

МЕТОДИЧНА РОЗРОБКА

**для проведення лекційних занять
з НАВЧАЛЬНОЇ ДИСЦИПЛІНИ**

«Основи програмування та алгоритмічні мови»
для студентів денної форми навчання за спеціальністю
121 «Інженерія програмного забезпечення»

Розроблено викладачем

_____/Сайко Т.С./

Розглянуто та схвалено

на засіданні ЦК ПІ

Протокол №__ від __. _____. 2023 р.

Голова ЦК ПІ

_____/Ланська С.С./

«__»_____ 2023 р.

Лекція № 1

з навчальної дисципліни «Основи програмування та алгоритмічні мови»

Тема Визначення, призначення, синтаксис вказівників. Операції над вказівниками

Мета заняття Ознайомити студентів із поняттям вказівника, призначенням вказівників, принципами визначення вказівників та дій над вказівниками

Матеріально-технічне забезпечення та дидактичні засоби, ТЗН:
конспект лекції, опорний конспект

Час – 2 години.

План проведення лекції

Структура лекції	Відведений час	Методичні вказівки
1 Організаційна частина	5 хв.	Включає в себе: привітання, яке має на меті привернути увагу, забезпечити контакт лектора з аудиторією; визначення присутності студентів на занятті
2 Актуалізація опорних знань, перевірка вивченого матеріалу та мотивація навчальної діяльності студентів	10 хв.	Включає в себе: вступне зауваження, мотивацію актуальності теми, перевірку попереднього матеріалу, формулювання мети лекції, огляд головних питань теми
3 Основна частина (викладення навчальних питань лекції)	60 хв.	Головне місце приділяється викладенню основного змісту матеріалу теми, його аналізу, узагальненню висунутих положень. Для успіху лекції важливе значення має поділ матеріалу на розділи, основні питання
4 Заключна частина Домашнє завдання: (відповідно до робочої програми)	5 хв.	Включає в себе: узагальнення, теоретичні і фактичні висновки, закріплення вивченого на лекції матеріалу, виставлення оцінок за роботу на занятті

Література

- 1 Шилдт, Г. С++: базовий курс, 3-е видання.: Пер. з англ. [Текст] / Г. Шилдт. – М.: Видавничий дім "Вільямс", 2010. – 624 с.: іл.
- 2 Страуструп, Б. Мова програмування С/С++ [Текст] / Б. Страуструп. – М.: Біном, 2006. – 501 с.: іл.
- 3 Паппас, К.Х. Відлагодження в С/С++. Керівництво для розробників [Текст] / К.Х. Паппас, У.Х. Мюррей III. – М.: "Біном", 2009. – 452 с.

Навчальні матеріали лекції

1 Визначення, призначення, синтаксис вказівників

Вказівник – змінна, що містить адресу об'єкту. Вказівник не несе ніякої інформації про сам об'єкт, а містить відомості про те, де розміщений об'єкт. Вказівники широко використовуються при програмуванні на мові C/C++. Програми з вказівниками — короткі і дуже ефективні.

Вказівники застосовуються:

- для доступу до елементів оперативної пам'яті і створення нових об'єктів в ході виконання програми;
- для доступу до складних елементів даних;
- для виконання різних операцій з елементами масиву;
- і так далі.

Поняття «вказівник» можна пояснити, використовуючи спрощену схему організації пам'яті ЕОМ (див. рисунок 1.1). Як правило, пам'ять ЕОМ можна представити у вигляді послідовності пронумерованих однобайтових комірок, з якими можна працювати окремо або блоками. Вказівник – це теж змінна, яка розміщується в пам'яті. Зазвичай вказівники займають 2 або 4 байти – залежно від моделі пам'яті. На рисунку 1.1 – змінна "с" має тип `char`, вказівник "р" містить адресу змінної "с". Взаємозв'язок змінних "р" і "с" показана стрілкою.

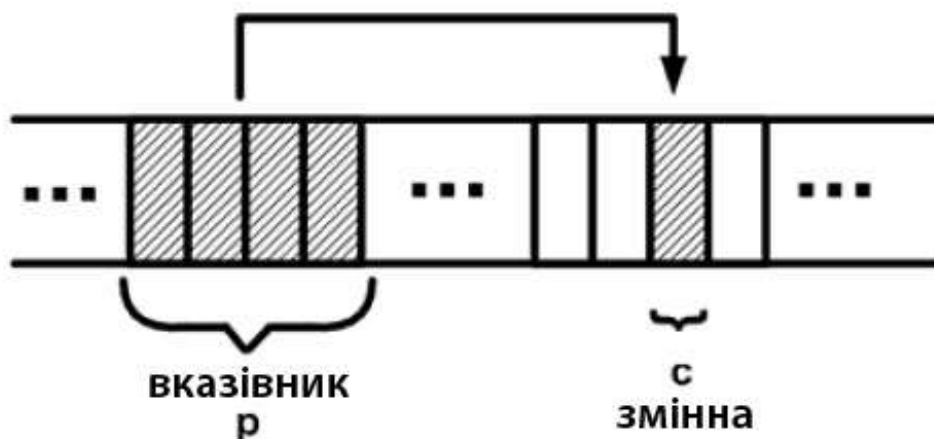


Рисунок 1.1 – Схематичне розміщення взаємодії вказівників та змінних

Вказівники діляться на дві категорії: вказівники на об'єкти і вказівники на функції. Розглянемо вказівники на об'єкти, які зберігають адресу області пам'яті, що містить дані певного типу.

Вказівник, як і будь-яка змінна, повинен бути оголошений.

Загальна форма оголошення вказівника

```
тип *ім'я_вказівника;
```

де "тип" може бути будь-яким, окрім посилання; "ім'я_вказівника" – формується як будь-який ідентифікатор.

Приклад оголошення вказівника:

```
int *p;
char *c;
float *f, *ff;
```

Вказівник може бути константою або змінною, а також вказувати на константу або змінну.

Для роботи з вказівниками в C/C++ визначено дві операції:

1) Операція "&" (отримання адреси) – дозволяє визначити адресу об'єкту.

Операція "&" видає адресу об'єкту, так що запис

```
p=&c;
```

привласнює вказівнику "p" адресу комірки пам'яті, в якій розміщується значення змінної "c".

2) Операція "*" (розіменування) – дозволяє згенерувати значення об'єкту за його адресою.

Операція "*", застосована до вказівника, видає значення об'єкту, на який даний вказівник посилається, так що запис

```
cout<<*p;
```

призведе до виведення на екран значення за вказівником p (беручи до уваги попередній приклад, то буде виведено значення змінної "c").

Для ініціалізації вказівників існують наступні способи:

1 За допомогою операції отримання адреси.

```
int a=5;
int *p=&a; //або int p(&a);
```

2 За допомогою іншого вказівника, який проініціалізований.

```
int *r=p;
```

3 Адреса привласнюється в явному вигляді.

```
char *cp=(char*) 0xB8000000;
```

де 0xB8000000 – шістнадцятирична константа, (char*) – операція приведення типу.

4 Привласнення порожнього значення:

```
int*N=NULL;
int *R=0;
```

Приклад програми для роботи з вказівниками на змінні.

```
#include <stdio.h>
#include <conio.h>
int main()
{
    int a;
    int *b;
    a=134;
    b=&a;
    printf("\n Значення змінної a = %i", a);
    printf("\n Адреса змінної a = %p", &a);
    printf("\n Дані за адресою вказівника b= %i", *b);
    printf("\n Значення вказівника b = %p", b);
    printf("\n Адреса розташування вказівника b=%p", &b);
    getch();
    return 0;
}
```

Результат роботи програми:

Значення змінної a = 134.

Адреса змінної a = 0012FED4.

Дані за адресою вказівника b = 134.

Значення вказівника b = 0012FED4.

Адреса розташування вказівника b рівна 0012FEC8.

2 Одномірні (лінійні) масиви і вказівники

При визначенні масиву йому виділяється пам'ять. Після цього ім'я масиву сприймається як константний вказівник того типу, до якого відносяться елементи масиву. Виключенням є використання операцій:

- `sizeof (ім'я_масиву)`, яка в якості результуючого значення повертає розмір всього масиву, визначене ім'ям масиву;
- `&ім'я_масиву`, яка в якості результуючого значення повертає адресу нульового елемента масиву, визначеним іменем масиву.

Приклади:

```
int a[100];
int k=sizeof(a); // результатом буде 4*100=400 (байтів).
int n=sizeof(a)/sizeof(a[0]); // кількість елементів масиву
```

Результатом операції `&` є адреса нульового елемента масиву:

```
ім'я_масиву=&ім'я_масиву[0]=&ім'я_масиву
```

Ім'я масиву є вказівником-константою, значенням якого служить адреса нульового елемента масиву, отже, до нього застосовні всі правила адресної арифметики, пов'язаної з вказівниками. Оскільки ім'я масиву є константним вказівником, то його неможливо змінити, отже, запис `*(a++)` буде помилковою, а `*(a+1)` – ні.

Запис `ім'я_масиву[індекс]` це вираз з двома операндами: ім'я масиву і індекс. `ім'я_масиву` – це вказівник константа, а індекс визначає зсув від початку масиву. Використовуючи вказівники, звернення по індексу можна записати таким чином:

```
*(ім'я_масиву+індекс);
```

Графічне представлення масиву в пам'яті ЕОМ представлене на рисунку 2.1, де `data` – адреса початку масиву; `sizeof(data)` – розмір масиву `data` в байтах; `sizeof(float)` – розмір пам'яті під один елемент масиву в байтах; `p1` і `p2` – вказівники для роботи з масивом.

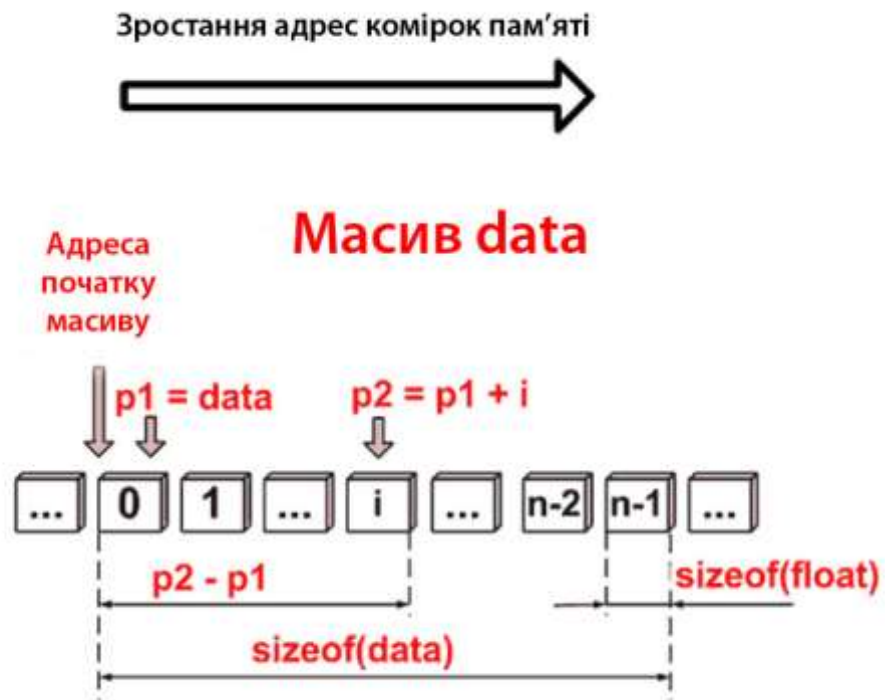


Рисунок 2.1 – Представлення масиву у пам'яті

Приклад звертання до елементів масиву за допомогою вказівників:

```
data[0] == *p1;
data[1] == *(p1 + 1);
data[2] == *(p1 + 2);
data[i] == *(p1 + i);
data[n-1] == *(p1 + n - 1).
```

Операції порівняння вказівників $p1$ та $p2$, які посилаються на масив:

- $p1 == p2$ – перевірка на рівність;
- $p1 != p2$ – перевірка на нерівність;
- $p1 < p2$ – порівняння вказівників на менше;
- $p1 <= p2$ – порівняння вказівників на менше або рівно;
- $p1 > p2$ – порівняння вказівників на більше;
- $p1 >= p2$ – порівняння вказівників на більше або рівно;
- $p2 - p1$ – обчислення кількості елементів між $p2$ і $p1$;
- $p1 + i$ – нарощування вказівника на i елементів;
- $p1 - i$ – зменшення вказівника на i елементів.

При роботі з вказівниками і масивами слід уважно стежити за тим, щоб адреси, що зберігаються в вказівниках, не виходили за рамки адрес масивів.

Приклад. Дано масив M(11). Написати програму, яка буде шукати мінімум з діапазону [-5;5].

```
#include <iostream.h>
#include <conio.h>
int main()
{float m[11];
float *pm=m;
float min=-6;
int i;
clrscr();
//введення елементів масиву
for(int i=0;i<11;i++)
{
    cout<<endl<<"Введіть M["<<i+1<<"]:";
    cin>>*(pm+i);
}
//виведення масиву на екран
clrscr();
cout<<endl<<"Введений масив:"<<endl;
for(int i=0;i<11;i++)
{
    cout<<*(pm+i)<<"    ";
}
//пошук хоча б одного мінімального елемента з діапазону
for(int i=0;i<11;i++)
{
    if (*(pm+i)>=-5 && *(pm+i)<=5)
    {
        min=*(pm+i);
    }
}
//пошук мінімального елемента з діапазону
for(int i=0;i<11;i++)
{
    if (*(pm+i)>=-5 && *(pm+i)<=5 && min>*(pm+i))
    {
        min=*(pm+i);
    }
}
if (min!=-6)
{
    cout<<endl<<"Мінімальний елемент з діапазону [-5;5]="<<min;
}
else
    cout<<endl<<"Елемент з діапазону [-5;5] - відсутній";
getch();
return 0;
}
```

3 Операції з вказівниками

З вказівниками можна виконувати наступні операції:

1 Розіменування (*).

