)

for (int j = 0; j < n; ++j)

matr[i][j] = rand() % 21 - 10;

}

template <typename T>

void outputMatr(T\*\* matr, int n)

{

for (int i = 0; i < n; ++i, cout << endl)

for (int j = 0; j < n; ++j)

cout << matr[i][j] << "\t";

cout << endl;

}

template <typename T>

void deleteMatr(T\*\*&matr, int n)

{

for (int i = 0; i < n; ++i)

delete[] matr[i];

delete[] matr;

matr = NULL;

}

template <typename T>

void pereStolb(T\*\* matr, int n)

{

for (int i = 0; i < n; ++i)

{

T c = matr[i][0];

matr[i][0] = matr[i][n - 1];

matr[i][n - 1] = c;

}

}

template <typename T>

int sumPobDiag(T\*\* matr, int n)

{

int res = 0;

for (int i = 0; i < n; ++i)

{

res += matr[i][n - 1 - i];

}

return res;

}

int main()

{

int n = 5;

int\*\* mas2d = createMatr<int>(n);

randMatr(mas2d, n);

cout << "Original matrix:" << endl;

outputMatr(mas2d, n);

pereStolb(mas2d, n);

cout << "Result matrix:" << endl;

outputMatr(mas2d, n);

int s = sumPobDiag(mas2d, n);

cout << "Summa = " << s << endl << endl;

deleteMatr(mas2d, n);

system("pause");

return 0;

}

3. Обчислити   де *a*=0,5; *b*=2,9; *x*=0,3.

#include <iostream>

#include <cmath>

using namespace std;

int main()

{

double a = 0.5, b = 2.9, x = 0.3, pi = 3.14;

double F =exp(2\*x)\*log(a+x)-(pow(b,3\*x)\*log(abs(x-b)));

double m = pow(a, 3) + exp(-x)\*(cos(b\*x));

double l = b\*x - exp(-x)\*sin(b\*x)+1;

double U = m / l;

cout << "F=" << F << endl;//-2.89771

cout << "U=" << U << endl;//0.462274

system("pause");

return 0;

}

4. Створити програму, яка записує у текстовий файл одновимірний масив.

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <cmath>

#include <cstdio>

#include <cstdlib>

using namespace std;

void randMass(int mass[], int size) {

for (int i = 0; i < size; i++)

mass[i] = rand() % 20;

}

void outputMass(int mass[], int size) {

for (int i = 0; i < size; i++, " ")

cout << mass[i];

cout << endl;

}

int main() {

const int size = 5;

int mass[size]{};

randMass(mass, size);

outputMass(mass, size);

FILE \*file1 = fopen("Text.txt", "w");

for (int i = 0; i < size; i++)

fprintf(file1, "%d ", mass[i]);

fclose(file1);

file1 = fopen("Text.txt", "rt");

int mass2[size]{};

for (int i = 0; i < size; i++) {

fscanf(file1, "%d", &mass2[i]);

}

outputMass(mass2, size);

system("pause");

return 0;

}

# 20.

1. Чим відрізняються змінні від констант? Навести приклади.

**Змінна** — це об’єкт програми, що займає в загальному випадку декілька комірок пам’яті, та призначений зберігати дані. При оголошенні змінної для неї резервується де-яка область пам’яті, розмір якої залежить від конкретного типу змінної. Будь-яка константа, змінна, значення чи функція в **С++** характеризується своїм типом. Тип цих об'єктів визначає безліч припустимих значень, форму внутрішнього уявлення, а так само безліч припустимих операцій. Тому всі дані, що використовуються в програмі, до їхньої появи обов'язково повинні бути оголошені (описані).

Наприклад,

**bool dd = true;** — логічна змінна dd=1;

**bool dd1 = 1;** — логічна змінна dd1=true;

**int a = 1, b = 0;** — цілі змінні a = 1, b = 0;

**сhar sim =’A’;** — символьна змінна sim =’A’;

**float Age = 18.5;** — десяткова змінна Age = 18.5 з крапкою, що плаває;

**void MyFunction();** — функція не вертає ніякого значення, тощо.

**Константи** відрізняються від змінних тим, що значення, що присвоєне константі первісно, не може бути змінено протягом виконання всієї програми. Оголошення констант виконується за наступною схемою:

сonst [тип] ідентифікатор = значення;

Наприклад:

**const double Pi = 3.1415;** // константа **Pi** принимает значение 3.1415;

**const a=1;** // по умолчанию тип **int**.

2. Створити функцію, яка вставляє задане число у відсортований масив будь-яку кількість разів.

#include <iostream>

#include<time.h>

using namespace std;

void outmas(int\*mas, int n)

{for (int i = 0; i < N - 1; i++)

cout << a[i] << " ";

}

void randmas(int\*mas, int n)

{ for (i = 0; i < N; i++)

{

a[i] = rand() % 20;

}

void sortmas (int\*mas, int n)

{for (int i = 0; i < N - 1; i++)

{

for (j = N - 1; j > i; j--)

if (a[j - 1] > a[j])

{

swap(a[j], a[j - 1]);

}

}

}

void abc (int\*mas, int n)

{

i = 0;

while ((a[i] < k) && (i < N))

i++;

for (j = N; j > i; j--) a[j] = a[j - 1];

a[i] = k;

}

int main()

{

srand(time(NULL));

int i, N, j, k;

cout << endl << "Dlina massiva - N: "; cin >> N;

cout << "\n";

int \*a = new int[N + 1]

randmas(a,N);

cout << "Original mas: " << endl;

outmas(a,N);

sortmas (a,N)

cout << "Massiv otsortirovannii po vozrastaniu: " << endl;

outmas(a,N);

cout << endl << "k: "; cin >> k;

abc(a,k,N);

cout << "Massiv: " << endl;

outmas(a,N);

delete[] a;

system("pause");

return 0;

}

3.Обчислити  де *x*=1,2; *y*=0,9; *z*=3,5.

#include <iostream>

#include <cmath>

using namespace std;

int main()

{

double x = 1.2, y = 0.9, z = 3.5, pi = 3.14;

double B = 1 + (pow(z,3)) / (3 + (pow(z,2) / 5));

double m = z\*(cos(x - pi / 6));

double l = ((1 / 2) + pow(sin(y), 3));

double A = m / l;

cout << "A=" << A << endl;// A=5.67736

cout << "B=" << B << endl;//B=8.86697

system("pause");

return 0;

}

4. Створити програму, яка записує у бінарний файл одновимірний масив.

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

int main()

{

int mas[4] = { 3, 5, 7, 45 };

FILE\*f;

f = fopen("testfile.txt", "wb");

fwrite(mas, 1, sizeof(int) \* 4, f);

fclose(f);

f = fopen("testfile.txt", "rb");

int mas2[4];

fread(mas2, 1, sizeof(int) \* 4, f);

fclose(f);

for (int i = 0; i < 4; ++i)printf("%d", mas2[i]);

system("pause");

return 0;

}

# 21.

1. Які існують види умовного оператора та чим вони відрізняються? Навести приклади.

Условный оператор if

Для организации вычислений в зависимости от какого-либо условия в C++ предусмотрен условный оператор if, который в общем виде записывается следующим образом:

**if (условие) оператор\_1; else оператор\_2;**

Здесь условие — это логическое выражение, переменная или константа.

Работает условный оператор следующем образом. Сначала вычисляется значения выражения, записанного в виде условия. Если оно имеет значение истина (true), выполняется оператор\_1. В противном случае (значение ложное (false) ) оператор\_2.

Если в задаче требуется, чтобы в зависимости от значения условия выполнялся не один оператор, а несколько, их необходимо заключить в фигурные скобки, как составной оператор.

**if (условие) { оператор\_1; оператор\_2; … }**

**else { оператор\_1; оператор\_2; … }**

Альтернативная ветвь **else** в условном операторе может отсутствовать, если в ней нет необходимости.

Пример:

Сравнить значения переменных a и b

int a, b;

cin>>a;

cin>>b;

if (a==b) cout<<"a equal b";

else cout<<"a not equal b";

**Оператор варианта switch**

Он необходим в тех случаях, когда в зависимости от значений переменной надо выполнить те или иные операторы:

**switch (выражение)**  
**{**  
**case значение\_1: операторы\_1; break;**  
**case значение\_2: операторы\_2; break;**  
**case значение\_3: операторы\_3; break;**  
**…**  
**case значение\_n: операторы\_n; break;**  
**default: операторы; break;**  
**}**

Оператор работает следующем образом. Вычисляется значение выражения. Затем выполняются операторы, помеченные значением, совпадающим со значением выражения. То есть если, выражение принимает значение\_1, то выполняются операторы\_1 и т.д.. Если выражение не принимает ни одного из значений, то выполняются операторы, расположенные после слова **default**.

Ветвь **default** может отсутствовать, тогда оператор имеет вид:

**switch (выражение)**  
**{**  
**case значение\_1: операторы\_1; break;**  
**case значение\_2: операторы\_2; break;**  
**case значение\_3: операторы\_3; break;**  
**…**  
**case значение\_n: операторы\_n; break;**  
**}**

Оператор **break** необходим для того, чтобы осуществить выход из операторы **switch**. Если он не указан, то будут выполняться следующие операторы из списка, несмотря на то, что значение, которым они помечены, не совпадает со значением выражения.

Пример:

Необходимо вывести на название дня недели, соответствующее заданному числу D, при условии, что в месяце 31 день и 1-е число — понедельник.

#include "stdafx.h"

#include <iostream>

using namespace std;

int main ()

{

unsigned int D, R; //описанны целые положительные числа

cout<<"D=" ;

cin>>D;

R=D%7;

switch (R)

{

case 1: cout<<"Monday \n"; break;

case 2: cout<<"Tuesday \n"; break;

case 3: cout<<"Wednesday \n"; break;

case 4: cout<<"Thursday \n"; break;

case 5: cout<<"Friday \n"; break;

case 6: cout<<"Saturday \n"; break;

case 0: cout<<"Sunday \n"; break;

}

system ("pause");

return 0;

}

2. Створити функцію, що визначає номери елементів, які зустрічаються у масиві тільки один раз.

#include <iostream>

#include <algorithm>

using namespace std;

void func(int \*mas,size\_t n)

{

for (size\_t i = 0; i < n; i++)

if (count (mas, mas + n, mas[i]) == 1)

cout << i + 1 << " ";

}

int main()

{

int mas[] = {1, 2, 3, 1, 3, 4, 5, 6, 4 };

func (mas, sizeof (mas) / sizeof (\*mas));

system("pause");

return 0;

}

3. Обчислити  де *x*=10,3;*y*=4,93; *z*=0,4.

#include <iostream>

#include <math.h>

#include <stdlib.h>

using namespace std;

int main()

{

double x = 10.3;

double y = 4.93;

double z = 0.4;

double P , S;

P = (3+exp(y-1))/(1+pow(x,2)\*fabs(y-tan(z)));

S = 1+y-(pow(y-x,2))/2+(fabs(y-x))/3;

cout <<"P = "<<P<<endl<<"S = "<<S<<endl;

system ("pause");

return 0;

}

4. Як дописати в існуючий текстовий файл інформацію. Навести приклад.

Запись информации в текстовый файл

Для того чтобы начать работать с текстовым файлом, необходимо описать переменную типа **ofstream**. Например, так:

**ofstream F;**

Будет создана переменная **F** для записи информации в файл. На следующим этапе файл необходимо открыть для записи. В общем случае оператор открытия потока будет иметь вид:

**F**.**open**(**«file»**, **mode**);

Здесь **F** — переменная, описанная как **ofstream**, **file** — полное имя файла на диске, **mode** — режим работы с открываемым файлом. Обратите внимание на то, что при указании полного имени файла нужно ставить двойной слеш. Для обращения, например к файлу **accounts.txt,** находящемуся в папке **sites**на диске **D**, в программе необходимо указать: **D:\\sites\\accounts.txt**.

Параметр mode может отсутствовать, в этом случае файл открывается в режиме по умолчанию для данного потока.

После удачного открытия файла (в любом режиме) в переменной **F** будет храниться **true**, в противном случае **false**. Это позволит проверить корректность операции открытия файла.

Если вы хотите открыть существующий файл в режиме дозаписи, то в качестве режима следует использовать значение **ios::app**.

После открытия файла в режиме записи, в него можно писать точно так же, как и на экран, только вместо стандартного устройства вывода **cout** необходимо указать имя открытого файла.

Например, для записи в поток **F**переменной **a**, оператор вывода будет иметь вид:

**F<<a;**

Для последовательного вывода в поток **G** переменных **b**, **c**, **d** оператор вывода станет таким:

**G<<b<<c<<d;**

Закрытие потока осуществляется с помощью оператора:

**F.close();**

В качестве примера рассмотрим следующую задачу.

...

fstream p(FILENAME, ios::in | ios::out | ios::ate);

if(!p)

{

     p.open(FILENAME, ios::app);

     p.close();

     p.open(FILENAME, ios::in | ios::out | ios::ate);

}

...

# 22.

1. Які дії можна виконувати над покажчиками? Навести приклади.

Указатели поддерживают ряд операций: присваивание, получение адреса указателя, получение значения по указателю, некоторые арифметические операции и операции сравнения.

**Присваивание**

Указателю можно присвоить либо адрес объекта того же типа, либо значение другого указателя.

Для получения адреса объекта используется операция **&**:

int a = 10;

int \*pa = &a;   // указатель pa хранит адрес переменной a

При этом указатель и переменная должны иметь один и тот же тип, в данном случае это тип int.

Присвоение указателю другого указателя:

#include <iostream>

using std::cout;

using std::endl;

int main()

{

    int a = 10;

    int b = 2;

    int \*pa = &a;

    int \*pb = &b;

    cout << "Variable a: address=" << pa << "\t value=" << \*pa << endl;

    cout << "Variable b: address=" << pb << "\t value=" << \*pb << endl;

    pa = pb;    // теперь указатель pa хранит адрес переменной b

    cout << "Variable b: address=" << pa << "\t value=" << \*pa << endl;

    return 0;

}

Когда указателю присваивается другой указатель, то фактически первый указатель начинает также указывать на тот же адрес, на который указывает второй указатель.

Нулевые указатели

Нулевой указатель (null pointer) - это указатель, который не указывает ни на какой объект. Если мы не хотим, чтобы указатель указывал на какой-то конкретный адрес, то можно присвоить ему условное нулевое значение. Для создания нулевого указателя можно применять различные способы:

int \*p1 = nullptr;

int \*p2 = NULL;

int \*p3 = 0;

Ссылки на указатели

Так как ссылка не является объектом, то нельзя определить указатель на ссылку, однако можно определить ссылку на указатель. Через подобную ссылку можно изменять значение, на которое указывает указатель или изменять адрес самого указателя:

#include <iostream>

int main()

{

    int a = 10;

    int b = 6;

    int \*p = 0;     // указатель

    int \*&pRef = p;     // ссылка на указатель

    pRef = &a;          // через ссылку указателю p присваивается адрес переменной a

    std::cout << "p value=" << \*p << std::endl;   // 10

    \*pRef = 70;         // изменяем значение по адресу, на который указывает указатель

    std::cout << "a value=" << a << std::endl;    // 70

    pRef = &b;          // изменяем адрес, на который указывает указатель

    std::cout << "p value=" << \*p << std::endl;   // 6

    return 0;

}

**Разыменование указателя**

Операция разыменования указателя представляет выражение в виде \*имя\_указателя. Эта операция позволяет получить объект по адресу, который хранится в указателе.

#include <iostream>

using std::cout;

using std::endl;

int main()

{

    int a = 10;

    int \*pa = &a;

    int \*pb = pa;

    \*pa = 25;

    cout << "Value on pointer pa: " << \*pa << endl;  // 25

    cout << "Value on pointer pb: " << \*pb << endl;  // 25

    cout << "Value of variable a: " << a << endl;    // 25

    return 0;

}

Через выражение \*pa мы можем получить значение по адресу, который хранится в указателе pa, а через выражение типа \*pa = значение вложить по этому адресу новое значение.

И так как в данном случае указатель pa указывает на переменную a, то при изменении значения по адресу, на который указывает указатель, также изменится и значение переменной a.

**Адрес указателя**

Указатель хранит адрес переменной, и по этому адресу мы можем получить значение этой переменной. Но кроме того, указатель, как и любая переменная, сам имеет адрес, по которому он располагается в памяти. Этот адрес можно получить также через операцию **&**:

int a = 10;

int \*pa = &a;

std::cout << "address of pointer=" << &pa << std::endl;        // адрес указателя

std::cout << "address stored in pointer=" << pa << std::endl;  // адрес, который хранится в указателе - адрес переменной a

std::cout << "value on pointer=" << \*pa << std::endl;          // значение по адресу в указателе - значение переменной a

|  |  |
| --- | --- |
|  |  |

**Операции сравнения**

К указателям могут применяться операции сравнения **>**, **>=**, **<**, **<=**,**==**, **!=**. Операции сравнения применяются только к указателям одного типа и к значениям **NULL** и **nullptr**. Для сравнения используются номера адресов:

#include <iostream>

using std::cout;

using std::endl;

int main()

{

    int a = 10;

    int b = 20;

    int \*pa = &a;

    int \*pb = &b;

    if(pa > pb)

        cout << "pa (" << pa << ") is greater than pb ("<< pb << ")" << endl;

    else

        cout << "pa (" << pa << ") is less or equal pb ("<< pb << ")" << endl;

    return 0;

}

**Приведение типов**

Иногда требуется присвоить указателю одного типа значение указателя другого типа. В этом случае следует выполнить операцию приведения типов с помощью операции (тип\_указателя \*):

#include <iostream>

int main()

{

    char c = 'N';

    char \*pc = &c;

    int \*pd = (int \*)pc;

    void \*pv = (void\*)pc;

    std::cout << "pv=" << pv << std::endl;

    std::cout << "pd=" << pd << std::endl;

    return 0;

Для преобразования указателя к другому типу в скобках перед указателем ставится тип, к которому надо преобразовать. Причем если мы не можем просто создать объект, например, переменную типа void, то для указателя это вполне будет работать. То есть можно создать указатель типа void.