2 Привласнення.

3 Арифметичні операції (складання з константою, віднімання, інкремент ++, декремент --).

4 Порівняння.

5 Приведення типів.

1 Операція розіменування (*) призначена для набуття значення змінної або константи, адреса якої зберігається у вказівнику. Якщо вказівник указує на змінну, то це значення можна змінювати, також використовуючи операцію розіменування (*).

Приклади:

```
int a, b; //змінні типу int
int *pb=&b; //вказівник на змінну b
*pb=10; //привласнили значення змінної b, на яку указує вказівник
a=*pb; //привласнили значення змінної a
```

Привласнювати значення вказівника-константам заборонено.

2 Приведення типів.

На одну і ту ж область пам'яті можуть посилатися вказівники різного типу. Якщо застосувати до них операцію розіменування (*), то вийдуть різні результати.

Наприклад,

```
int a=123;
int *pi=&a;
char *pc=(char*) &a;
float *pf=(float*) &a;
printf("\n%x\t%i", pi, *pi);
printf("\n%x\t%c", pc, *pc);
printf("\n%x\t%f", pf, *pf);
```

При виконанні цієї програми вийдуть наступні результати:

```
66fd9c 123
66fd9c {
66fd9c 0.000000
```

Тобто адреса у трьох вказівників одна і та ж, але при розіменюванні виходять різні значення залежно від типу вказівника.

У прикладі при ініціалізації вказівника була використана операція приведення типів – (char*) та (float*). При використанні у виразі вказівників різних типів, явне перетворення потрібне для всіх типів, окрім void*. Вказівник може неявно перетворюватися в значення типу bool, при цьому ненульовий вказівник перетвориться в true, а нульовий в false.

З Арифметичних операції застосовні тільки до вказівників одного типу.

- інкремент збільшує значення вказівника на величину sizeof(тип);

Наприклад,

```
char *pc;
int *pi;
float *pf;
. . . . .
pc++; //значення збільшиться на 1
pi++; //значення збільшиться на 4
pf++; //значення збільшиться на 4
```

- декремент зменшує значення вказівника на величину sizeof(тип);
- різниця двох вказівників – це різниця їх значень (адрес пам'яті об'єктів), що ділиться на розмір типу в байтах;

Наприклад:

```
int a=123, b=456, c=789;
int *pi1=&a;
int *pi2=&b;
int *pi3=&c;
printf("\n%x",pi1-pi2);
printf("\n%x",pi1-pi3);
```

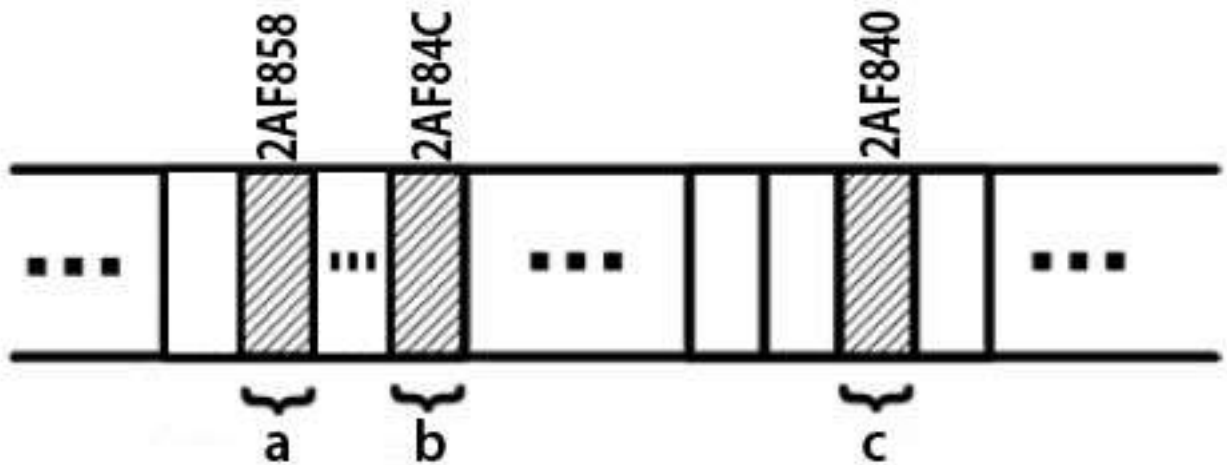


Рисунок 3.1 – Представлення змінних для підрахування різниці двох вказівників

Результат:

3 $//2AF858_{16}-2AF84C_{16}=2816088_{10}-2816076_{10}=12 / 4 (\text{sizeof(int)}) = 3$
 6 $//2AF858_{16}-2AF840_{16}=2816088_{10}-2816064_{10}=24 / 4 (\text{sizeof(int)}) = 6$

Складання та знаходження добутку двох вказівників не допускається.

Можна підсумовувати (знаходити добуток) вказівник і константу:

```
pi3=pi3+6;
pi2=pi2+3;
printf("\n%x\t%d",pi1,*pi1);
printf("\n%x\t%d",pi2,*pi2);
printf("\n%x\t%d",pi3,*pi3);
```

Результат виконання програми:

```
2af858 123
2af858 123
2af858 123
```

Контрольні питання

- 1 Дайте визначення поняття "вказівник".
- 2 Яку інформацію зберігає вказівник?
- 3 Для чого найчастіше використовується вказівники?
- 4 На які категорії (групи) діляться вказівники?
- 5 Наведіть загальну форму оголошення вказівника.
- 6 Для чого призначення операція "&" по відношенню до вказівника?

- 7 Для чого призначення операція "*" по відношенню до вказівника?
- 8 Які існують способи ініціалізації вказівників?
- 9 Яку операцію необхідно використовувати для отримання значення об'єкта на який посилається вказівник?
- 10 Яку операцію необхідно використовувати для отримання адреси об'єкту для ініціалізації вказівника?
- 11 Що являє собою ім'я масиву у контексті вказівників?
- 12 Як звернутися до елементу масиву з використанням вказівників?
- 13 Які операції можливо виконувати над вказівниками?
- 14 Для чого призначена операція розіменування вказівника?
- 15 При використанні операції явного приведення типів при визначенні адреси змінної одного типу чи буде результат розіменування вказівника однаковий?
- 16 На яке значення збільшується значення вказівника при використанні операції інкремент?
- 17 На яке значення зменшується значення вказівника при використанні операції декремент?
- 18 Що являє собою різниця двох вказівників на один масив?
- 19 Чи допускається складання вказівників?
- 20 Чи допускається знаходження добутку вказівників?
- 21 Чи вірне твердження `ім'я_масиву=&ім'я_масиву[0]=&ім'я_масиву`?
- 22 Чи вірне твердження `ім'я_масиву=&ім'я_масиву[1]`?
- 23 Чи вірне твердження `*(a++)`, якщо `a` – ім'я масиву?
- 24 Чи вірне твердження `*(a+1)`, якщо `a` – ім'я масиву?
- 25 Який обсяг пам'яті займає змінна-вказівник?

Лекція № 2

з навчальної дисципліни «Основи програмування та алгоритмічні мови»

Тема Динамічний розподіл пам'яті. Динамічні змінні та масиви. Складання програми з обробки динамічних масивів

Мета заняття Ознайомити студентів із поняттям динамічної пам'яті у C/C++, принципами її розподілу та схемами використання операторів new та delete

Матеріально-технічне забезпечення та дидактичні засоби, ТЗН:

конспект лекції, опорний конспект

Час – 2 години.

План проведення лекції

Структура лекції	Відведений час	Методичні вказівки
1 Організаційна частина	5 хв.	Включає в себе: привітання, яке має на меті привернути увагу, забезпечити контакт лектора з аудиторією; визначення присутності студентів на занятті
2 Актуалізація опорних знань, перевірка вивченого матеріалу та мотивація навчальної діяльності студентів	10 хв.	Включає в себе: вступне зауваження, мотивацію актуальності теми, перевірку попереднього матеріалу, формулювання мети лекції, огляд головних питань теми
3 Основна частина (викладення навчальних питань лекції)	60 хв.	Головне місце приділяється викладенню основного змісту матеріалу теми, його аналізу, узагальненню висунутих положень. Для успіху лекції важливе значення має поділ матеріалу на розділи, основні питання
4 Заключна частина Домашнє завдання: (відповідно до робочої програми)	5 хв.	Включає в себе: узагальнення, теоретичні і фактичні висновки, закріплення вивченого на лекції матеріалу, виставлення оцінок за роботу на занятті

Література

- 1 Шилдт, Г. С++: базовий курс, 3-е видання.: Пер. з англ. [Текст] / Г. Шилдт. – М.: Видавничий дім "Вільямс", 2010. – 624 с.: іл.
- 2 Страуструп, Б. Мова програмування C/C++ [Текст] / Б. Страуструп. – М.: Біном, 2006. – 501 с.: іл.
- 3 Паппас, К.Х. Відлагодження в C/C++. Керівництво для розробників [Текст] / К.Х. Паппас, У.Х. Мюррей III. – М.: "Біном", 2009. – 452 с.

Навчальні матеріали лекції

1 Види пам'яті

До основних видів пам'яті належать наступні:

1 Статична пам'ять. Місце в статичній пам'яті резервується в момент запуску програми і звільняється тільки після того, як програма завершить свою роботу.

Туди потрапляють глобальні змінні (оголошені поза будь-якими блоками, зокрема, до початку функції `main`) і статичні змінні (перед типом яких вказується ключове слово `static`, при цьому змінна може знаходитися де завгодно, в тому числі і в тілі якоїсь функції, але поводити себе статична змінна всередині функції буде специфічно: зберігати своє значення між повторними викликами функції). Різниця між статичною та глобальною змінною проявляється, коли програма складається з декількох файлів: глобальні змінні доступні в суміжних файлах, а статичні – тільки в тому файлі, де були оголошені.

У статичній пам'яті не рекомендується тримати великі за обсягами займаної пам'яті об'єкти (наприклад, масиви).

2 Локальна пам'ять (стек). Виділяється при вході в блок програми, звільняється – при виході з блоку. Туди потрапляють змінні оголошені всередині блоків (зокрема, в тілі функцій).

3 Динамічна пам'ять. Виділяється і звільняється – за допомогою спеціальних інструкцій (тобто з ініціативи програміста). Це дозволяє по ходу роботи програми контролювати і коригувати обсяг використовуваної пам'яті і створювати програми здатні обробляти великі обсяги даних, обходячи обмеженість фізичної пам'яті машини.

Доступ до динамічної пам'яті можливий тільки через покажчики, тобто не можна створювати у ній змінні, але можна виділяти її фрагменти і пов'язувати з деяким покажчиком.

Динамічні змінні створюються за допомогою спеціальних функцій і операцій. Вони існують або до кінця роботи програми, або до тих пір, пока не будуть знищені за допомогою спеціальних функцій або операцій.

2 Оператори для роботи з динамічною пам'яттю

Для створення динамічних змінних використовують оператор `new`, визначений у C++.

Загальний вигляд використання оператора `new`:

тип_даних *ім'я_вказівника = new тип_даних[(ініціалізатор)];

де *ініціалізатор* – вираз у круглих дужках.

Оператор `new` дозволяє виділити і зробити доступною область динамічної пам'яті, яка відповідає заданому типу даних. Якщо заданий ініціалізатор, то в цю область буде занесено значення, вказане в ініціалізаторі.

Для звільнення динамічної пам'яті використовується оператор `delete`, визначений у C++.

Загальний вигляд використання оператора `delete`:

`delete` вказівник;

Наприклад. Написати програму, яка виділяє динамічну пам'ять під змінну цілого типу, проініціалізувати динамічну змінну та вивести задане значення на екран.

```
#include <iostream.h>
```



```

int main()
{
    int *p= new int (10);
    cout << "Динамічна змінна *p="<<*p;
    *p = 20;
    cout << "Динамічна змінна *p після зміни="<<*p;

    delete p;
    return 0;
}

```

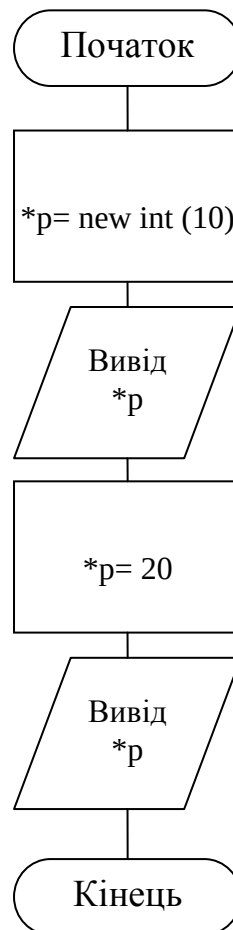


Рисунок 2.1 – Блок-схема програми

Результат роботи програми:

Динамічна змінна *p=10
 Динамічна змінна *p після зміни=20

3 Функції мови C для роботи з динамічною пам'яттю

У мові C визначені бібліотечні функції для роботи з динамічною пам'яттю, вони знаходяться у бібліотеці <stdlib.h>:

1 void *malloc (unsigned s) – повертає вказівник на початок області динамічної пам'яті довжиною s байт, при невдалому завершенні повертає NULL;

2 void *calloc (unsigned n, unsigned m) – повертає вказівник на початок області динамічної пам'яті для розміщення n елементів довжиною m байт кожен, при невдалому завершенні повертає NULL;

3 void *realloc (void *p, unsigned s) – змінює розмір блоку раніше виділеної динамічної пам'яті до розміру s байт, p – адреса початку змінюваного блоку, при невдалому завершенні повертає NULL;

4 void *free (void *p) – звільняє раніше виділену ділянку динамічної пам'яті, p – адреса початку ділянки.