2. Створити дві функції:

- знаходження суми елементів квадратної матриці;

- сортування головної діагоналі квадратної матриці.

#include <iostream>

using namespace std;

int sumMatr(int\* matr, int n)

{

int res = 0;

for (int i = 0; i <n; ++i)

for (int j = 0; j <n; ++j)

res+=matr[i\*n+j];

return res;

}

void sortMatr (int \*matr, int n)

{

for (int i = 0; i<n; ++i)

for (int j = 0; j<n+1; ++j)

if(matr[j\*n+j]>matr[(j+1)\*n+j+1])

{

int c=matr[j\*n+j];

matr[j\*n+j]=matr[(j+1)\*n+j+1];

matr[(j+1)\*n+j+1]=c;

}

}

int main()

{

const int n = 4;

int matr[n][n];

for (int i = 0; i<n; ++i)

{

for (int j = 0; j<n; ++j)

{

matr[i][j]=rand()%21-10;

cout<<matr[i][j]<<"\t";

}

cout<<endl;

}

int s = sumMatr(&matr[0][0], n);

cout <<"Summa = "<< s << endl;

sortMatr (&matr[0][0], n);

cout<<"\nOtsortirovanaya glavnaya diagonal"<<endl;

for (int i = 0; i<n; ++i)

{

for (int j = 0; j<n; ++j)

{

cout<<matr[i][j]<<"\t";

}

cout<<endl;

}

system ("pause");

return 0;

}

3. Обчислити  де  
*c*=0,5; *b*= -0,5; *x*=0,61; *y*=1,2.

#include <iostream>

#include <math.h>

#include <stdlib.h>

using namespace std;

int main()

{

double c = 0.5;

double b = -0.5;

double x = 0.61;

double y = 1.2;

double P , S , a ,d;

a = fabs(c\*pow(x,2)+b\*pow(y,2));

d = c\*pow(x,3)+b\*pow(y,2);

P = fabs(pow(sin(a),3)/(pow(fabs(d),1/3)))+tan(d);

S = pow(cos(pow(x,3)),2)-x\*(x-b);

cout <<"P = "<<P<<endl<<"S = "<<S<<endl;

system ("pause");

return 0;

}

4. Створити макропідстановку (макрос) який підраховує кількість від’ємних елементів в одновимірному масиві.

#include<iostream>

#include<cmath>

using namespace std;

#define calculateNegative(mas,n,c){\

for(int i = 0; i < n; ++i){\

if(0 > mas[i]){\

++c;\

}}}

int main()

{

const int n = 10;

int mas[n];

for(int i=0;i<n;i++)

{

mas[i]=rand()%21-10;

cout<<mas[i]<<"\t";

}

int c = 0;

calculateNegative(mas,n,c);

cout<<c<<endl;

system("pause");

return 0;

}

# 23.

Теория:

1. Яким чином можна ввести інформацію(на екран/у файл)? Навести приклади.

4. Функції введення/виведення С-рядків. Навести приклади буферного введення рядків.

**Чтение и запись в языке С**

int printf ( const char \* format, ... );

Функция printf() записывает в stdout аргументы из списка arg-list под управлением строки, на которую указывает аргумент format.

Строка, на которую указывает format, состоит из объектов двух различных назначений. Во-первых, это символы, которые сами должны быть выведены на экран. Во-вторых, это специфика­торы формата, определяющие вид, в котором будут выведены аргументы из списка arg-list.

Пример:  
printf ("Hi%с %d %s", '!', 10, "there!");  
  
приведет к выводу «Hi! 10 there!».

Существует парная функция для чтения

int scanf ( const char \* format, ... );

Функция scanf() является процедурой ввода общего назначения, считывающей данные из пото­ка stdin. Она может считывать данные всех базовых типов и автоматически конвертировать их в нужный внутренний формат.

Принцип работы с ней аналогичен printf

|  |  |
| --- | --- |
| %с | Символ типа char |
| %d | Десятичное число целого типа со знаком |
| %i | Десятичное число целого типа со знаком |
| %е | Научная нотация (е нижнего регистра) |
| %Е | Научная нотация (Е верхнего регистра) |
| %f | Десятичное число с плавающей точкой |
| %g | Использует код %е или %f — тот из них, который короче (при использовании %g используется е нижнего регистра) |
| %G | Использует код %Е или %f — тот из них, который короче (при использовании %G используется Е верхнего регистра) |
| %о | Восьмеричное целое число без знака |
| %s | Строка символов |
| %u | Десятичное число целого типа без знака |
| %х | Шестнадцатиричное целое число без знака (буквы нижнего регистра) |
| %Х | Шестнадцатиричное целое число без знака (буквы верхнего регистра) |
| %р | Выводит на экран значение указателя |
| %n | Ассоциированный аргумент — это указатель на переменную целого типа, в которую помещено количество символов, записанных на данный момент |
| %% | Выводит символ % |

Есть несколько способов работы с файлами c использованием языков.   
  
Самый распространенный связан со структурой FILE (это не класс, потому что сущность языка C). Эта структура определена в заголовочном файле стандартной библиотеки <stdio.h>. Размер этой структуры и ее поля зависят от ОС и от версии компилятора. Например:   
  
FILE \*f = fopen("file1.txt", "r");   
  
fopen -- функция из стандартной библиотеки. Первый параметр -- имя файла (в текущем каталоге). Второй параметр задает режим открытия файла; в данном случае "r" означает, что файл будет открыт только для чтения. Эта функция возвращает ненулевой указатель, если открытие прошло успешно; и возвращает NULL, если произошла ошибка. Ошибка может возникать в следующих ситуациях:

1. не существует файла;
2. у программы недостаточно прав доступа для работы с файлом;

Для дальнейшей корректной работы следует писать примерно такой код:  
  
if (f == NULL) {  
  // файл не удалось открыть  
}  
else {  
  // Работа с файлом  
}  
  
Допустим, что нам удалось открыть файл, т.е. f != NULL. Тогда для того, чтобы считывать файл, можно использовать функцию:   
  
fscanf(f, "%s", ptr);   
  
Эта функция работает аналогично функции scanf.

Если мы хотим записать в файл что-то, то мы должны сначала открыть его на запись:   
  
FILE \*f = fopen("file2.html", "w");

Затем можно использовать функцию fprintf(f, ...)

Открыть файлы можно поразному. Вот список допустимых флагов:

| **Режим** | **Значение** |
| --- | --- |
| "r" | Создает файл для записи (по умолчанию файл открывается как текстовый). |
| "w" | Создает файл для записи (по умолчанию файл открывается как текстовый). |
| "а" | Дописывает информацию к концу файла (по умолчанию файл открывается как текстовый). |
| "rb" | Открывает двоичный файл для чтения. |
| "wb" | Создает двоичный файл для записи. |
| "ab" | Дописывает информацию к концу двоичного файла. |
| "r+" | Открывает файл для чтения/записи (по умолчанию файл открывается как текстовый). |
| "w+" | Создает файл для чтения/записи (по умолчанию файл открывается как текстовый). |
| "а+" | Дописывает информацию к концу файла или создает файл для чтения/записи (по умолчанию открывается как текстовый файл). |
| "r+b" | Открывает двоичный файл для чтения / записи. |
| "w+b" | Создает файл для чтения / записи. |
| "а+b" | Дописывает информацию к концу файла или создает двоичный файл для чтения. |

На самом деле не все файлы выглядят как текст. Файле могут быть записаны числовые данные.  
  
size\_t fread(void \*ptr, size\_t size, size\_t nelts, FILE \*f);

* void \*ptr -- указатель на ту область памяти, в которую мы читаем;
* size\_t size -- размер элемента, который мы читаем;
* size\_t nelts -- максимальное количество элементов, которые можно записать;
* FILE \*f -- файл, из которого читаем;
* size\_t fread() -- сама функция возвращает количество элементов, которые удалось прочитать.

Есть парная функция:  
  
size\_t fwrite(const void \*ptr, size\_t size, size\_t nelts, FILE \*F);  
  
Аналогично fread эта функция возвращает количество элементов, которые удалось записать.  
Тут параметр nelts просто показывает, сколько элементов надо вывести.

Хороший способ чтения из файла дает функция fgets()

char \*fgets(char \*buffer, size\_t length, FILE \*file);  
Тут

* buffer -- это указатель на буфер, в который мы читаем;
* length -- это размер буфера;
* file -- это файл, из которого мы читаем (если читаем с клавиатуры, то разумно использовать stdin).
* Функция возвращает строку

Эта функция делает примерно следующее. Она читает из файла file в буфер buffer не больше length-1 символов. Функция может прочитать не все length-1 символов в том случае, если она встретит конец строки, либо конец файла. Функция читает length-1 символ потому, что последний символ функция добавляет сама -- '\0'. Главное отличие от scanf'а заключается в том, что функция перестанет читать в тот момент, когда закончится буфер.