Наприклад. Написати програму, яка виділяє динамічну пам'ять під змінну цілого типу, проініціалізувати динамічну змінну та вивести задане значення на екран.

```
#include <iostream.h>
#include <stdlib.h>
int main()
{
    int *p = malloc(sizeof(int));
    *p = 5;
    cout << "Динамічна змінна *p="<<*p;
    *p = 15;
    cout << "Динамічна змінна *p після зміни="<<*p;
    free(p);
    return 0;
}
```

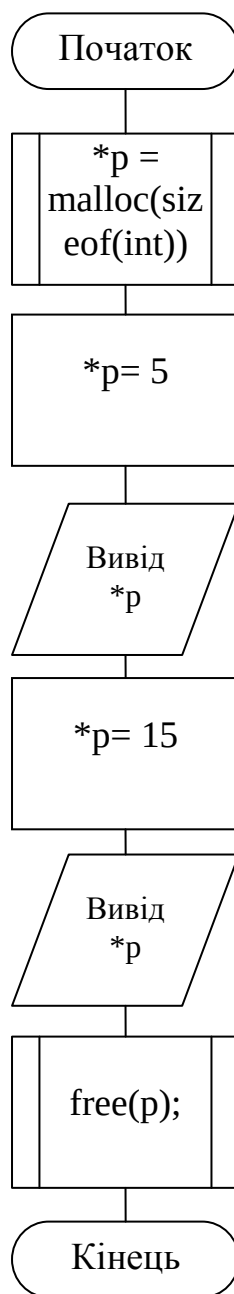


Рисунок 2.2 – Блок-схема програми

Результат роботи програми:

Динамічна змінна `*p=5`

Динамічна змінна `*p` після зміни=15

Контрольні питання

- 1 Назвіть види пам'яті, які використовуються у мові C/C++.
- 2 Охарактеризуйте статичну пам'ять.
- 3 Назвіть призначення оператора new.
- 4 Охарактеризуйте динамічну пам'ять.
- 5 Назвіть призначення оператора delete.
- 6 Охарактеризуйте локальну пам'ять.
- 7 Наведіть загальний вигляд оператора new.
- 8 Наведіть загальний вигляд оператора delete.

Лекція № 3-4

з навчальної дисципліни «Основи програмування та алгоритмічні мови»

Тема Динамічні матриці.

Мета заняття Пояснити принципи виділення динамічної пам'яті під матриці даних. Розглянути приклади формування та обробки динамічних лінійних масивів та матриць.

Матеріально-технічне забезпечення та дидактичні засоби, ТЗН:
конспект лекції, опорний конспект

Час – 4 години

План проведення лекції

Структура лекції	Відведений час	Методичні вказівки
1 Організаційна частина	5 хв.	Включає в себе: привітання, яке має на меті привернути увагу, забезпечити контакт лектора з аудиторією; визначення присутності студентів на занятті
2 Актуалізація опорних знань, перевірка вивченого матеріалу та мотивація навчальної діяльності студентів	10 хв.	Включає в себе: вступне зауваження, мотивацію актуальності теми, перевірку попереднього матеріалу, формулювання мети лекції, огляд головних питань теми
3 Основна частина (викладення навчальних питань лекції)	60 хв.	Головне місце приділяється викладенню основного змісту матеріалу теми, його аналізу, узагальненню висунутих положень. Для успіху лекції важливе значення має поділ матеріалу на розділи, основні питання
4 Заключна частина Домашнє завдання: (відповідно до робочої програми)	5 хв.	Включає в себе: узагальнення, теоретичні і фактичні висновки, закріплення вивченого на лекції матеріалу, виставлення оцінок за роботу на занятті

Література

- 1 Шилдт, Г. С++: базовий курс, 3-е видання.: Пер. з англ. [Текст] / Г. Шилдт. – М.: Видавничий дім "Вільямс", 2010. – 624 с.: іл.
- 2 Страуструп, Б. Мова програмування С/С++ [Текст] / Б. Страуструп. – М.: Бі-

ном, 2006. – 501 с.: іл.

З Паппас, К.Х. Відлагодження в C/C++. Керівництво для розробників [Текст] / К.Х. Паппас, У.Х. Мюррей III. – М.: "Біном", 2009. – 452 с.

Навчальні матеріали лекції

Для формування динамічної матриці використовується поняття *непрямої адресації*. Пояснимо це.

Іноді вказівник може вказувати на інший вказівник, який, в свою чергу, містить адресу звичайної змінної. Така схема називається *непрямою адресацією* або *вказівником на вказівник*. Концепція непрямої адресації проілюстрована на рисунку 2.1.



Рисунок 2.1. Пряма та непряма адресація

Змінну, яка є вказівником на вказівник, необхідно описати відповідним чином. Для цього достатньо перед її ім'ям поставити додатковий символ - зірочку (*). Наприклад, наступний опис повідомляє компілятор про те, що `balance` – це вказівник на вказівник на значення типу `int`.

```
int **balance;
```

Необхідно пам'ятати, що змінна `balance` - це не вказівник на цілочисельне значення, а вказівник на вказівник на `int`-значення. Для того, щоб отримати значення, що адресується вказівником на вказівник, необхідно двічі застосувати операцію розіменування `**`.

Загальна форма виділення пам'яті під багатовимірний (n-мірний) масив наступна:

```
Тип **Ідентифікатор=(Тип **) new Тип [розмір_1]...[розмір-n];
```

Де Тип – це тип елементу масиву, Ідентифікатор – ім'я масиву, `[розмір_1]...[розмір-n]` – це розмір (кількість елементів) кожної розмірності.

Наприклад:

```
int **m=(int **) new int [5][10][7]; /* формування трьох-  
вимірного масиву */
```

При цьому треба враховувати, що тільки сама ліва розмірність може бути задана змінною.

Тому для формування динамічної матриці, в якій кількість рядків та кількість стовбців задається змінними будемо використовувати більш універсальний (і безпечний) спосіб.

Динамічну матрицю можна розглядати як лінійний масив рядків, але при цьому кожен рядок є лінійним масивом розміром в кількість стовбців. Наприклад, нам необхідно сформувати динамічну матрицю дійсних чисел, що містить `nr` рядків та `ns` стовбців (`nr` та `ns` вводить користувач). Спочатку опишемо вказівник на вказівник на тип `float` та, за допомогою `new`, виділимо пам'ять під `nstr` вказівників на тип `float`

```
float **a= new float * [nr];/*створюється масив неіні-  
ціалізованих вказівників на тип float та адреса цього масиву зано-  
ситься до вказівника a */
```

Тепер в циклі будемо виділяти пам'ять під кожен рядок матриці:

```
for (inti=0; i<nr; i++)  
    a[i]=new int [ns]; /* кожному елементу масиву вказівників  
привласнюється адреса початку рядка матриці (усього ns елементів у  
рядку) */
```

Схема розподілу пам'яті при такому способі формування динамічної матриці наведена на рисунку 2.2.

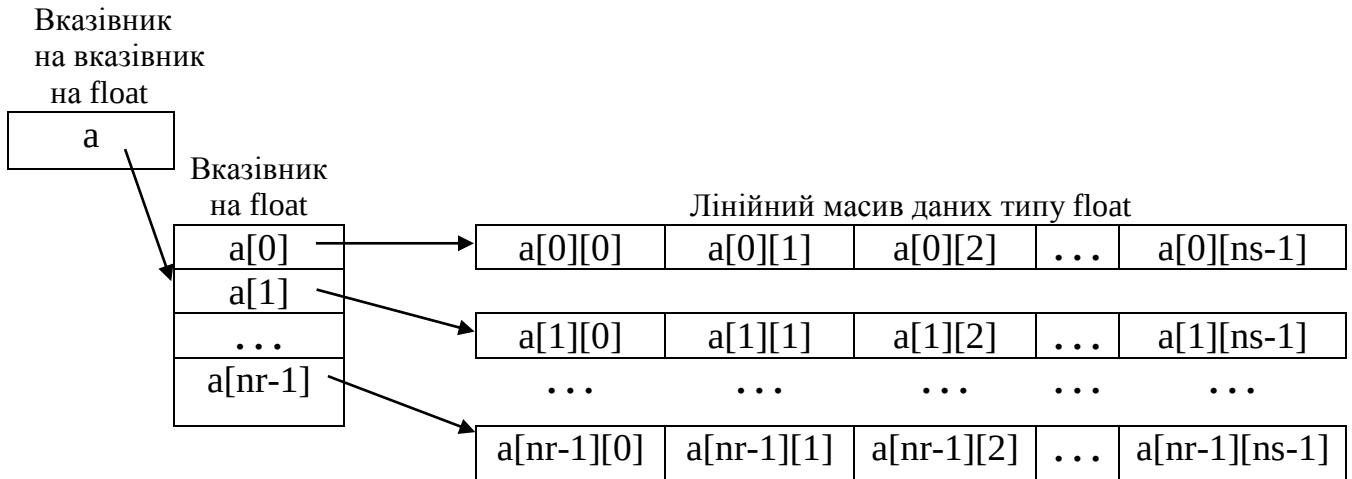


Рисунок 2.2. Розподіл пам'яті для динамічної матриці.

Розглянемо приклад програми обробки двовимірного динамічного масиву.

Постановка задачі:

Розмірність матриці (кількість рядків та стовпчиків) ввести з клавіатури. Створити відповідну динамічну матрицю цілих чисел. Знайти MAX та обчислити суму від'ємних непарних елементів у кожному з рядків.

На лістингу 2.1 приведено програму обробки динамічної матриці.

Лістинг 2.1.

```
#include <iostream.h>
#include <conio.h>
int main (void)
{
    int str, stb;
    cout<<endl<<"Введіть кількість строк ";
    cin>>str;
    cout<<endl<<"Введіть кількість стовбцов ";
    cin>>stb;
    int max; // «поточний» максимум для кожного рядка
    int sum; // змінна для обчислення суми
    int i,j;
```



```

// формування динамічної матриці
int **m=new int*[str];
for (i=0;i<str;i++) m[i]=new int[stb];
// введення елементів матриці
for (i=0;i<str;i++)
    for (j=0; j<stb; j++)
    {
        cout<<endl<<"Введите элемент матрицы m["<<i+1<<","<<j+1<<"]:";
        cin>>*(*(m+i)+j);
    }
// обробка матриці
for (i=0; i<str; i++)
{
    k=0; max=m[i][0]; sum=0;
    for (j=1; j<stb; j++)
    {
        if (m[i][j]>max) max=m[i][j];
        if ( (m[i][j]<0) && (m[i][j]%2!=0)) sum=sum+ m[i][j];
    }
    cout<<endl<<"В"<<i+1<<"строке max= "<<max;
    if (sum==0)
        cout<<endl<<"В"<<i+1<<"строке отрицательных нечётных нет";
    else
        cout<<endl<<"sum="<<sum;
}
delete []m;
return 0;
}

```

Питання та завдання до контролю знань студентів

Питання для узагальнення та перевірки засвоєного матеріалу на лекції

1 Яка загальна форма виділення пам'яті під багатомірний динамічний масив?

2 Знайдіть помилку в наступному фрагменті:

```

int n=5, m=10;
int **b=(int **) new int* [5][10];

```

Виправте цю помилку.

3 Опишіть спосіб виділення пам'яті під динамічну матрицю, в якій кількість рядків та кількість стовбців задаються змінними.

Завдання для самоперевірки засвоєного матеріалу на лекції

Написати мовою C++ програми обробки динамічних матриць:

1 Дано лінійний масив дійсних чисел $M(n)$. Розмір масиву n вводиться користувачем. Розмістити у масиві $M1$ тільки ті елементи вхідного масиву, які не перевищують суми усіх його елементів.

2 Дана матриця дійсних чисел $A(n, m)$ (n та m вводять користувач). Знайти кількість входжень: максимальних елементів; мінімальних елементів.

3 Знайти різницю між найбільшим та найменшим елементами головної діагоналі квадратної матриці $C(n,n)$ (n вводять користувач).

4 Дана матриця цілих чисел $B(n,n)$ (n вводять користувач). Знайти MAX та обчислити середнє арифметичне додатних елементів у кожному із стовпчиків.

Лекція № 5

з навчальної дисципліни «Основи програмування та алгоритмічні мови»

Тема Функція: призначення, визначення та прототипи. Параметри функції та способи їх передавання. Приклад складання програми з використанням функцій.

Мета заняття Пояснити правила визначення функції в C++ програмах, поняття формальних та фактичних параметрів, виклик функції, повернення функцією значення, опис прототипу функції.

Матеріально-технічне забезпечення та дидактичні засоби, ТЗН:
конспект лекції, опорний конспект

Час – 2 години

План проведення лекції

Структура лекції	Відведений час	Методичні вказівки
1 Організаційна частина	5 хв.	Включає в себе: привітання, яке має на меті привернути увагу, забезпечити контакт лектора з аудиторією; визначення присутності студентів на занятті
2 Актуалізація опорних знань, перевірка вивченого матеріалу та мотивація навчальної діяльності студентів	10 хв.	Включає в себе: вступне зауваження, мотивацію актуальності теми, перевірку попереднього матеріалу, формулювання мети лекції, огляд головних питань теми
3 Основна частина (викладення навчальних питань лекції)	60 хв.	Головне місце приділяється викладенню основного змісту матеріалу теми, його аналізу, узагальненню висунутих положень. Для успіху лекції важливе значення має поділ матеріалу на розділи, основні питання
4 Заключна частина Домашнє завдання: (відповідно до робочої програми)	5 хв.	Включає в себе: узагальнення, теоретичні і фактичні висновки, закріплення вивченого на лекції матеріалу, виставлення оцінок за роботу на занятті

Література

- 1 Шилдт, Г. C++: базовий курс, 3-е видання.: Пер. з англ. [Текст] / Г. Шилдт. – М.: Видавничий дім "Вільямс", 2010. – 624 с.: іл.
- 2 Страуструп, Б. Мова програмування C/C++ [Текст] / Б. Страуструп. – М.: Бі-

ном, 2006. – 501 с.: іл.

3 Паппас, К.Х. Відлагодження в C/C++. Керівництво для розробників [Текст] / К.Х. Паппас, У.Х. Мюррей III. – М.: "Біном", 2009. – 452 с.

Навчальні матеріали лекції

1 Визначення функції в C++ програмах

Із збільшенням об'єму програми стає неможливо утримувати в пам'яті всі деталі. Щоб зменшити складність програми, її розбивають на частини. У C++ задача може бути розділена на простіші під задачі за допомогою функцій. Розрізняють системні (у складі систем програмування) і власні функції.

Розбиття програм на функції дає наступні переваги:

- функцію можна викликати з різних місць програми, що дозволяє уникнути повторного програмування;
- одну і ту ж функцію можна використовувати в різних програмах;
- функції підвищують рівень модульності програми і полегшують її проектування;
- використання функцій полегшує читання і розуміння програми, прискорює пошук і виправлення помилок.

Функція – це іменована послідовність описів і операторів, що виконує закінчену дію, наприклад, пошук максимального значення серед двох чисел, пошук модулю числа тощо.

Функція, по-перше, є одним з похідних типів C++, а, по-друге, мінімальним виконуваним модулем програми.

Будь-яка функція повинна бути оголошена і визначена. Оголошення функції (прототип) задає ім'я функції, тип значення, яке повертається, і список параметрів, які передаються. Визначення функції містить, окрім прототипу (оголошення), тіло функції, яке є послідовністю описів і операторів.

Загальний вигляд визначення функції:

```
тип_значення ім'я_функції([список_формальних_параметрів])
{
    тіло_функції;
    return [вираз];
}
```

Приклад визначення функції знаходження суми двох чисел.

Функція з ім'ям `sum()`, в яку передаються 2 параметри – дійсні змінні `x`, `y`. Функція повертає дійсне значення */

```
float sum(float x, float y)
{
    float z; //оголошення дійсної змінної z
    // знаходження суми змінних x та y та запис результату у змінну z //
    z=x+y;
    return z; // повернення з функції значення змінної z
}
```

Тип значення, яке повертається, може бути будь-яким, окрім масиву.

Список формальних параметрів – це ті величини, які потрібно передати у функцію. Елементи списку розділяються комами. Для кожного параметра вказується тип і ім'я. У оголошенні імена можна не вказувати.

Тіло функції – це блок або складений оператор. У середині функції **не можна визначити іншу функцію**.

У тілі функції може бути оператор, який повертає значення функції в точку її виклику. Він може мати 2 форми:

- `return вираз;`
- `return.`

Перша форма використовується для повернення результату, тому "вираз" повинен мати той же тип, що і тип функції у визначенні.

Друга форма використовується, якщо функція не повертає значення, тобто має тип `void`.

Для того, щоб виконувалися оператори, записані в тілі функції, функцію необхідно викликати. При виклику указуються: ім'я функції і фактичні параметри, які будуть передані функції.

Фактичні параметри замінюють формальні параметри при виконанні операторів тіла функції. Фактичні і формальні параметри повинні співпадати по кількості і типу.

Оголошення функції повинне знаходитися в тексті раніше виклику функції, щоб компілятор міг здійснити перевірку правильності виклику. Якщо функція має тип не `void`, то її виклик може бути операндом виразу.

У лістингу 1.1 наведено приклад програми обчислення факторіалу введеного числа.

Лістинг 1.1.

```
#include <iostream.h>
#include <conio.h>
/* визначення функції з ім'ям factorial(), якій передається
одне ціле значення - змінна "i". Функція повертає дійсне значення.
*/
float factorial(int i)
{
    int j;
    float k;
    k=1;
    for (j=2;j<i+1;j++) k=k*j;
    // повернення в програму обчисленого значення
    return k;
}
// головна функція програми
int main()
{
    clrscr();
    int i;
    cout<<endl<<"Введіть ціле число: ";
    cin>>i;
    // виклик функції factorial() з фактичним параметром i
    cout<<endl<<i<<"!="<<factorial(i);
    getch();
    return 0;
}
```

Блок – схема алгоритму рішення задачі наведена на рисунку 1.1.

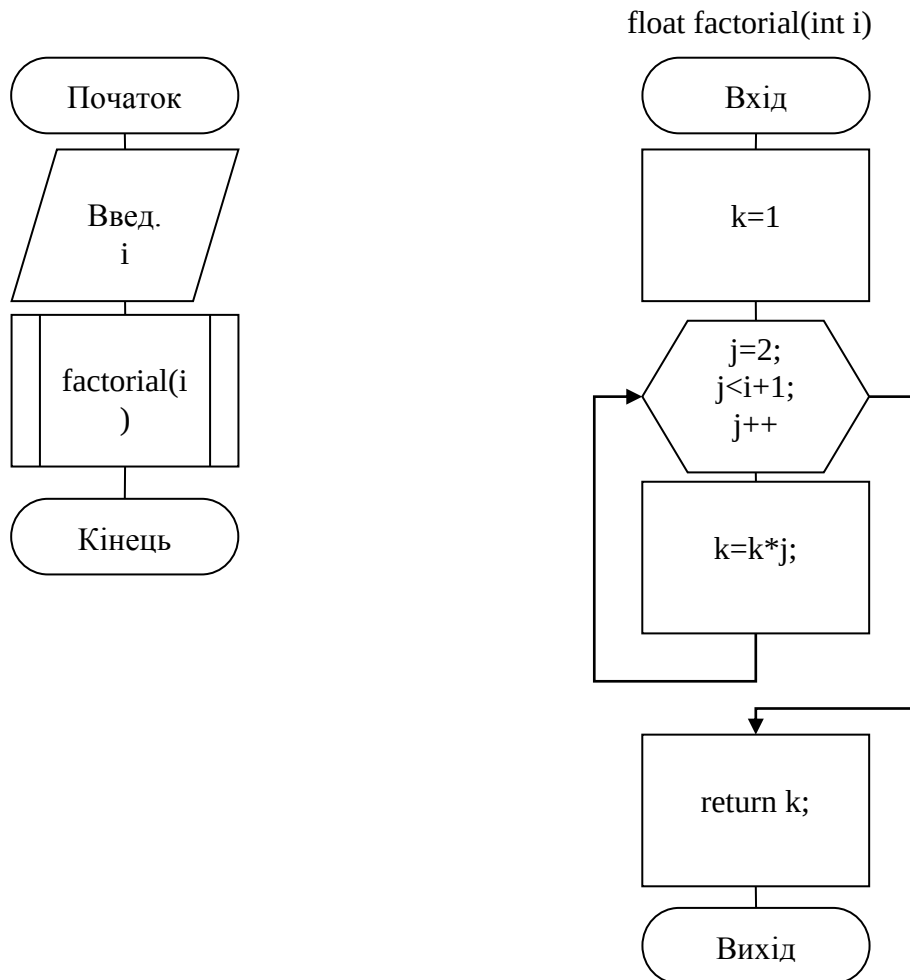


Рисунок 1.1. Блок – схема алгоритму пошуку факторіалу цілого числа.

Результат роботи програми:

Введіть ціле число: 7
7.000000!=5040.000000

2 Прототип функцій

У практиці програмування бувають випадки, коли тіло функції розташовується в програмі нижче за місцем, де виконується її виклик, або функція взагалі компілюється окремо. В цьому випадку до використання функції повинен бути вказаний прототип функції (або оголошення функції), що містить:

- тип функції;
- ім'я функції;

– інформацію про параметри.

Прототип необхідний для того, щоб компілятор зміг здійснити перевірку відповідності типів і кількості фактичних параметрів, які передаються, типам і кількості формальних параметрів. Оголошення функції має той же вигляд, що і визначення функції, проте тіло функції відсутнє, і імена формальних параметрів також можуть бути відсутніми.

Загальний вигляд визначення прототипу функції.

```
тип_значення ім'я_функції([список_формальних_параметрів]);
```

Приклад визначення прототипу (оголошення) функції модуля числа:

```
int abs(int i);
```

При програмуванні на мові C/C++ широко використовуються бібліотечні функції. Ці функції були заздалегідь розроблені і записані до складу системи програмування. Прототипи бібліотечних функцій знаходяться в спеціальних заголовних файлах з розширенням *.h, які необхідно підключати за допомогою директиви #include.

Не лістингу 2.1 наведено приклад програми генерації таблиці чисел 2^n , де n – числа від 1 до 10.

Лістинг 2.1.

```
#include <iostream.h>
#include <conio.h>

/* Опис прототипу функції power() з 2-ма цілими параметрами –
змінні base, index. Функція повертає ціле значення. */

int power(int base, int index);

// головна функція програми
int main()
{
    clrscr();
    int i;
    for (i=0; i <=10; i++ )
    {
        cout<<power(2,i)<<"    ";
    }
    getch();
}
```



```

return 0;
}
// Безпосередньо функція power()
int power(int base, int index)
{
    int i, p;
    p = 1;
    for (i = 0; i <= index; i++)
    {
        p = p*base;
    }
    return p;
}

```

Блок – схема алгоритму рішення задачі наведена на рисунку 2.1.

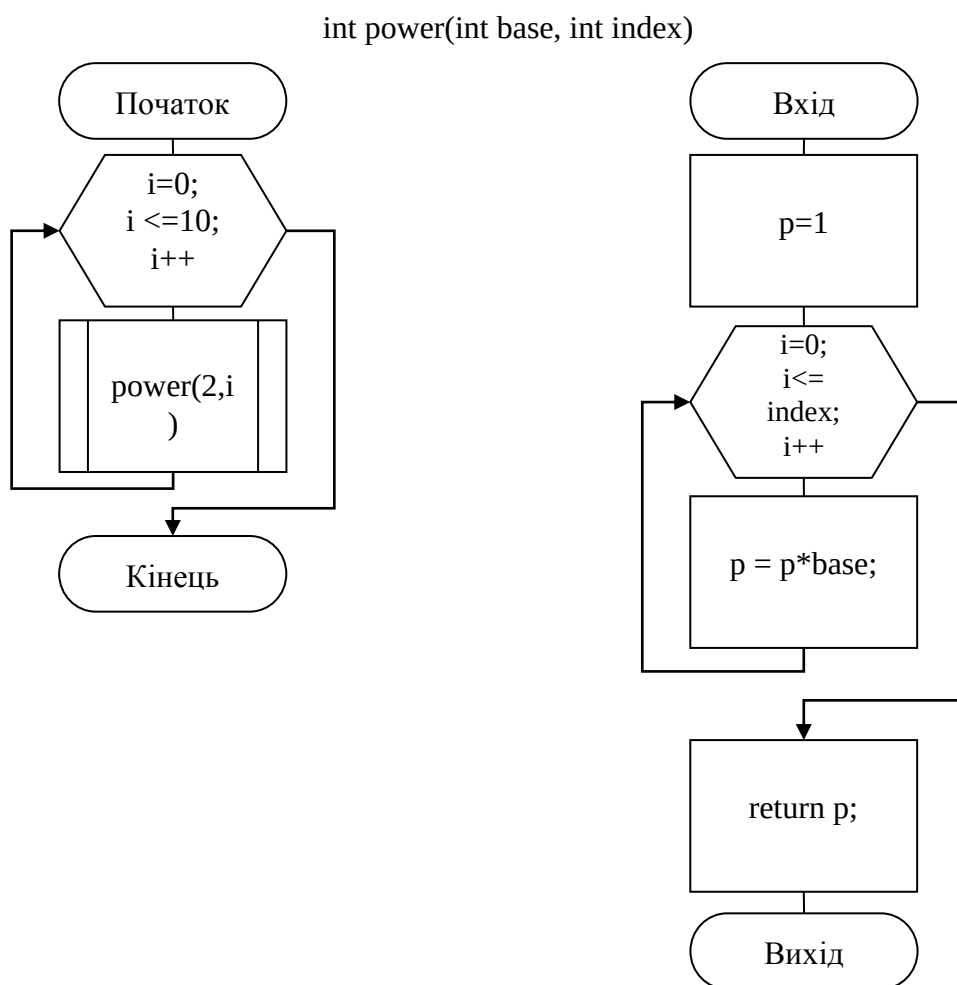


Рисунок 2.1. Блок – схема алгоритму обчислення цілого ступеня числа 2

Результат роботи програми:

2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048
--

З Виклик функцій. Повернення значення в точку виклику функції

Загальний вигляд виклику функції:

```
ім'я_функції([список_фактичних_параметрів]);
```

Фактичний параметр – це величина, яка привласнюється формальному параметру при виклику функції. Таким чином, формальний параметр – це змінна у функції, що викликається, а фактичний параметр – це конкретне значення, привласнене цій змінній. Фактичний параметр може бути константою, змінною або виразом. Фактичний параметр спочатку обчислюється, а потім його значення передається у функцію. Якщо у функцію потрібно передати декілька параметрів, то вони записуються через кому.

Функція може передати в точку виклику програми тільки одне значення, для цього використовується оператор `return`.

```
return [вираз];
```

Працює оператор так. Обчислюється значення виразу, вказаного після `return` і перетворюється до типу значення, що повертається функцією. Виконання функції завершується, а обчислене значення передається в функцію, яка її викликала. Будь-які оператори, наступні у функції за оператором `return`, ігноруються. Програма продовжує свою роботу з оператора наступного за оператором виклику даної функції.

Якщо функція має тип `void`, тобто не повертає в точку виклику ніякого значення, в операторі `return [вираз]` можна опустити, якщо повернення відбувається перед фігурною дужкою, що закривається, і у функції `main`.