Парная функция char \*fputs(const char \*str, FILE \*file);

**Потоки**

Есть три стандартные константы:  
FILE \*stdin  
FILE \*stdout  
FILE \*stderr

stdin -- это стандартный файл (поток) ввода, а stdout -- стандартный файл (поток) вывода. Таким образом:  
scanf(...) в точности эквивалентно fscanf(stdin, ...)  
printf(...) в точности эквивалентно fprintf(stdout, ...)

stderr -- это стандартный файл (поток) ошибок. По умолчанию выводит данные на экран.  
Но существует заметное отличие этого потока от stdin и stdout: **stderr -- небуферизованный файл (поток).** Поэтому в этот файл (поток) все байты уходят без "задержки", которая могла бы возникнуть при буферизированном подходе.

**Чтение и запись в языке С++**

В языке C++ объекты для работы с файлами называются потоками (streams). В данном случае слово "поток" означает то же самое, что и "файл" в языке C.

В header'е <iostream> объявлена глобальная переменная std::cout; она используется как стандартный поток вывода на экран. Эта переменная является объектом класса std::ostream.  
В этом классе есть перегруженный оператор <<, который выводит на экран:  
  
std::cout << 1; В данном случае на экран будет выведена единица.

Еще в header'е <iostream> объявлены переменные std::cin и std::cerr для стандартного потока ввода и потока ошибок соответственно. Они являются объектами классов std::istream и std::ostream соотсветственно.  
  
Аналогично тому, как stderr отличается от stdin, в языке C++ std::cerr отличается от std::cout отсутствием буферизации.  
  
В классе std::istream есть перегруженный оператор >>. Можно считывать информацию из стандартного потока ввода (с клавиатуры).

**Практика:**

1. Створити функцію знаходження добутку ненульових елементів, розташованих нижче головної діагоналі у квадратній матриці.

#include<iostream>

#include<ctime>

using namespace std;

int \*\*createSqMatr(int sz)

{

if (sz < 1)

return NULL;

int \*\*matr = new int\*[sz];

for (int i = 0; i < sz; i++)

matr[i] = new int[sz];

return matr;

}

void randSqMatr(int \*\*matr, int sz)

{

for (int i = 0; i < sz; i++)

for (int j = 0; j < sz; j++)

matr[i][j] = rand() % 10;

}

void printSqMatr(int \*\*matr, int sz)

{

for (int i = 0; i < sz; i++, cout << endl)

for (int j = 0; j < sz; j++)

cout << matr[i][j] << ' ';

}

void deleteSqMatr(int \*\*&matr, int sz)

{

for (int i = 0; i < sz; i++)

delete[] matr[i];

delete[] matr;

matr = NULL;

}

int multUnderDiagonal(int \*\*matr, int sz)

{

int sum = 1;

for (int i = 1; i < sz; i++)

for (int j = sz - i; j < sz; j++)

sum \*= matr[i][j];

return sum;

}

int main()

{

srand(time(NULL));

int sz = 0;

cin >> sz;

int \*\*matr = createSqMatr(sz);

randSqMatr(matr, sz);

printSqMatr(matr, sz);

int sum = multUnderDiagonal(matr, sz);

cout << endl << "Sum under main diagonal: " << sum << endl << endl;

deleteSqMatr(matr, sz);

cout << endl;

system("pause");

return 0;

}

3. Обчислити   де x=0,92; *y*=5,3; *a*=0,25.

#include <cmath>

#include <iostream>

#define pi 3.1485

int main()

{

double x = 0.92;

double y = 5.3;

double a = 0.25;

double A = (cos(pow(x, 3) - pi/2))/(1 + log10(y - 2.0)) + pow((y-2.0), (1/3));

double B = (pow(a, 2\*x) + pow(y, -x) \* cos(a+x)\*x)/(fabs(x-1.0));

cout << "A = " << A << "\tB = " << B;

cout << endl;

system("pause");

return 0;

}

# 24.

1.Выводить информацию можно в файл или на экран

**Экран**

int printf ( const char \* format, ... );

Функция printf() записывает в stdout аргументы из списка arg-list под управлением строки, на которую указывает аргумент format.

Строка, на которую указывает format, состоит из объектов двух различных назначений. Во-первых, это символы, которые сами должны быть выведены на экран. Во-вторых, это специфика­торы формата, определяющие вид, в котором будут выведены аргументы из списка arg-list.

Пример:  
printf ("Hi%с %d %s", '!', 10, "there!");  
  
приведет к выводу «Hi! 10 there!».

|  |  |
| --- | --- |
| %с | Символ типа char |
| %d | Десятичное число целого типа со знаком |
| %i | Десятичное число целого типа со знаком |
| %е | Научная нотация (е нижнего регистра) |
| %Е | Научная нотация (Е верхнего регистра) |
| %f | Десятичное число с плавающей точкой |
| %g | Использует код %е или %f — тот из них, который короче (при использовании %g используется е нижнего регистра) |
| %G | Использует код %Е или %f — тот из них, который короче (при использовании %G используется Е верхнего регистра) |
| %о | Восьмеричное целое число без знака |
| %s | Строка символов |
| %u | Десятичное число целого типа без знака |
| %х | Шестнадцатиричное целое число без знака (буквы нижнего регистра) |
| %Х | Шестнадцатиричное целое число без знака (буквы верхнего регистра) |
| %р | Выводит на экран значение указателя |
| %n | Ассоциированный аргумент — это указатель на переменную целого типа, в которую помещено количество символов, записанных на данный момент |
| %% | Выводит символ % |

В языке C++ объекты для работы с файлами называются потоками (streams). В данном случае слово "поток" означает то же самое, что и "файл" в языке C.

В header'е <iostream> объявлена глобальная переменная std::cout; она используется как стандартный поток вывода на экран. Эта переменная является объектом класса std::ostream.  
В этом классе есть перегруженный оператор <<, который выводит на экран:  
  
std::cout << 1; В данном случае на экран будет выведена единица.

**В Файл**

Если мы хотим записать в файл что-то, то мы должны сначала открыть его на запись:   
  
FILE \*f = fopen("file2.html", "w");

Затем можно использовать функцию fprintf(f, ...)

Открыть файлы можно поразному. Вот список допустимых флагов:

| **Режим** | **Значение** |
| --- | --- |
| "r" | Создает файл для записи (по умолчанию файл открывается как текстовый). |
| "w" | Создает файл для записи (по умолчанию файл открывается как текстовый). |
| "а" | Дописывает информацию к концу файла (по умолчанию файл открывается как текстовый). |
| "rb" | Открывает двоичный файл для чтения. |
| "wb" | Создает двоичный файл для записи. |
| "ab" | Дописывает информацию к концу двоичного файла. |
| "r+" | Открывает файл для чтения/записи (по умолчанию файл открывается как текстовый). |
| "w+" | Создает файл для чтения/записи (по умолчанию файл открывается как текстовый). |
| "а+" | Дописывает информацию к концу файла или создает файл для чтения/записи (по умолчанию открывается как текстовый файл). |
| "r+b" | Открывает двоичный файл для чтения / записи. |
| "w+b" | Создает файл для чтения / записи. |
| "а+b" | Дописывает информацию к концу файла или создает двоичный файл для чтения. |

На самом деле не все файлы выглядят как текст. Файле могут быть записаны числовые данные.

Эта функция дает нам записать информацию в бинарном виде  
size\_t fwrite(const void \*ptr, size\_t size, size\_t nelts, FILE \*F);  
  
Аналогично fread эта функция возвращает количество элементов, которые удалось записать.  
Тут параметр nelts просто показывает, сколько элементов надо вывести.

**Потоки**

Есть три стандартные константы:  
FILE \*stdin  
FILE \*stdout  
FILE \*stderr

stdin -- это стандартный файл (поток) ввода, а stdout -- стандартный файл (поток) вывода. Таким образом:  
scanf(...) в точности эквивалентно fscanf(stdin, ...)  
printf(...) в точности эквивалентно fprintf(stdout, ...)

2. Створити дві функції:

- знаходження мінімального елемента квадратної матриці;

- сортування заданого стовпчика матриці; матриця та номер стовпчика передаються у якості параметрів функції.

int minElSqMatr(int \*\* matr, int n, int m)

{

int min = matr[0][0];

for (int i = 0; i < n; i++)

for (int j = 0; j < m; j++)

if (matr[i][j] < min)

min = matr[i][j];

return min;

}

void sortCol(int \*\*matr, int sz, int col)

{

int \*\*p = new int\*[sz];

for (int i = 0; i < sz; i++)

p[i] = &matr[i][col];

for (int i = 0; i < sz - 1; i++)

for (int j = 0; j < sz - i - 1; j++)

if (\*p[j] > \*p[j+1])

{

int temp = \*p[j];

\*p[j] = \*p[j + 1];

\*p[j+1] = temp;

}

delete[] p;

}

3. Обчислити  

#include <iostream>

#include <cmath>

using namespace std;

int main()

{

double m = 0.7, x = 1.77, a = 0.5, b = 1.08;

double Z = 0, S = 0;

Z = (sin(x) / (pow((1 + pow(m, 3)\*pow(sin(x), 2)), 1.0 / 2))) - m \* x\*log(m\*x);

S = exp(-a \* x)\*pow((x + 1), 1.0 / 2) + exp(-b \* x)\*pow(fabs(x - 3), 1.0 / 3);

system("pause");

return 0;

}

4. Принцип роботи з бінарними файлами. Навести приклад.