Приклад використання `return` в середині функції.

```
void function()
{
...
return;
```

```
...
}
```

У прикладі оператор `return` завершує виконання функції `function()` і передає управління в функцію, яка здійснила виклик. Ніяке значення при цьому не передається.

Також функція може містити декілька операторів `return`, якщо це визначено потребами алгоритму. Наприклад, в наступній програмі функція `f` порівнює значення змінної з нулем. Якщо число додатне, то в програму передається значення рівне 1, якщо від'ємне, то -1, а якщо нульове, то 0.

```
#include <iostream>
int f(int a)
{
    if (a>0) return 1;
    else if (a<0) return -1;
    else return 0;
}
//функцію можна переписати так
//int f(int a)
//{ int r;
//if (a>0) r=1;
//else if (a<0) r=-1;
//else r=0;
//return(r);
//}
int main ()
{
    int b;
    cout<<"b=";
    cin>>b;
    if (f(b)==1) cout<<"positive value \n";
    else if (f(b)==-1) cout<<"negative value \n";
    else cout<<"zero value \n";
    system ("pause");
} //у функції main відсутній оператор return
```

Виклик функції залежить від того, повертає вона значення чи ні. Якщо функція повертає значення, то її виклик проводиться з математичних і логічних виразів.

У лістингу 3.1 наведено приклад функції визначення максимуму серед двох чисел.

Лістинг 3.1.

```
#include <iostream.h>
#include <conio.h>

int max(int a, int b)
{
    if (a>b)
        return a;
    else
        return b;
}

int main()
{
    clrscr();
    int x, y;
    cout<<endl<<"Введіть 2 числа: ";
    cin>>x>>y;
    cout<<endl<<"Максимум з них = "<<max(x,y);
    getch();
    return 0;
}
```

Блок – схема алгоритму рішення задачі наведена на рисунку 3.1.

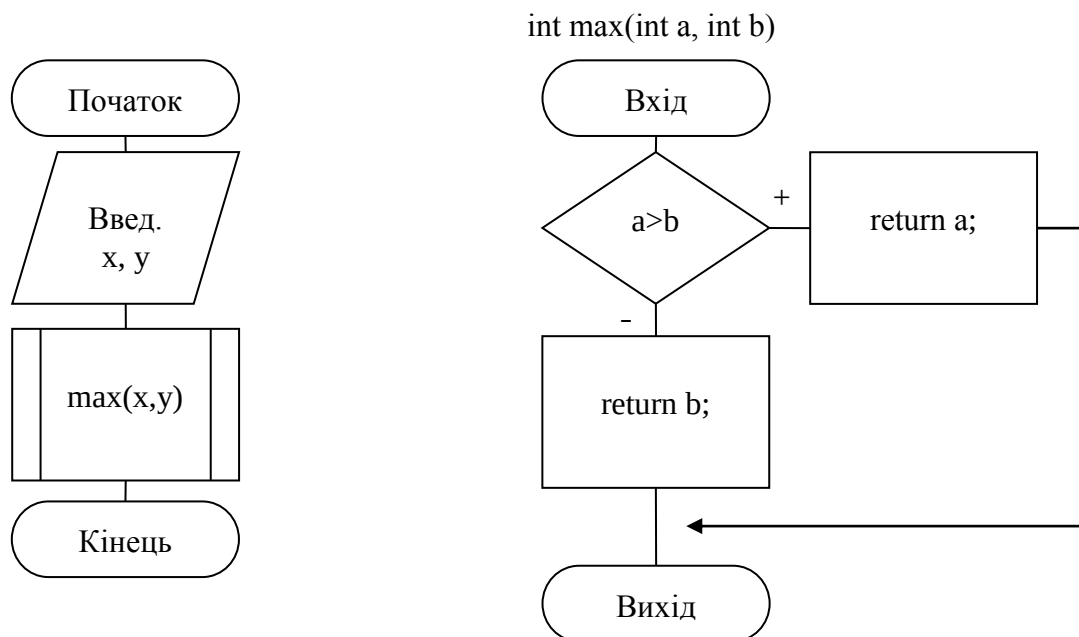


Рисунок 3.1. Блок – схема алгоритму пошуку максимуму двох чисел

Результат роботи програми:

Введіть 2 числа: 10 7

Максимум з них = 10

Функції можуть і не повертати значення, а просто виконувати деякі обчислення або вивід на екран. У такому разі вказується порожній тип даних `void`, і оператор `return` не використовується.

На лістингу 3.2 наведено приклад програми, в якій функція `star(int a)` не повертає значення в точку виклику, тому що призначення цієї функції - виведення на екран заданої користувачем кількості «зірочок».

Лістинг 3.2.

```
#include <iostream.h>
#include <conio.h>

void star (int a);

int main()
{
    clrscr();
    int k;
    cout<<endl<<"Скільки символів надрукувати?";
    cin>>k;
    star(k);
    getch();
    return 0;
}

void star (int a)
{
    int i;
    for(i=1; i<=a;i++)
    {
        cout<<"*";
    }
}
```

Блок – схема алгоритму рішення задачі наведена на рисунку 3.2.

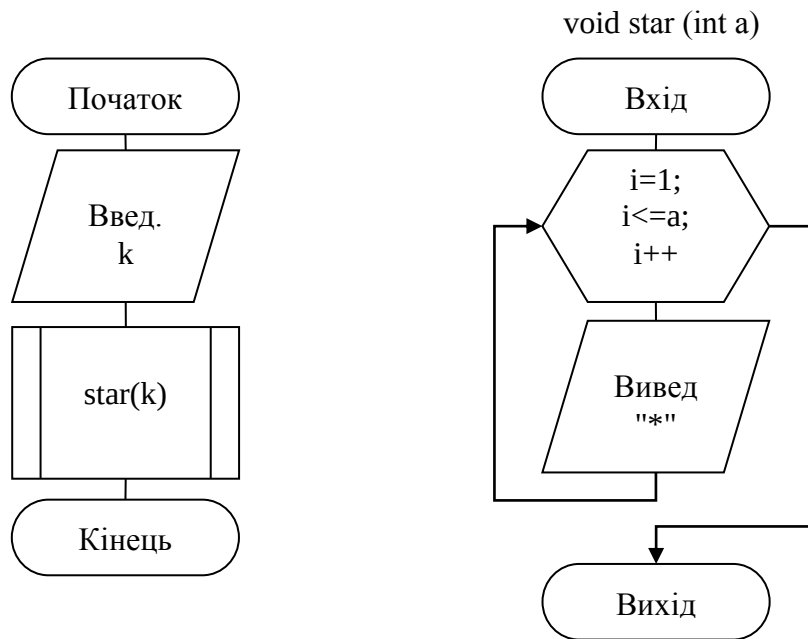


Рисунок 3.2. Блок – схема програми виведення заданої кількості символу «*»

Функції використовують пам'ять зі стека. Деяка область стека приділяється функції й залишається пов'язаною з нею до закінчення її роботи, по завершенню якої відведена їй пам'ять звільняється й може бути зайнята іншою функцією.

Всі величини, описані усередині функції, а також її параметри, є локальними. Областю їхньої дії є функція. При виклику функції, як і при вході в будь-який блок, у стеці виділяється пам'ять під локальні автоматичні змінні. При виході з функції відповідна ділянка стека звільнюється, тому значення локальних змінних між викликами однієї й тієї ж функції не зберігаються. Якщо цього потрібно уникнути, при оголошенні локальних змінних використовується модифікатор `static`:

```
#include <iostream.h>
void f(int a)
{
    int m=0;
    cout<<"n m p\n";
    while (a--)
    {
        static int n=0;
```

```

    int p=0;
    cout<<n++<< ' ' <<m++<< ' ' <<p++<< '\n';
}

int main()
{
    f(3); f(2);
    return 0;
}

```

Статична змінна *n* розміщується в сегменті даних й ініціалізується один раз при першому виконанні оператора, що містить її визначення. Автоматична змінна *m* ініціалізується при кожному вході у функцію. Автоматична змінна *p* ініціалізується при кожному вході в блок циклу. Програма виведе на екран:

n	m	p
0	0	0
1	1	0
2	2	0
n	m	p
3	0	0
4	1	0

Питання та завдання до контролю знань студентів

Питання для узагальнення та перевірки засвоєного матеріалу на лекції

- 1 Дайте визначення поняття "функція"?
- 2 Дайте визначення поняття "прототип (оголошення) функції"?
- 3 Дайте визначення поняття "виклик функції"?
- 4 Дайте визначення поняття "визначення функції"?
- 5 Дайте визначення поняття "формальний параметр"?
- 6 Дайте визначення поняття "фактичний параметр"?

- 7 Яким чином повертаються та передаються значення у функцію?
- 8 Яка роль оператора return при поверненні значень функцією?
- 9 Які можуть бути типи даних, що повертаються функцією?
- 10 Навіщо використовувати прототип функції до моменту її використання?

Завдання для самоперевірки засвоєного матеріалу на лекції

1 Скласти програму, в якій для обчислення значення функції $Y(X) = \sqrt{1-x}$ організована відповідна функція. Організувати обчислення значень функції для різних значень аргументу, що вводяться користувачем і циклі, поки користувач не припинить цей процес.

2 Скласти програму, яка містить дві, обумовлені користувачем функції, та виводить на екран наступний текст:

Three blind mice
 Three blind mice
 See how they run
 See how they run

Одна функція, що викликається двічі, виводить на екран перші два рядки, інша функція, яка також викликається двічі, виводить на екран наступні рядки.

3 Написати функцію, що виводить на екран послідовність цілих чисел, які належать заданому користувачем діапазону дійсних чисел $[a, b]$.

4 Для заданих користувачем значень a та k у написати функцію для обчислення значення $z = \begin{cases} k + 2, a < \frac{7}{5} \\ 3, a = \frac{7}{5} \\ \sqrt[4]{\ln(a^2) - k}, a > \frac{7}{5} \end{cases}$

Лекція № 6

з навчальної дисципліни «Основи програмування та алгоритмічні мови»

Тема Обмін між функціями. Повернення значення функцією. Оператор return. Передача параметрів за значенням та за посиланням

Мета заняття Ознайомити студентів із принципами обміну даними між функціями, вивчити способи передачі та повернення значень функцією, особливості використання оператора return

Матеріально-технічне забезпечення та дидактичні засоби, ТЗН:
конспект лекції, опорний конспект

Час – 2 години.

План проведення лекції

Структура лекції	Відведений час	Методичні вказівки
1 Організаційна частина	5 хв.	Включає в себе: привітання, яке має на меті привернути увагу, забезпечити контакт лектора з аудиторією; визначення присутності студентів на занятті
2 Актуалізація опорних знань, перевірка вивченого матеріалу та мотивація навчальної діяльності студентів	10 хв.	Включає в себе: вступне зауваження, мотивацію актуальності теми, перевірку попереднього матеріалу, формулювання мети лекції, огляд головних питань теми
3 Основна частина (викладення навчальних питань лекції)	60 хв.	Головне місце приділяється викладенню основного змісту матеріалу теми, його аналізу, узагальненню висунутих положень. Для успіху лекції важливе значення має поділ матеріалу на розділи, основні питання
4 Заключна частина Домашнє завдання: (відповідно до робочої програми)	5 хв.	Включає в себе: узагальнення, теоретичні і фактичні висновки, закріплення вивченого на лекції матеріалу, виставлення оцінок за роботу на занятті

Література

- 1 Шилдт, Г. С++: базовий курс, 3-е видання.: Пер. з англ. [Текст] / Г. Шилдт. – М.: Видавничий дім "Вільямс", 2010. – 624 с.: іл.
- 2 Страуструп, Б. Мова програмування С/С++ [Текст] / Б. Страуструп. – М.: Біном, 2006. – 501 с.: іл.

З Паппас, К.Х. Відлагодження в C/C++. Керівництво для розробників [Текст] / К.Х. Паппас, У.Х. Мюррей III. – М.: "Біном", 2009. – 452 с.

Навчальні матеріали лекції

ВСТУП

Мова C++ як і більшість інших мов програмування дозволяють створювати програми, що складаються з безлічі функцій, а також з одного або декількох файлів початкового коду.

При вивченні роботи функцій важливо розуміти, що таке локальна і що таке глобальна змінні. У мові програмування C++ глобальні (зовнішні) змінні оголошуються поза якою-небудь функцією. З їх допомогою зручно організовувати обмін даними між функціями, проте це вважається поганим тоном, оскільки легко заплутує програму. Локальні змінні в мові програмування C++ називають автоматичними. Область дії автоматичних змінних розповсюджується тільки на ту функцію, в якій вони були оголошені. Параметри функції також є локальними змінними.

1 Структурна декомпозиція програми. Обмін даними між функціями

Структурна організація файлу програми на мові C++, що містить декілька функцій, може виглядати трохи по-різному. Оскільки виконання починається з `main()`, то їй повинні бути відомі специфікації (імена, кількість і тип параметрів, тип значення, що буде повернуто) всіх функцій, які з неї викликаються. Звідси витікає, що оголошуватися функції повинні до того, як будуть викликані. А ось визначення функції вже може слідувати і до, і після `main()`.

Способи передачі даних у функцію, що викликається:

- шляхом передачі аргументів функції;
- з використанням глобальних змінних;
- через файли на зовнішніх пристроях, що запам'ятовують;

Способи передачі даних з функції, що викликається:

- через значення, що повертається;
- через формальні параметри, що викликаються по посиланню;
- шляхом зміни значень глобальних змінних;
- через файли на зовнішніх пристроях, що запам'ятовують;

Крім того, функція може виконувати яку-небудь дію, що не вимагає передачі з неї даних в функцію, яка її викликала.

2 Способи передачі даних у функцію та повернення значень

Передача аргументів за значенням.

Наступна програма передає два параметри з іменами `big` і `small` у функцію `display_values`. Функція `display_values`, у свою чергу, привласнює обом параметрам число 1001 і потім виводить значення кожного параметра. Коли функція завершується, програма поновлюється і виводить значення цих же параметрів:

```
#include <iostream.h>

void display_values(int a, int b)

{
    a = 1001;
    b = 1001;
    cout << "Значення у функції display_values рівні " << a << " і
" << b << endl;
}

void main(void)

{
    int big = 2002, small = 0;
    cout << "Значення до функції " << big << " і " << small <<
endl;
    display_values(big, small);
    cout << "Значення після функції " << big << " і " << small <<
endl;
}
```

Коли буде відкомпільовано і запущено цю програму, на екрані з'явиться наступний вивід:

Значення до функції 2002 і 0
Значення у функції display_values рівні 1001 і 1001
Значення після функції 2002 і 0

Як видно, значення параметрів у функції `display_values` були змінені (1001). Проте після завершення функції значення змінних `big` і `small` в `main` залишилися колишніми. Щоб зрозуміти, чому зміна параметрів не вплинула на змінні `big` і `small` в `main`, необхідно зрозуміти, як C++ передає параметри у функції.

Коли передаються параметри у функцію, C++ розміщує копії значень параметрів в тимчасовій ділянці пам'яті, званій стеком. Будь-які зміни, що виконуються функцією над параметрами, виявляються тільки усередині стека. Коли функція завершується, C++ скидає вміст стека разом з будь-якими змінами, які функція провела в параметрах. Оскільки функція не знає адресу пам'яті параметра, вона не може змінити його значення.

Передача параметрів по посиланням.

Щоб повідомити функції адресу параметра, програми повинні використовувати оператор адреси C++ (`&`). Наступний виклик функції ілюструє, як програма використовуватиме оператор адреси для передачі адрес змінних `big` і `small` у функцію `change_values`:

```
change_values (&big, &small); //Передача параметрів за адресою
```

Усередині функції необхідно повідомити C++, що програма передаватиме параметри за допомогою адреси. Для цього оголошуються змінні-вказівники, що позначаються "*" перед іменем кожної змінної, як показано нижче:

```
void change_values (int *big, int *small) //Вказівник на тип int
```

Змінна-вказівник є змінною, яка містить адресу пам'яті. Усередині функції необхідно повідомити C++, що функція працює з адресою параметра. Для цього ви передуйте імені параметра "*", як показано нижче:

```
*big  = 1001;
*small = 1001;
```

Наступна програма використовує оператор адреси для передачі адрес параметрів `big` і `small` у функцію `change_values`. Функція, у свою чергу, використовує вказівники ділянок пам'яті параметрів. Отже, зміни параметрів, зроблені функцією, залишаються і після завершення функції:

```
#include <iostream.h>

void change_values (int *a, int *b)
{
    *a = 1001;
    *b = 1001;
    cout << "Значення у функції display_values" << " рівні " << *a
    << " і " << *b << endl;
}

void main(void)
{
    int big = 2002, small = 0;
    cout << "Значення перед функцією " << big << " і " << small <<
    endl;
    change_values(&big &small);
    cout << "Значення після функції " << big << " і " << small <<
    endl;
}
```

Коли буде відкомпільовано і запущено цю програму, на екрані з'явиться наступний вивід:

Значення перед функцією 2002 і 0

Значення у функції `display_values` рівні 1001 і 1001

Значення після функції 1001 і 1001

Як видно, значення, які функція `change_values` привласнює параметрам, залишаються і після завершення функції. Щоб зрозуміти, чому зміни, які функція виконала над змінними, залишилися після її завершення, необхідно пригадати, що функція має доступ до елементу пам'яті кожної змінної. Якщо передаються параметри за адресою, C++ поміщає адресу кожної змінної в стек.

Використовуючи вказівник (адреси пам'яті) усередині функції, `change_values` може звернутися до пам'яті за адресою кожного параметра, змінюючи значення параметрів, що і потрібно.

Виклик функцій за допомогою масивів.

Це питання розглядається тому, що ця операція є виключенням по відношенню до звичайної передачі параметрів, що виконується шляхом виклику за значенням.

Коли як аргумент функції використовується масив, то функції передається його адреса. У цьому і полягає виключення по відношенню до правила, яке свідчить, що при передачі параметрів використовується виклик за значенням. У разі передачі масиву функції її внутрішній код працює з реальним вмістом цього масиву і цілком може змінити цей вміст. Проаналізуємо, наприклад, функцію `print_upper()`, яка друкує свій строковий аргумент на верхньому регістрі:

```
#include <iostream.h>
#include <stdio.h>
#include <ctype.h>

void print_upper(char *string);

int main(void)
{
    char s[80];

    cout<<"Введіть рядок символів: ";
    gets(s);
    print_upper(s);
    cout<<endl<<"s тепер на верхньому регістрі: "<<s;
    return 0;
}

/* Друкувати рядок на верхньому регістрі. */
void print_upper(char *string)
{
```

```

register int t;

for(t=0; string[t]; ++t) {
    string[t]= toupper(string[t]);
    putchar(string[t]);
}
}

```

Ось що буде виведене у разі фрази "This is a test." (це тест):

Введіть рядок символів: This is a test.

THIS IS A TEST.

s тепер у верхньому регістрі: THIS IS A TEST.

Правда, ця програма не працює з символами кирилиці.

Після виклику `print_upper()` вміст масиву `s` в `main()` переводиться в символи верхнього регістра. Якщо це не потрібно, програму можна написати таким чином:

```

#include <iostream.h>
#include <stdio.h>
#include <ctype.h>

void print_upper(char *string);

int main(void)
{
    char s[80];

    cout<<"Введіть рядок символів: ";
    gets(s);
    print_upper(s);
    cout<<endl<<"s не змінилася: "<<s;

    return 0;
}

void print_upper(char *string)
{
    register int t;

    for(t=0; string[t]; ++t)
        putchar(toupper(string[t]));
}

```

Ось який цього разу вийде фразза "This is a test.":

Введіть рядок символів: This is a test.

THIS IS A TEST.

s не змінилася: This is a test.

Цього разу вміст масиву не змінився, тому що усередині `print_upper()` не змінювалися його значення.

Класичним прикладом передачі масивів у функції є стандартна бібліотечна функція `gets()`. Хоча `gets()`, яка знаходиться у стандартній бібліотеці, і складніша, ніж пропонована версія `xgets()`, але за допомогою функції `xgets()` можна отримати уявлення про те, як працює `gets()`.

```
/* Спрощена версія стандартної бібліотечної функції gets(). */
char *xgets(char *s)
{
    char ch *p;
    int t;
    p = s; /* xgets() повертає покажчик s */
    for(t=0; t<80; ++t){
        ch = getchar();
        switch(ch){
            case '\n':
                s[t]= '\0'; /* завершує рядок */
                return p;
            case '\b':
                if(t>0) t--;
                break;
            default:
                s[t]= ch;
        }
    }
    s[79]= '\0';
    return p;
}
```

Функцію `xgets()` слід викликати з вказівником `char *`. Ним, звичайно ж, може бути ім'я символьного масиву, яке за визначенням є вказівником `char *`. На самому початку програми `xgets()` виконується цикл `for` від 0 до 80. Це не дасть вводити з клавіатури рядки, що містять більше 80 символів. При спробі введення більшої кількості символів відбувається повернення з функції (у справжній функції `gets()` такого обмеження немає). Оскільки в мові C++ немає вбудованої перевірки меж, програміст винен сам поклопотатися, щоб в будь-якому масиві, використовуваному при виклику `xgets()`, поміщалося не менше 80 символів. Коли символи вводяться з клавіатури, вони відразу записуються в рядок. Якщо користувач натискає клавішу `<Backspace>`, то лічильник `t` зменшується на 1, а з масиву віддаляється останній символ,

введений перед натисненням цієї клавіші. Коли користувач натисне <ENTER>, в кінець рядка запишеться нуль, тобто ознака кінця рядка. Оскільки масив, використаний для виклику `xgets()`, модифікується, то при поверненні з функції в нім знаходитимуться введені користувачем символи.

3 Повернення результату за допомогою оператора `return`

Повернення результату з функції в функцію, яка її викликала, здійснюється оператором

```
return (значення);
```

Працює оператор так. Обчислюється значення виразу, вказаного після `return` і перетворюється до типу повертаного функцією значення. Виконання функції завершується, а обчислене значення передається в функцію, яка її викликала. Будь-які оператори, наступні у функції за оператором `return`, ігноруються. Програма продовжує свою роботу з оператора наступного за оператором виклику даної функції.

Оператор `return` може бути відсутнім у функціях типу `void`, якщо повернення відбувається перед фігурною дужкою, що закривається, і у функції `main`.

Також функція може містити декілька операторів `return`, якщо це визначено потребами алгоритму. Наприклад, в наступній програмі функція `f` порівнює значення змінної з нулем. Якщо число додатне, то в програму передається значення рівне 1, якщо від'ємне, то -1, а якщо нульове, то 0.

```
#include <iostream>
int f(int a)
{
    if (a>0) return 1;
    else if (a<0) return -1;
    else return 0;
}
//функцію можна переписати так
//int f(int a)
//{ int r;
//if (a>0) r=1;
//else if (a<0) r=-1;
//else r=0;
```

```
//return(r);
//}
int main ()
{
int b;
cout<<"b=";
cin>>b;
if (f(b)==1) cout<<"positive value \n";
else if (f(b)==-1) cout<<"negative value \n";
else cout<<"zero value \n";
system ("pause");
} //у функції main відсутній оператор return
```

Контрольні питання:

- 1 Що таке функція?
- 2 Які поняття пов'язані з використанням функції?
- 3 Що таке прототип та опис функції? Яка між ними різниця?
- 4 Що таке формальні та фактичні параметри?
- 5 Яка область видимості локальних та глобальних змінних?
- 6 Які специфікатори класів пам'яті існують?
- 7 З яких частин складається програма на мові C++?
- 8 Яким чином повертаються та передаються значення у функцію?
- 9 Чим відрізняється передача параметрів за значенням та за посиланням?
- 10 Яка особливість передачі параметру масиву?
- 11 Яка роль оператора return при поверненні значень функцією?

Лекція № 8

з навчальної дисципліни «Основи програмування та алгоритмічні мови»

Тема Основні відомості про функції. Поняття функції. Представлення функції. Формальні та фактичні параметри

Мета заняття Засвоїти основні поняття функції, відомості про функції, передача фактичних параметрів у функцію, заміна фактичними параметрами формальних

Матеріально-технічне забезпечення та дидактичні засоби, ТЗН:
конспект лекції, опорний конспект

Час – 2 години.

План проведення лекції

Структура лекції	Відведений час	Методичні вказівки
1 Організаційна частина	5 хв.	Включає в себе: привітання, яке має на меті привернути увагу, забезпечити контакт лектора з аудиторією; визначення присутності студентів на занятті
2 Актуалізація опорних знань, перевірка вивченого матеріалу та мотивація навчальної діяльності студентів	10 хв.	Включає в себе: вступне зауваження, мотивацію актуальності теми, перевірку попереднього матеріалу, формулювання мети лекції, огляд головних питань теми
3 Основна частина (викладення навчальних питань лекції)	60 хв.	Головне місце приділяється викладенню основного змісту матеріалу теми, його аналізу, узагальненню висунутих положень. Для успіху лекції важливе значення має поділ матеріалу на розділи, основні питання
4 Заклучна частина Домашнє завдання: (відповідно до робочої програми)	5 хв.	Включає в себе: узагальнення, теоретичні і фактичні висновки, закріплення вивченого на лекції матеріалу, виставлення оцінок за роботу на занятті

Література

- 1 Шилдт, Г. С++: базовий курс, 3-е видання.: Пер. з англ. [Текст] / Г. Шилдт. – М.: Видавничий дім "Вільямс", 2010. – 624 с.: іл.
- 2 Страуструп, Б. Мова програмування С/С++ [Текст] / Б. Страуструп. – М.: Біном, 2006. – 501 с.: іл.

З Паппас, К.Х. Відлагодження в C/C++. Керівництво для розробників [Текст] / К.Х. Паппас, У.Х. Мюррей III. – М.: "Біном", 2009. – 452 с.

Навчальні матеріали лекції

1 Поняття функції. Основні відомості про функції

Із збільшенням об'єму програми стає неможливо утримувати в пам'яті всі деталі. Щоб зменшити складність програми, її розбивають на частини. У C++ задача може бути розділена на простіші підзадачі за допомогою функцій. Розрізняють системні (у складі систем програмування) і власні функції.

Розбиття програм на функції дає наступні переваги:

- функцію можна викликати з різних місць програми, що дозволяє уникнути повторного програмування;
- одну і ту ж функцію можна використовувати в різних програмах;
- функції підвищують рівень модульності програми і полегшують її проектування;
- використання функцій полегшує читання і розуміння програми, прискорює пошук і виправлення помилок.

2 Визначення функцій

Функція – це іменована послідовність описів і операторів, що виконує закінчену дію, наприклад, пошук максимального значення серед двох чисел, пошук модулю числа тощо.

Функція, по-перше, є одним з похідних типів C++, а, по-друге, мінімальним виконуваним модулем програми.

Будь-яка функція повинна бути оголошена і визначена.

Оголошення функції (прототип) задає ім'я функції, тип значення, яке повертається, і список параметрів, які передаються.

Визначення функції містить, окрім прототипу (оголошення), тіло функції, яке є послідовністю описів і операторів.

Загальний вигляд визначення функції:

```
тип_поверт_знач ім'я_функції([список_формальних_параметрів])
{
    тіло_функції;

    return [вираз];
}
```

Приклад визначення функції знаходження суми двох чисел.

```
/* функція з ім'ям sum(), якій передаються 2 параметри -
дійсні змінні x, y. Функція повертає дійсне значення */
float sum(float x, float z)
{
    float y; //оголошення змінної дійсної y

    /* знаходження суми змінних "x", "z" та запис
результату у змінну "y" */
    y=x+z;

    return y; //повернення значення із змінної "y" з функції
}
```

Тип значення, яке повертається, може бути будь-яким, окрім масиву.