Работа с двоичными файлами. Двоичные  файлы  отличаются  от  текстовых  тем,  что  в  них  записана  информация  во внутреннем машинном представлении. Двоичный файл нельзя просмотреть на экране (вернее, можно просмотреть, но очень сложно понять). Но есть и преимущества - из двоичных файлов можно читать сразу весь массив в виде единого блока. Также можно записать весь массив или его любой непрерывный кусок за одну команду. При   открытии   двоичного   файла   вместо   режимов  "r”,  "w”   и   "a”   используют соответственно "rb”, "wb” и "ab”. Дополнительная буква "b” указывает на то, что файл двоичный   (от   английского   слова   binary   -   двоичный).   Решение   предыдущей   задачи, использующее двоичный файл, показано ниже.

**Пример**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <ctime>

using namespace std;

int main()

{

srand(time(NULL));

int sz = 0;

cout << "Input sz ->";

cin >> sz;

int \* arr = new int[sz];

for (int i = 0; i < sz; i++)

{

arr[i] = rand() % 10;

cout << arr[i] << ' ';

}

FILE \*f = fopen("binFile.bin", "wb"); //Пишем бинарный

if (!f)

cout << "Can't open file!" << endl;

else

{

fwrite(&sz, sizeof(int), 1, f);

fwrite(arr, sizeof(int), sz, f);

}

fclose(f);

cout << endl;

int \*newArr = NULL, newSz = 0;

f = fopen("binFile.bin", "rb"); //Читаем бинарный

if (!f)

cout << "Can't open file!" << endl;

else

{

fread(&newSz, sizeof(int), 1, f);

newArr = new int[newSz];

fread(newArr, sizeof(int), newSz, f);

}

fclose(f);

cout << "From bin file: " << endl;

for (int i = 0; i < newSz; i++)

cout << newArr[i] << ' ';

delete[] newArr;

delete[] arr;

system("pause");

return 0;

}

# 25.

1. Що таке розадресування у мові С++? Навести приклади.

Эти операции используются для работы с переменными типа указатель.

Операция разадресации (\*) осуществляет косвенный доступ к адресуемой величине через указатель. Операнд должен быть указателем. Результатом операции является величина, на которую указывает операнд. Типом результата является тип величины, адресуемой указателем. Результат не определен, если указатель содержит недопустимый адрес.

Рассмотрим типичные ситуации, когда указатель содержит недопустимый адрес:

* указатель является нулевым;
* указатель определяет адрес такого объекта, который не является активным в момент ссылки;
* указатель определяет адрес, который не выровнен до типа объекта, на который он указывает;
* указатель определяет адрес, не используемый выполняющейся программой.

Операция адрес (&) дает адрес своего операнда. Операндом может быть любое именуемое выражение. Имя функции или массива также может быть операндом операции «адрес», хотя в этом случае знак операции является лишним, так как имена массивов и функций являются адресами. Результатом операции адрес является указатель на операнд. Тип, адресуемый указателем, является типом операнда.

Операция адрес не может применятся к элементам структуры, являющимися полями битов, и к объектам с классом памяти register.

**Примеры:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | int t, f=0, \* adress;  adress = &t */\* переменной adress, объявляемой как*  *указатель, присваивается адрес переменной t \*/*  \* adress =f; */\* переменной находящейся по адресу, содержащемуся*  *в переменной adress, присваивается значение*  *переменной f, т.е. 0 , что эквивалентно*  *t=f; т.е. t=0; \*/* |

#include <stdio.h>

int main(){

unsigned long int A = 0Xcc77ffaa;

unsigned short int\* pint = (unsigned short int\*) &A;

unsigned char\* pchar = (unsigned char \*) &A;

printf("| %x | %x | %x |", A, \*pint, \*pchar);

return 0;

}

1. Розробити функцію знаходження суми від'ємних елементів одновимірного масиву та сортування масиву за зростанням.

#include <iostream>

#include <ctime>

using namespace std;

void SortMass(int\* m, int n)

{

for (int i = n - 1; i >= 1; i--)

for (int j = 0; j < i; j++)

{

if (m[j] > m[j + 1])

{

int foo = m[j];

m[j] = m[j + 1];

m[j + 1] = foo;

}

}

}

int sumMatr(int\* m, int n)

{

int s = 0;

for (int i = n-1; i >=1 ; i--)

for (int j = 0; j < i; j++)

if (m[j] < 0)

s += m[j];

return s;

}

int main()

{

cout << "Bedute n: ";

int n = 0;

int\* m;

cin >> n;

m = new int[n];

srand(time(0));

for (int i = 0; i < n; i++)

{

int x = rand() % 21-10;

m[i] = x;

cout << m[i] << ' ';

}

cout << endl;

SortMass(m, n);

cout << "After sort" << endl;

for (int i = 0; i < n; i++)

cout << m[i] << ' ';

int s=sumMatr(m,n);

cout << "Summa = " << s << endl;

cin.get();

cin.get();

}

3. Обчислити  *b=(1+tg2 (x/2))ln x*; де *x*=1,23; *y*=0,79; *z*=0,9.

#include <iostream>

#include <math.h>

using namespace std;

int main()

{

double x = 1.23;

double y = 0.79;

double z = 0.9;

double a = y + (x / (pow(y, 3) + abs((pow(x, 2) / (y + pow(x, (2 / 3)))))));

double b = (1 + tan(x / 2)\*tan(x / 2)\*log(x));

cout << "a = " << a << "\tb = " << b;

system ("pause");

}

4.Що таке директива умовної компіляції? Навести приклади.

Существует несколько директив условной компиляции, позволяющих изменять порядок компиляции программы в зависимости от состояния системы. Данный процесс называется условной компиляцией и широко используется при разработке коммерческого программного обеспечения, предоставляющего и поддерживающего много различных версий одной и той же программы.

Если после #if константное выражение принимает истинное значение, то код между #if и #endif компилируется, в противном случае код пропускается. Директива #endif используется для обо­значения конца блока #if.  
Стандартный вид #if следующий:  
  
#if константное\_выражениепоследовательность операторов#endif

Например:  
/\* простой пример с #if \*/  
#include <stdio.h>  
#define MAX 100  
int main(void)  
{  
#if MAX>99  
printf("Compiled for array greater than 99.\n");  
#endif  
return 0;  
}

Данная программа выводит сообщение на экран, поскольку, как определено в программе, МАХ имеет значение больше 99. Данный пример демонстрирует очень важный момент: выражение, следующее зa #if, вычисляется на этапе компиляции, следовательно, оно должно содержать ранее определенные идентификаторы и константы, а не переменные.

Работа #else во многом похожа на работу оператора else — она предоставляет альтернативный вариант, если #if содержит ложное состояние. Предыдущий пример можно расширить следую­щим образом:

/\* простой пример с #if / #else \*/  
#include <stdio.h>  
#def ine MAX 10  
int main(void)  
{  
# if MAX>99  
printf("Compiled for array greater than 99.\n");  
#else  
printf("Compiled for small array.\n");  
#endif  
return 0;  
}

В данном случае MAX определяется так, чтобы значение было меньше 99, в результате чего ком­пилируется не код, соответствующий #if, а код, соответствующий #else. Следовательно, выводится сообщение «Compiled for small array».

Обратим внимание, что #else используется для обозначения конца блока #if и начала блока #else. Это делается потому, что может быть только один #endif, связанный c #if.

#elif означает «иначе если» и используется для построения лесенки if-else-if с целью определе­ния различных опций компиляции. За #elif следует константное выражение. Если выражение истинно, то блок кода компилируется и остальные выражения не проверяются. В противном слу­чае рассматривается следующий блок. Стандартный вид #elif следующий:  
  
#if выражениепоследовательность операторов#elif выражение 1последовательность операторов#elif выражение 2последовательность операторов#elif выражение 3последовательность операторов#elif выражение 4последовательность операторов...#elif выражение Nпоследовательность операторов#endif

Например, следующий фрагмент использует значение ACTIVE\_COUNTRY для определения де­нежного знака:  
  
#define US 0  
#define ENGLAND 1  
#define FRANCE 2  
#define ACTIVE\_COUNTRY US  
#if ACTIVE\_COUNTRY==US  
char currency[] = "dollar";  
#elif ACTIVE\_COUNTRY==ENGLAND  
char currency[] = "pound";  
#else  
char currency[] = "franc";  
#endif

#if и #elif могут быть вложенными. Если это имеет место, то каждый #endif, #else или #elif ассоциируется с ближайшим #if или #elif. Например, следующее совершенно корректно:  
  
#if МАХ>100  
#if SERIAL\_VERSION int port = 198;  
#elif  
int port = 200;  
#endif  
#else  
char out\_buffer[100] ;  
#endif

В Borland С++ в операторе #if можно использовать оператор времени компиляции sizeof. На­пример, следующая программа определяет, компилируется программа для больших или малень­ких массивов:  
  
#if (sizeof(char \*) == 2)  
printf("Program compiled for small array.");  
#else  
printf("Program compiled for large array.");  
#endif

Другой метод условной компиляции состоит в использовании директив #ifdef и #ifndef, что соответ­ственно означает «если определено» и «если не определено». Стандартный вид #ifdef следующий:  
  
#ifdef имя\_макросапоследовательность операторов#endif

Если имя макроса определено ранее в операторе #define, то последовательность операторов, сто­ящих между #ifdef и #endif, будет компилироваться. Стандартный вид #ifndef следующий:  
  
#ifndef имя\_макросапоследовательность операторов#endif

Если имя макроса не определено ранее в операторе #define, то последовательность операторов, стоящих между #ifdef и #endif, будет компилироваться.