Список формальних параметрів – це ті величини, які потрібно передати у функцію. Елементи списку розділяються комами. Для кожного параметра вказується тип і ім'я. У оголошенні імена можна не вказувати.

Тіло функції – це блок або складений оператор. У середині функції не можна визначити іншу функцію.

У тілі функції може бути оператор, який повертає значення функції в точку її виклику. Він може мати 2 форми:

– return вираз;

– return.

Перша форма використовується для повернення результату, тому "вираз" повинен мати той же тип, що і тип функції у визначенні.

Друга форма використовується, якщо функція не повертає значення, тобто має тип void.

Для того, щоб виконувалися оператори, записані в тілі функції, функцію необхідно викликати. При виклику указуються: ім'я функції і фактичні параметри, які будуть передані функції.

Фактичні параметри замінюють формальні параметри при виконанні операторів тіла функції. Фактичні і формальні параметри повинні співпадати по кількості і типу.

Оголошення функції повинне знаходитися в тексті раніше виклику функції, щоб компілятор міг здійснити перевірку правильності виклику. Якщо функція має тип не void, то її виклик може бути операндом виразу.

Приклад програми обчислення факторіалу введеного числа.

```
#include <iostream.h>
#include <conio.h>

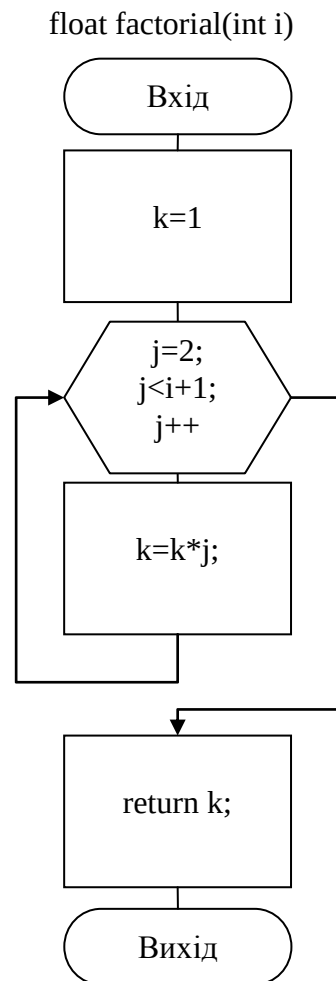
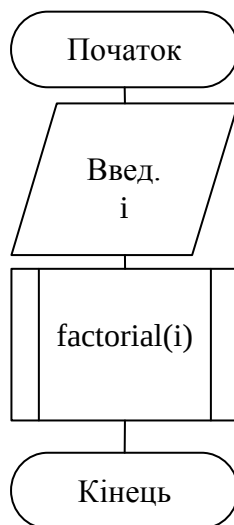
/* визначення функції з ім'ям factorial(), якій передається
одне ціле значення – змінна "i". Функція повертає дійсне
значення. */
float factorial(int i)
{
    int j;
    float k;
    k=1;
    for (j=2;j<i+1;j++)
    {
        k=k*j;
    }
    /* повернення в програму обчисленого значення */
    return k;
}

/* головна функція програми*/
int main()
{
```

```

clrscr();
int i;
cout<<endl<<"Введіть ціле число: ";
cin>>i;
/* виклик функції factorial() з фактичним параметром "i" */
cout<<endl<<i<<"!="<<factorial(i);
getch();
return 0;
}

```



Результат роботи програми:

Введіть ціле число: 7
7.000000!=5040.000000

3 Прототип функцій

У практиці програмування бувають випадки, коли тіло функції розташовується в програмі нижче за місцем, де виконується її виклик, або функція взагалі компілюється окремо. В цьому випадку до використання функції повинен бути вказаний прототип функції (або оголошення функції), що містить:

- тип функції;
- ім'я функції;
- інформацію про параметри.

Прототип необхідний для того, щоб компілятор зміг здійснити перевірку відповідності типів і кількості фактичних параметрів, які передаються, типам і кількості формальних параметрів. Оголошення функції має той же вигляд, що і визначення функції, проте тіло функції відсутнє, і імена формальних параметрів також можуть бути відсутніми.

Загальний вигляд визначення прототипу функції.

```
тип_поверт_знач ім'я_функції([список_формальних_параметрів]);
```

Приклад визначення прототипу (оголошення) функції модуля числа:

```
int abs(int i);
```

При програмуванні на мові C/C++ широко використовуються бібліотечні функції. Ці функції були заздалегідь розроблені і записані до складу системи програмування. Прототипи бібліотечних функцій знаходяться в спеціальних заголовних файлах з розширенням *.h, які необхідно підключати за допомогою директиви #include.

Приклад програми генерації таблиці чисел 2^n , де n – числа від 1 до 10.

```
#include <iostream.h>
#include <conio.h>

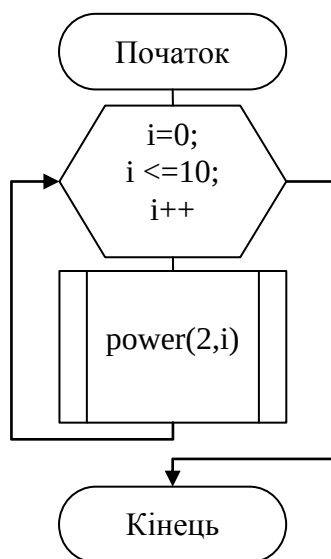
/* Опис прототипу функції power() з 2-ма цілими параметрами -
змінні base, index. Функція повертає ціле значення. */
int power(int base, int index);

//головна функція програми
int main()
{
    clrscr();
    int i;
    for (i=0; i <=10; i++ )
    {
        cout<<power(2,i)<<"    ";
    }
    getch();
    return 0;
}

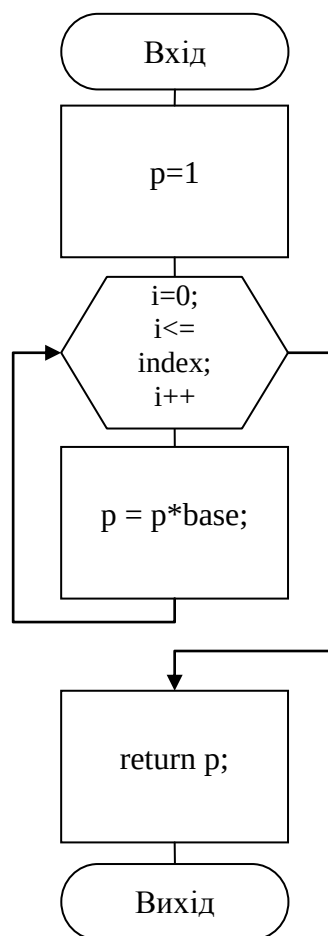
int power(int base, int index)
{
    int i, p;
    p = 1;
    for (i = 0; i <= index; i++)
    {
        p = p*base;
    }
    return p;
}
```

Результат роботи програми:

2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048
--



`int power(int base, int index)`



4 Виклик функцій

Загальний вигляд виклику функції:

```
ім'я_функції([список_фактичних_параметрів]);
```

Фактичний параметр – це величина, яка привласнюється формальному параметру при виклику функції. Таким чином, формальний параметр – це змінна у функції, що викликається, а фактичний параметр – це конкретне значення, привласнене цій змінній функцією, яка викликає.

Фактичний параметр може бути константою, змінною або виразом.

Фактичний параметр спочатку обчислюється, а потім його значення передається у функцію. Якщо у функцію потрібно передати декілька параметрів, то вони записуються через кому.

5 Повернення значення в точку виклику функції

Функція може передати в точку виклику програми тільки одне значення, для цього використовується оператор `return`.

```
return [вираз];
```

Дія оператора наступна: значення виразу, включеного в дужки, обчислюється і передається в точку виклику функції. Значення, яке повертається, може використовуватися в програмі як частина деякого виразу.

Оператор `return` виконує і іншу дію. Він завершує виконання функції і передає управління наступному операторові в функції, яка здійснила виклик.

Оператор `return` не обов'язково повинен знаходитися в кінці тіла функції.

Приклад використання `return` в середині функції.

```
void function()
{
    ...
    return;
    ...
}
```

У прикладі оператор `return` завершує виконання функції `function()` і передає управління в функцію, яка здійснила виклик. Ніяке значення при цьому не передається.

Виклик функції залежить від того, повертає вона значення чи ні. Якщо функція повертає значення, то її виклик проводиться з математичних і логічних виразів.

Приклад функції визначення максимуму серед двох чисел.

```
#include <iostream.h>
#include <conio.h>

int max(int a, int b)
{
    if (a>b)
        return a;
```

```

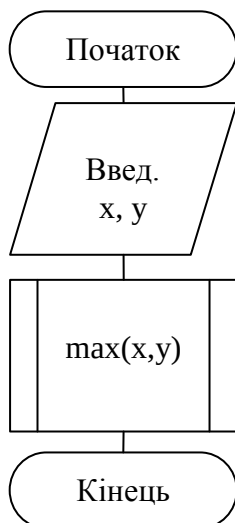
        else
            return b;
    }
    int main()
    {
        clrscr();
        int x, y;
        cout<<endl<<"Введіть 2 числа: ";
        cin>>x>>y;
        cout<<endl<<"Максимум з них = "<<max(x,y);
        getch();
        return 0;
    }

```

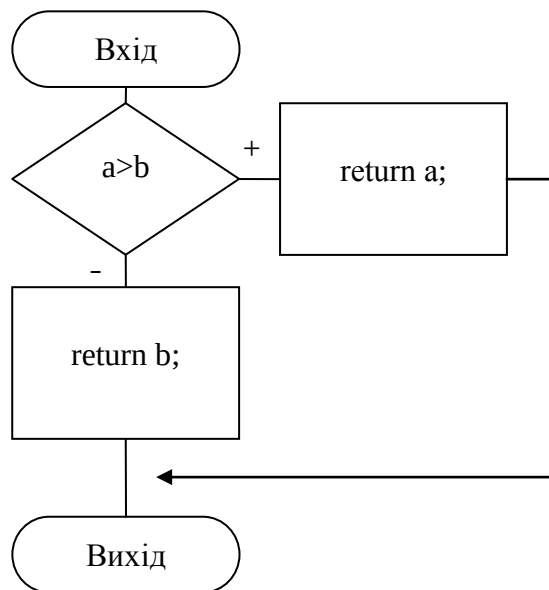
Результат роботи програми:

Введіть 2 числа: 10 7

Максимум з них = 10



int max(int a, int b)



Функції можуть і не повертати значення, а просто виконувати деякі обчислення або вивід на екран. У такому разі вказується порожній тип даних `void`, і оператор `return` не використовується.

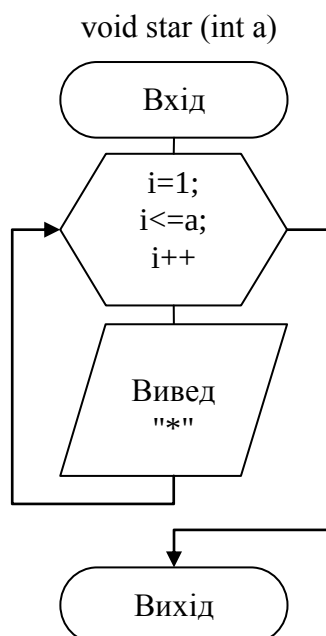
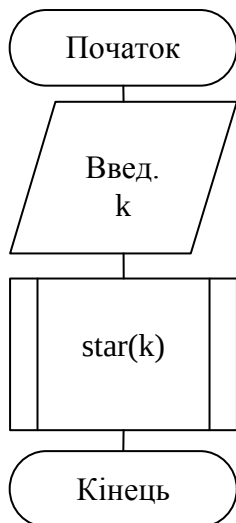
Приклад функції, що не повертає значення, - `star(int a)`.

```
#include <iostream.h>
#include <conio.h>

void star (int a);

int main()
{
    clrscr();
    int k;
    cout<<endl<<"Скільки символів надрукувати?";
    cin>>k;
    star(k);
    getch();
    return 0;
}

void star (int a)
{
    int i;
    for(i=1; i<=a;i++)
    {
        cout<<"*";
    }
}
```



Контрольні питання:

- 1 Дайте визначення поняття "функція"?
- 2 Дайте визначення поняття "прототип (оголошення) функції"?
- 3 Дайте визначення поняття "виклик функції"?
- 4 Дайте визначення поняття "визначення функції"?
- 5 Дайте визначення поняття "формальний параметр"?
- 6 Дайте визначення поняття "фактичний параметр"?
- 7 Яким чином повертаються та передаються значення у функцію?
- 8 Яка роль оператора return при поверненні значень функцією?
- 9 Які можуть бути типи даних, що повертаються функцією?
- 10 Навіщо використовувати прототип функції або визначати функції до моменту її використання?

Лекція № 9-10

з навчальної дисципліни «Основи програмування та алгоритмічні мови»

Тема Поняття про символ, рядок. Уявлення символів та рядків у пам'яті.
Введення-виведення символів та рядків.

Мета заняття Ознайомити студентів із поняттям вільних масивів, визначенням поняття "рядок", правилами вживання динамічних (вільних) масивів рядків

Матеріально-технічне забезпечення та дидактичні засоби, ТЗН:
конспект лекції, опорний конспект

Час – 4 години.