Kaк #ifdef, так и #ifndef могут использовать оператор #else, но нe #elif. Например:  
  
#include <stdio.h>  
#define TED 10  
int main(void)  
{  
#ifdef TED  
printf("Hi Ted\n");  
#else  
printf("Hi anyone\n");  
#endif  
#ifndef RALPH  
printf("RALPH not defined\n");  
#endif  
return 0;  
}

выводит «Hi Ted» и «RALPH not defined». Если TED не определен, то выведется «Hi anyone», a за ним «RALPH not defined».

#ifdef и #ifndef можно вкладывать друг в друга так же, как и #if.

# 26.

1. Які дії можна виконувати над покажчиками? Навести приклади.

Покажчики — це змінні, котрі містять адресу пам’яті, роз­поділеної для об’єкта відповідного типу. Усі змін­ні, розглянуті до цього, зберігали якісь значення (дані). Ці дані могли бути різних типів: символьного, цілого, дійсного тощо. При оголошенні змінної-покажчика слід вказати тип даних, адресу яких буде містити змінна, та ім’я покажчика з симво­лом «\*».

Загальний формат опису покажчика має вигляд:

тип \*ім’я;

де тип — тип значень, на який вказує покажчик;

ім’я — ім’я змінної-покажчика;

«\*» — операція над типом, що читається «покажчик на тип».

Наприклад:

int \*рn – покажчик на ціле значення;

float \*pf1, \*pf2; — два покажчики на дійсні значення.

Покажчики не прив’язують дані до якого-небудь визначе­ного імені змінної і можуть містити адреси будь-якого неімено­ваного значення. Існує адресна константа NULL, що означає порожню адресу.

Слід нагадати, що мова C++ налічує лише дві операції, які стосуються адрес змінних, а саме:

«&» — операція взяття адреси («адреса значення»);

«\*» — операція розіменування («значення за адресою»).

Операція взяття адреси «&» застосовується разом зі змін­ною і повертає адресу цієї змінної. Операція розіменування «\*» використовується разом з покажчиками і вилучає значення, на яке вказує змінна-покажчик, розташована безпосередньо після символа «\*».

Оголошення покажчиків можна здійснити одним з таких способів:

<тип> \*ptr;

<тип> \*ptr = <змінна-покажчик>;

<тип> \*ptr = &<ім’я змінної>;.

Наприклад:

int \*ptx, b; float у; — оголошені змінна-покажчик ptx та змінні b і у;

float \*sp = &у; — покажчику sp присвоюється адреса змінної у;

float \*р = sp; — покажчику р присвоюється значення (адреса зна­чення), яке міститься в змінній sp, тобто адреса змінної у.

При оголошенні покажчиків символ «\*» може знаходитися перед ім’ям покажчика або відразу після оголошення типу по­кажчика і поширювати свою дію тільки на одну змінну-покаж­чик, перед якою він записаний:

long \*pt; long\*Uk; int \*ki, x, h; — оголошення описів.

За потреби для опису покажчика на комірку довільного типу замість ідентифікатора типу записується слово void, а саме:

void \*р, \*pt; — опис двох покажчиків на довільний тип даних.

Перед використанням покажчика у програмі його обо­в’язково необхідно ініціювати, іншими словами, необхідно присвоїти адресу якого-небудь даного, інакше можуть бути не­передбачені результати.

Для одержання доступу до значення змінної, адреса якої зберігається в покажчику, досить у відповідному операторі про­грами записати ім’я покажчика з символом «\*» — здійснити операцію розіменування.

Розглянемо фрагмент програми з поясненнями:

int \*р, \*р1; — оголошені два покажчики на комірку пам’яті типу int;

int х = 12, у = 5, m[7]; — оголошені змінні х, у і масив m, змінні ініційовані;

р = &у; // р (&у); — покажчику р присвоєна адреса змінної у.

Якщо для цього фрагмента програми записати оператор ви­ведення у вигляді

cout << “Адрес р ” << р << “Значение по этому адресу = ” << \*р;,

то виведеться адреса комірки пам’яті, де записана змінна у і зна­чення цієї змінної (тобто 5).

Використовуючи запис х = \*р;, одержимо х = 5, тому щo \*р = у = 5;.

Змінити величину параметра у можна так:

у = 10; // \*Р= 10;

\*р = \*р+5; //у +=5;.

Остання операція означає збільшення значення змінної у цi­лого типу на 5, тобто у= 15.

При ініціюванні покажчиків їм можна присвоювати або адресу об’єкта (змінної), або адресу конкретного місця пам’яті (масиву), або число 0 (нуль), а саме:

int \*pt = (char \*) 0x00147; — присвоюється адреса комірки;

int \*arrpt = new int [10]; — визначається початкова адреса розмі­щення динамічного масиву;

char \*р = 0; — здійснюється ініціювання нулем.

Оскільки покажчики — це спеціальні змінні, то в операці­ях з іншими покажчиками вони можуть використовуватися без символа «\*», тобто без розкриття посилання, наприклад:

float \*pt1, \*pt2, х=15, m[5];

pt1 = &x;

pt2 = pt1;

pt1 = m; //pt1 = &m[0];

де m — ім’я масиву, що розглядається як спеціальний покаж­чик-константа.

Приклад 6.6. Написати ілюстраційну програму з використанням покажчиків.

// P6\_6.CPP — применение указателей

#include <iostream.h>

#include <conio.h>

int main ( )

{ int x = 10;

int \*px (&x); // int \*px = &x;

cout << "x =" << x << endl;

cout << "\*px =" << \*px << endl;

x \*= 2; //x=x\*2;

cout << "Новое значение \*px = " << \*px << endl;\*

px += 2; // \*px=\*px + 2;

cout << "Результат \*px, т. e. x = " << x << endl;

getch(); //задержка экрана

}

Результат виконання програми:

х = 10

\*рх = 10

Новое значение \*рх = 20

Результат \*рх, т. е. х = 22.

4. Розробити функцію, що знаходить індекси максимального і мінімального елементів.

#include<ctime>

#include <iostream>

#include <cstdlib>

using namespace std;

void randMatr(int\* matr, int N)

{

setlocale(LC\_ALL, "RUS");

for (int i = 0; i < N; i++) {

cout << "[" << i + 1 << "]" << ":";

cin >> matr[i];

}

cout << "\nВаш массив: ";

for (int i = 0; i < N; ++i) {

cout << matr[i] << " ";

}

}

/\*void outputMatr(int\* matr, int N)

{

for (int i = 0; i < N; ++i)

cout << matr[i] << " ";

cout << endl;

}\*/

int MINelement(int \*matr, int N, int &imin)

{

int min = matr[0];

for (int i = 0; i < N; ++i)

{

if (min > matr[i]) {

min = matr[i];

imin = i;

}

}

return imin;

}

int MAXelement(int \*matr, int N, int &imax)

{

int max = matr[0];

for (int i = 0; i < N; ++i)

{

if (max < matr[i]) {

max = matr[i];

imax = i;

}

}

return imax;

}

int main()

{

const int N = 10;

int matr[N];

int imin, imax = 0;

int mas2d[N];

int i;

randMatr(&mas2d[0], N);

int min = MINelement(&mas2d[0], N, imin);

cout << "Min: " << imin << endl;

int max = MAXelement(&mas2d[0], N, imax);

cout << "max: " << imax << endl;

//outputMatr(&mas2d[0], N);

system("pause");

return 0;

}

1. Обчислити ; де a=16,5; b=3,4; x=0,61.



#include <iostream>

#include <math.h>

using namespace std;

int main()

{

double a = 16.5;

double b = 3.4;

double x = 0.61;

double c = ((b\*x\*x) - a) / (exp(a\*x) - 1);

double S = ((pow(x, 3)\*pow(tan(pow(x + b, 2)), 2) + a)\*c) / (sqrt(x + b));

double b = (1 + tan(x / 2)\*tan(x / 2)\*log(x));

cout << "S = " << S << "\tb = " << b;

system("pause");

}

1. Як за допомогою умовної компіляції виключити повторне підключення файлу. Навести приклад.

Защита от множественного подключения заголовочного файла

Какой метод лучше использовать:

#pragma once

или

#ifndef XXX\_H

#define XXX\_H

...

#endif

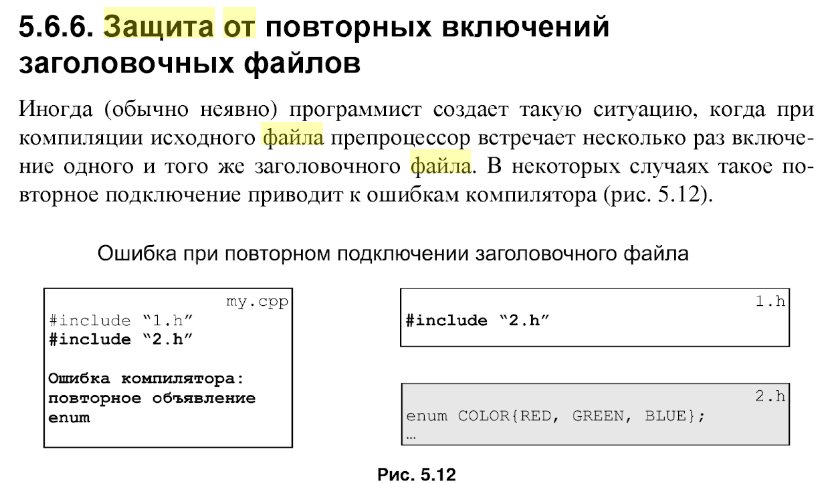
У обоих методов есть свои преимущества и недостатки.

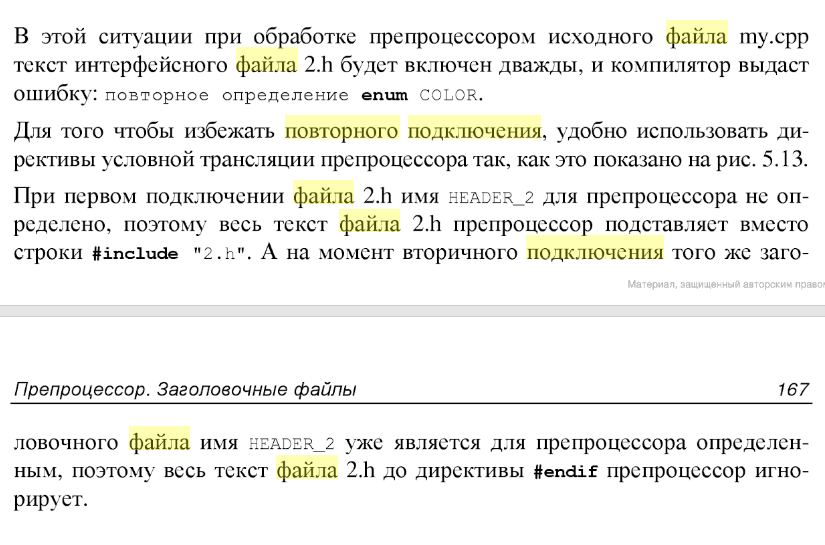
#define кроссплатформенный, соответствует стандарту, поддерживается всеми компиляторами, но у него есть очевидный недостаток: если в проекте случайно найдутся два header-файла с одинаковым именем, необъяснимые ошибки гарантированы. Особенно неприятно это в случае больших проектов с 3rdparty-кодом. Можно придумать более продвинутую схему, при которой вероятность коллизии уменьшится (например, включать в guard полный путь к файлу в проекте или UUID), но гарантии вам никто не даст. (И вы не сможете потребовать от разработчиков посторонних библиотек следовать вашему стандарту.)

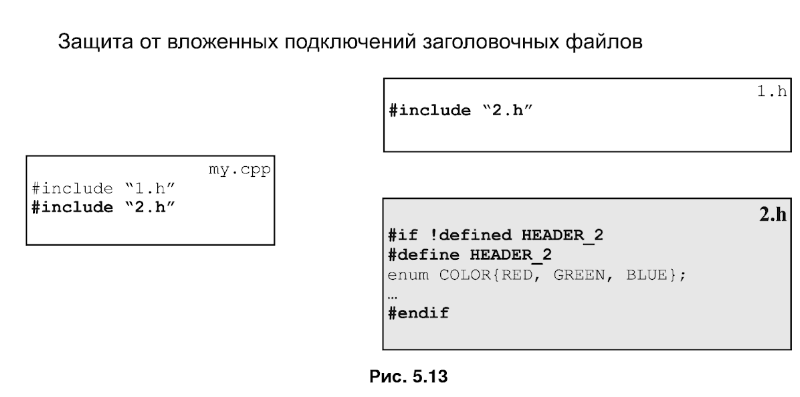
#pragma once лишено этих недостатков, потому что теперь ваше намерение прямо сообщается компилятору, а не посредством маломощного препроцесора. Но этот формат поддерживается не всеми компиляторами, поэтому наверняка будут проблемы с портируемостью.

Тем не менее, #pragma once поддерживается большим количеством компиляторов, так что если вы рассчитываете лишь на популярные компиляторы (MSVC, GCC, Clang, Intel compiler) свежих версий, имеет смысл остановиться на этом варианте. Имейте в виду, что многие опытные Unix-разработчики недолюбливают #pragma once, так что вам придётся обосновывать свой выбор.

Если ваш код есть часть какого-либо проекта, всё становится проще: просто спросите руководителя проекта о том, что рекомендуется к использованию. Выбор между include guard и #pragma once — часть coding standards любого проекта.







#ifndef \_MACROS\_

#define \_MACROS\_

//Код

#endif

Данная функция позволяет избежать конфликт имен функции, то есть исключается повторное подключение файла.

Приклад:

#ifndef \_MY\_HEADER\_

#define \_MY\_HEADER\_

#define OUTPUT(n, mas,i) cout << "Input size:\n> ";\

cin >> n;\

mas = new int[n];\

for (i = 0; i < n; ++i)\

cin >> mas[i];

#define MINN(mas,n,imin) {for (i = 1; i < n; ++i) \

if (abs(mas[i]) < abs(mas[imin])) \

imin = i; \

cout << "Absolute minimum: \n< mas[" << imin << "] = " << mas[imin] << endl;

#define SABS(mas, n, i, abssum) i = 0; \

while (mas[i] && i < n) \

++i;\

while (i < n)\

abssum += mas[i], ++i;\

cout << "Absolute sum after first zero:\n< " << abssum <<endl;}

#define CHAN(t, mas, i, n) for(int i = 0; i < n/2; i++) \

{\

int t = mas[n / 2 + i];\

mas[n / 2 + i] = mas[i];\

mas[i] = t;\

}\

for(int i = 0; i < n; i++)\

cout << " "<< mas[i] << " " ; \

#endif

# 27.

1.Оператори continue, break, goto. Навести приклади.

* Оператор **continue**используется только в циклах. В операторах for, while, do while**,**оператор continueвыполняет пропуск оставшейся части кода тела цикла и переходит к следующей итерации цикла.

ПРИМЕР:

 Разработаем программу с оператором continue. Программа должна работать циклически. Внутри цикла необходимо организовать ввод чисел. Если введено число от 0 до 10 включительно, то необходимо напечатать квадрат этого числа, иначе используя оператор continue пропустить оператор возведения в квадрат введенного числа. При введении отрицательного числа осуществить выход из цикла.

#include <iostream>

using namespace std;

int main()

{

    int in\_number; // переменная для хранения введённого числа

    do

    {

        cout << "Enter number: ";

        cin >> in\_number; // считываем введённое число в переменную in\_number

        if ( in\_number > 10 || in\_number < 0 ) // если введённое число не входит в заданный интервал

            continue; // переход на следующую итерацию цикла do while

        cout << "square = " << in\_number \* in\_number << endl; // возводим в квадрат введённое число

    }

while ( in\_number >= 0 ); // пока введённое число больше либо равно нулю цикл будет работать

    system("pause");

    return 0;

}

Оператор **break** имеет два назначения. Первое - это окончание работы оператора switch. Второе - это принудительное окончание цикла, минуя стандартную проверку условия. Данное назначение здесь и рассматривается. Когда оператор break встречается в теле цикла, цикл немедленно заканчивается и выполнение программы переходит на строку, следующую за циклом. Например:  
  
#include <stdio.h>  
int main()  
{  
int t;  
for(t=0; t<100; t++)  
{  
printf ("%d ", t) ;  
if (t==10) break;  
}  
return 0;  
}  
  
Данная программа выводит числа от 0 до 10 включительно и заканчивает работу, поскольку break вызывает немедленный выход из цикла, минуя условие t< 100.

* **Оператор goto** — это [**оператор управления порядком выполнения кода**](https://ravesli.com/urok-63-poryadok-vypolneniya-programm-operatory-upravleniya/), который заставляет ЦП сделать переход из одного участка кода на другой, осуществить так называемый прыжок. Другой участок кода идентифицируется с помощью **statement label**. Ниже приведен пример с использованием goto и statement label:

int main()

{

    double z;

tryAgain: // это statement label

    std::cout << "Enter a non-negative number";

    std::cin >> z;

    if (z < 0.0)

        goto tryAgain; // а это оператор goto

    std::cout << "The sqrt of " << z << " is " << sqrt(z) << std::endl;

    return 0;

}

В этой программе пользователю предлагается ввести неотрицательное число. Однако, если пользователь введет отрицательное число, то программа, используя оператор goto, сделает переход назад к строчке tryAgain. Затем пользователю снова нужно будет ввести число. Таким образом, мы можем постоянно запрашивать у пользователя ввод числа, пока он или она не введут корректные числа.

2. Застосування функцій покажчиків. Навести приклад

В языке программирования C функция тоже имеет адрес и может иметь указатель. Указатель на функцию представляет собой выражение или переменную, которые используются для представления адреса функции. Указатель на функцию содержит адрес первого байта в памяти, по которому располагается выполняемый код функции.

тип (\*имя\_указателя) (параметры);

#include <stdio.h>

void hello(void)

{

    printf("Hello, World \n");

}

void goodbye()

{

    printf("Good Bye, World \n");

}

int main(void)

{

    void (\*message) (void);

    message=hello;

    message();

    message = goodbye;

    message();

    return 0;

}

Кроме одиночных указателей на функции мы можем определять их массивы. Для этого используется следующий формальный синтаксис:

тип (\*имя\_массива[размер]) (параметры)

#include <stdio.h>

void add(int x, int y)

{

    printf("x+y=%d \n", x+y);

}

void subtract(int x, int y)

{

    printf("x+y=%d \n", x-y);

}

void multiply(int x, int y)

{

    printf("x\*y=%d \n", x\*y);

}

int main(void)

{

    int a = 10;

    int b = 5;

    void (\*operations[3])(int, int) = {add, subtract, multiply};

    // получаем длину массива

    int length = sizeof(operations)/sizeof(operations[0]);

    for(int i=0; i<length;i++)

    {

        operations[i](a, b);    // вызов функции по указателю

    }

    return 0;

}

Здесь массив operations содержит три функции add, subtract и multiply, которые последовательно вызываются в цикле через перебор массива в функции main.

3. Обчислити  де  a=1,5; b=15,5; x=-2,9.

#include <iostream>

#include<math.h>

using namespace std;

double square(double k)

{

k=k\*k;

return k;

}

double cube(double k)

{

k=k\*k\*k;

return k;

}

double rote(double k)

{

k=sqrt(k);

return k;

}

int main()

{

double y=0,x,P=0,a,b;

x=-2.9;

a=1.5;

b=15.5;

y=square(sin(cube(x)))-(x/(rote(square(b)+square(a))));

P=rote(square(x)+b)- square(b)\*cube(sin(x+a))/x\*y;

cout<<y<<" "<<P;

return 0;

}

4. Створити макропідстановку (макрос) для знаходження мінімального елемента одновимірного масиву. Навести приклад застосування

#include <iostream>

using namespace std;

#define MIN(mas,n) int min=0;\

for(int i=0;i<n;++i){\

if(mas[i]<min)\

min=mas[i];}\

cout<<endl<<"min is "<<min;

int main()

{

int n=10;

int\*mas=new int[n];

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

mas[i]=rand()%21-10;

cout<<mas[i]<<" ";

}

MIN(mas,n);

return 0;

}

# 28.

1. Які існують оператори циклу в мові С++? Навести приклади.

В языке Си следующие виды циклов:

* while — цикл с предусловием;
* do…while — цикл с постусловием;
* for — параметрический цикл (цикл с заданным числом повторений).

**Цикл с предусловием while**

Общая форма записи

while (Условие)  
{  
  БлокОпераций;  
}

Если **Условие** выполняется (выражение, проверяющее **Условие**, не равно нулю), то выполняется **БлокОпераций**, заключенный в фигурные скобки, затем **Условие** проверяется снова.  
Последовательность действий, состоящая из проверки **Условия** и выполнения **БлокаОпераций**, повторяется до тех пор, пока выражение, проверяющее **Условие**, не станет ложным (равным нулю). При этом происходит выход из цикла, и производится выполнение операции, стоящей после оператора цикла.  
#include <stdio.h>  
int main() {  
  int k;  // объявляем целую переменную key  
  int i = 1;  
  int sum = 0; // начальное значение суммы равно 0  
  printf("k = ");  
  scanf("%d", &k);   // вводим значение переменной k  
  while (i <= k)     // пока i меньше или равно k  
  {  
    sum = sum + i; // добавляем значение i к сумме  
    i++;           // увеличиваем i на 1  
  }  
  printf("sum = %d\n", sum); // вывод значения суммы  
  getchar(); getchar();  
  return 0;  
}

**Цикл с постусловием do...while**

Общая форма записи

do {  
  БлокОпераций;  
} while (Условие);

Цикл do...while — это цикл с постусловием, где истинность выражения, проверяющего **Условие**проверяется после выполнения **Блока Операций**, заключенного в фигурные скобки. Тело цикла выполняется до тех пор, пока выражение, проверяющее **Условие**, не станет ложным, то есть тело цикла с постусловием выполнится хотя бы один раз.  
  
Использовать цикл do...while лучше в тех случаях, когда должна быть выполнена хотя бы одна итерация, либо когда инициализация объектов, участвующих в проверке условия, происходит внутри тела цикла.

#include <stdlib.h>   
int main() {  
  int num;             // объявляем целую переменную для числа  
  
  do {  
    printf("Введите число от 0 до 10: "); // приглашение пользователю  
    scanf("%d", &num); // ввод числа  
  } while ((num < 0) || (num > 10)); // повторяем цикл пока num<0 или num>10  
  printf("Вы ввели число %d", num); // выводим введенное значение num - от 0 до 10  
  getchar(); getchar();  
  return 0;  
}

**Параметрический цикл for**

Общая форма записи

for (Инициализация; Условие; Модификация)  
{  
  БлокОпераций;  
}  
for — параметрический цикл (цикл с фиксированным числом повторений). Для организации такого цикла необходимо осуществить три операции:

* **Инициализация** - присваивание параметру цикла начального значения;
* **Условие** - проверка условия повторения цикла, чаще всего - сравнение величины параметра с некоторым граничным значением;
* **Модификация** - изменение значения параметра для следующего прохождения тела цикла.

Эти три операции записываются в скобках и разделяются точкой с запятой **;**;. Как правило, параметром цикла является целочисленная переменная.  
**Инициализация** параметра осуществляется только один раз — когда цикл for начинает выполняться.  
Проверка **Условия** повторения цикла осуществляется перед каждым возможным выполнением тела цикла. Когда выражение, проверяющее **Условие** становится ложным (равным нулю), цикл завершается. **Модификация** параметра осуществляется в конце каждого выполнения тела цикла. Параметр может как увеличиваться, так и уменьшаться.

#include <stdio.h>  
int main() {  
  int k;  // объявляем целую переменную key  
  int sum = 0; // начальное значение суммы равно 0  
  printf("k = ");  
  scanf("%d", &k);   // вводим значение переменной k  
  for(int i=1; i<=k; i++) // цикл для переменной i от 1 до k с шагом 1  
  {  
    sum = sum + i; // добавляем значение i к сумме  
  }  
  printf("sum = %d\n", sum); // вывод значения суммы  
  getchar(); getchar();  
  return 0;  
}

2. Створити функцію, яка обчислює і вертає через return суму додатних елементів одновимірного масиву.

#include <iostream>

#include <math.h>

using namespace std;

int Sum(int\*mas,int l)

{

int sum=0;

for(int i=0;i<l;++i)

{

if(mas[i]>0)

{

sum+=mas[i];

}

}

return sum;

}

int main()

{

int n;

cout<<"input quality of array"<<" ";

cin>>n;

int\*mas=new int[n];

for(int i=0;i<n;++i)

{

mas[i]=rand()%20-10;

cout<<mas[i]<<" ";

}

int s=Sum(mas, n);

cout<<endl<<s;

system("pause");

return 0;

}

3. Обчислити  де *a*=0,3; b=0,9; x=0,53.

#include <iostream>

#include<math.h>

using namespace std;

double square(double k)

{

k=k\*k;

return k;

}

double cube(double k)

{

k=k\*k\*k;

return k;

}

double rote(double k)

{

k=sqrt(k);

return k;

}

int main()

{

double R,Y,x=0.53,a=0.3,b=0.9;

Y=(pow(a,(2\*x))+pow(b,-x)\*cos(a+b)\*x)/(fabs(x-1));

R=rote((square(x)+b))- square(b)\*cube(sin(x+a))/x;

cout<<Y<<" "<<R;

return 0;

}

4.Навести приклад, як прочитати з файлу два дійсних числа.

#include <iostream>  
using namespace std;  
int main()  
{  
setlocale (LC\_ALL, «RUS»);  
int n=0; M=2  
float a;  
fstream F;  
  
F.open(«D: \\accounts.txt»);  
while (n<=M)  
{  
F>>a;  
cout<<a<<«\t«;  
n++;  
}  
F.close();

}  
system(«pause»);  
return 0;  
}