План проведення лекції

Структура лекції	Відведений час	Методичні вказівки
1 Організаційна частина	5 хв.	Включає в себе: привітання, яке має на меті привернути увагу, забезпечити контакт лектора з аудиторією; визначення присутності студентів на занятті
2 Актуалізація опорних знань, перевірка вивченого матеріалу та мотивація навчальної діяльності студентів	10 хв.	Включає в себе: вступне зауваження, мотивацію актуальності теми, перевірку попереднього матеріалу, формулювання мети лекції, огляд головних питань теми
3 Основна частина (викладення навчальних питань лекції)	60 хв.	Головне місце приділяється викладенню основного змісту матеріалу теми, його аналізу, узагальненню висунутих положень. Для успіху лекції важливе значення має поділ матеріалу на розділи, основні питання
4 Заключна частина Домашнє завдання: (відповідно до робочої програми)	5 хв.	Включає в себе: узагальнення, теоретичні і фактичні висновки, закріплення вивченого на лекції матеріалу, виставлення оцінок за роботу на занятті

Література

- 1 Шилдт, Г. С++: базовий курс, 3-е видання.: Пер. з англ. [Текст] / Г. Шилдт. – М.: Видавничий дім "Вільямс", 2010. – 624 с.: іл.
- 2 Страуструп, Б. Мова програмування С/С++ [Текст] / Б. Страуструп. – М.:

Біном, 2006. – 501 с.: іл.

3 Паппас, К.Х. Відлагодження в C/C++. Керівництво для розробників [Текст] / К.Х. Паппас, У.Х. Мюррей III. – М.: "Біном", 2009. – 452 с.

Навчальні матеріали лекції

1 Вільні масиви рядків

Для символічних даних в C++ введений тип `char`. Для представлення символічної інформації використовуються символи, символічні змінні і текстові константи.

Рядок в C++ – це масив символів, що закінчується нуль-символом, – `'\0'` (нуль-термінатором). По положенню нуль-термінатора визначається фактична довжина рядка. Кількість елементів в такому масиві на 1 більше, ніж реальна кількість символів в рядку.

'A'	"A\0"
Символьна змінна (1 байт)	Рядок (2 байти)

Рисунок 1.1 – Порівняння символу та рядка

У програмі рядки можуть визначатися наступним чином:

1) Рядкові константи.

Будь-яка послідовність символів, ув'язнена в подвійні лапки, розглядається як рядкова константа. Послідовність символів буде розміщена в оперативній пам'яті ЕОМ, включаючи нульовий байт. Під рядок виділяються елементи оперативної пам'яті, що йдуть послідовно. Для кодування одного символу достатньо одного байту.

```
const char *s="Приклад рядка\n" ; //рядкова константа
```

Рядкові константи розміщуються в статичній пам'яті. Сама послідовність символів в подвійних лапках трактується як адреса рядка.

Це аналогічно використанню імені масиву, який використовується як вказівник на розташування масиву.

2) Масиви символів.

При визначенні масиву символьних рядків необхідно повідомити компілятору необхідний розмір пам'яті.

```
char m2[24]="Я тут, а зараз я - там";
```

Компілятор також може самостійно визначити розмір масиву символів.

```
char m3[]="Я навчаюсь на 1 курсі";
```

```
char m3[]={'Я', '-', 'с', 'т', 'у', 'д', 'е', 'н', 'т', '\0'};
```

Як і для інших масивів, m2 і m3 є покажчиками на перші елементи масивів.

```
m2 == &m2[0]
*m2 == 'Я'
*(m2 + 2) == 'т'
m3 == &m3[0]
*(m3 + 1) == '-'
```

3) Через вказівник на символьний тип.

Для створення рядка можна використовувати вказівник.

```
char *m4 = "31 грудня - свято Нового року";
```

Це майже те ж саме, що і

```
char m5[] = "Почекай небагато, відпочинеш і ти";
```

У обох випадках m4 і m5 є вказівниками на рядки, і розміри масивів визначаються компілятором. Проте між m4 і m5 є і деяка різниця.

Масив або вказівник. У другому випадку оголошення масиву викликає створення в пам'яті масиву з 34 елементів (поодинокі на кожен символ плюс один на завершуючий символ '\0'). Кожен елемент ініціалізується відповідним символом. Надалі компілятор розглядатиме ім'я m5 як синонім адреси першого елемента масиву, тобто &m5[0]. Слід зазначити, що m5 є константою-вказівником. Не можна змінити m5, оскільки це означало б зміну положення (адреси) масиву в пам'яті. Можна використовувати операції, подібні (m5 + 1),

для ідентифікації наступного елементу масиву, проте не дозволяється вираз `++m5`. Оператор збільшення можна використовувати з іменами змінних, але не констант.

У випадку з вказівником `m4` в пам'яті створюється 30 елементів для запам'ятовування рядка. Але, крім того, виділяється ще один елемент пам'яті для змінної `m4`, що є вказівником. Спочатку ця змінна указує на початок рядка, але її значення може змінюватися. Тому допустимо використовувати операцію приросту на одиницю: `++m4` указуватиме на другий символ рядка — `'1'`.

4) Масиви рядків.

Загальна форма оголошення масиву рядків:

```
char ім'я_масиву[кіл-ть_рядків][кіл-ть_символів];
```

Приклад.

```
char poet[4][23];
```

Тут визначається масив з 4-х рядків з ім'ям `poet`. Кожен рядок матиме максимальну довжину в 23 символи.

Програміст повинен розуміти, що рядок — це масив символів, і під неї необхідно виділяти ділянку оперативної пам'яті.

Приклади.

```
const char c='c'; //символ – займає один байт, його значення
                  //не змінюється
char a,b; //символьні змінні, займають по одному байту,
          //значення можуть змінюватися
```

Привласнити значення рядку за допомогою оператора привласнення не можна. Помістити рядок в масив можна або при введенні, або за допомогою ініціалізації.

Приклади.

```
char *s="String5"; // виділяється 8 байт для рядка

char *ss; //описаний вказівник
```

```
ss="String6"; // пам'ять не виділяється, тому програма
              //завершиться аварійно
char *sss=new char[10]; // виділяємо динамічну пам'ять
strcpy(sss,"String7"); // копіюємо рядок в пам'ять
```

Приклад використання різних способів описання рядків.

```
void main()
{
char s1[10]="string1";
int k=sizeof(s1);
cout<<s1<<"\t"<<k<<endl;
char s2[]="string2";
k=sizeof(s2);
cout<<s2<<"\t"<<k<<endl;
char s3[]={ 's','t','r','i','n','g',' ','3','\0' };
k=sizeof(s3);
cout<<s3<<"\t"<<k<<endl;
char *s4="string4"; // вказівник на рядок, його не можна змінити
k=sizeof(s4);
cout<<s4<<"\t"<<k<<endl;
}
```

Результат:

string1 10 – виділено 10 байт, зокрема під \0

string2 8 – виділено 8 байт (7+1байт під \0)

string3 8 – виділено 8 байт (7+1байт під \0)

string4 4 – розмір покажчика

2 Правила вживання вільних масивів рядків

Для введення і виведення символьних даних в бібліотеці мови C визначені наступні функції.

`int getchar(void)` – здійснює введення одного символу з вхідного потоку, при цьому вона повертає один байт інформації (символ) у вигляді значення типу `int`. Це зроблено для розпізнавання ситуації, коли при читанні буде досягнутий кінець файлу.

`int putchar (int c)` – поміщає в стандартний вихідний потік символ `c`.

`char* gets(char *s)` – прочитає рядок `s` із стандартного вхідного потоку до появи символу `'\n'`, сам символ `'\n'` в рядок не заноситься.

`int puts(const char*s)` – записує рядок в стандартний вихідний потік, додаючи в кінець рядка символ `'\n'`, у разі вдалого завершення повертає значення більше або рівне 0 і негативне значення (`EOF=-1`) у разі помилки.

Приклади.

```
1. char s[20];  
cin>>s; //введення рядка із стандартного потоку  
cout<<s; //виведення рядка в стандартний потік
```

Результат роботи програми:

При введенні рядка "123 456 789", читання байтів здійснюється до першого пропуску, тобто в рядок `s` занесеться тільки перше слово рядка "123\0", отже, виведеться: "123".

```
2. char s[20];  
gets(s); //введення рядка із стандартного потоку  
puts(s); //виведення рядка в стандартний потік
```

Результат роботи програми:

При введенні рядка "123 456 789", читання байтів здійснюється до символу `'\n'`, тобто в `s` занесеться рядок "123 456 789\n\0", при виведенні рядка функція `puts` повертає ще один символ `'\n'`, отже, буде виведений рядок "123 456 789\n\n".

```
3. char s[20];
scanf("%s",s); //введення рядка із стандартного потоку
printf("%s",s); //виведення рядка в стандартний потік
```

Результат роботи програми:

При введенні рядка "123 456 789", читання байтів здійснюється до першого пропуску, тобто в рядок s занесеться тільки перше слово рядка "123\0", отже, виведеться: "123". Оскільки s – ім'я масиву, тобто адреса його першого елемента, операція & у функції scanf не використовується.

Для роботи з рядками існують спеціальні бібліотечні функції, які містяться в заголовному файлі string.h.

Таблиця 2.1 – Функції для роботи з рядками

Прототип функції	Короткий опис	Примітка
unsigned strlen (const char *s);	Обчислює довжину рядка s	
int strcmp(const char *s1, const char *s2)	Порівнює рядки s1 і s2	Якщо s1<s2, то результат від'ємний, якщо s1==s2, то результат рівний 0, якщо s2>s1 – результат додатний
int strncmp(const char *s1, const char *s2, int n);	Порівнює перші n символів рядків s1 і s2	Якщо s1<s2, то результат від'ємний, якщо s1==s2, то результат рівний 0, якщо s2>s1 – результат додатний
char*strcpy(char *s1, const char*s2);	Копіює символи рядка s1 в рядок s2	
char*strncpy(char *s1, const char *s2, int n);	Копіює n символів рядка s1 в рядок s2	Кінець рядка відкидається або доповнюється пропусками
char*strcat(char *s1, const char *s2);	Приписує рядок s2 до рядка s1	
char*strncat(char *s1, const char *s2, int n);	Приписує перші n символів рядка s2 до рядка s1, завершуючи s1 символом '\0'	

Прототип функції	Короткий опис	Примітка
<code>char*strdup(const char*s);</code>	Виділяє пам'ять і переносить в неї копію рядка s.	При виділенні пам'яті використовується функція <code>malloc()</code>
<code>char *strset(char *s, char c)</code>	Заповнює весь рядок s символами c	
<code>char *strnset(char *s, char c, int n)</code>	Заповнює перші n символів рядка s символами c	
<code>char *strchr(char *s, char c)</code>	Відшукує у рядку s перше входження символу c і повертає значення вказівника на даний символ	Якщо символу не знайдений, повертається значення <code>NULL</code>
<code>char *strrchr(char *s, char c)</code>	Відшукує у рядку s останнє входження символу c і повертає значення вказівника на даний символ	Якщо символу не знайдений, повертається значення <code>NULL</code>

Приклад.

Вводити з клавіатури рядки, доки не буде введено 5 рядків, що закінчуються будь-якими пробільними символами. Вивести усі введені рядки та рядки, що відповідають вимогам завдання. Відсортувати рядки. Вивести найкоротший рядок (рядки, якщо їх декілька).

```
#include <iostream.h>
#include <conio.h>
#include <string.h>

int main()
{
    //максимальна кількість потрібних за завданням рядків
    const int max_string=5;
    //загальна кількість введених рядків
    int count_allstring=0;
    //кількість тільки тих рядків, які підходять за умовою
    int count_string=0;
```

```

//масив символів для введення одного рядка
char one_string[255];
//масив вказівників на рядки даних, які будуть вводиться
//з клавіатури. Під кожний рядок пам'ять буде виділятися
//після визначення реальної довжини кожного введенного рядка
char **pall_string = new char*[max_string];
//масив вказівників на рядки даних, які виконують умову
//завдання, тобто містять символ пробілу
//під кожний рядок пам'ять буде виділятися після визначення
//реальної довжини кожного введенного рядка
char **pstring = new char*[max_string];
//змінна для довжини найкоротшого рядка
int min_length=256;
//очищення екрану
clrscr();
//цикл з передумовою
//поки кількість рядків, підходящих за умовою, менше необхідної кількості
while(count_string<max_string)
{
    //виведення повідомлення на екран
    cout<<endl<<"Enter "<<count_allstring+1<<"-th string, please: ";
    //зчитування рядка даних з клавіатури
    gets(one_string);
    //виділяємо динамічну пам'ять під новий рядок
    pall_string[count_allstring]=new char[strlen(one_string)];
    //копіюємо щойно введенний рядок до масиву ВСІХ рядків
    strcpy(pall_string[count_allstring],one_string);
    //збільшуємо на 1 змінну кількості ВСІХ рядків
    count_allstring++;
    //перевіряємо останній символ щойно введенного рядка
    //на рівність з символом ' ' (пробіл) або '\t' (табуляція)
    if(one_string[strlen(one_string)-1]==' ' ||
        one_string[strlen(one_string)-1]=='\t')
    {
        //виділяємо динамічну пам'ять під новий рядок
        pstring[count_string]=new char[strlen(one_string)];
    }
}

```

```

        //копіємо щойно введений рядок до масиву рядків, які
        //підходять за умовою
        strcpy(pstring[count_string],one_string);
        //збільшуємо змінну кількості рядків, які виконують
        //умову завдання - містять пробільні символи
        count_string++;
        //пошук найкоротшого рядка даних
        if (strlen(one_string)<min_length)
        //якщо чергова довжина рядка менша ніж у змінній min_length
        //заносимо нову довжину рядка до змінної min_length
            min_length=strlen(one_string);
    }

}

//сортування всіх рядків
//бульбашковим способом
for(int i=count_allstring-1;i>=1;i--)
    for (int j=0;j<i;j++)
    {
        //якщо рядок № j за алфавітом більший рядка № j+1
        if (strcmp(pall_string[j],pall_string[j+1])>0)
        {
            //міняємо місцями рядки
            strcpy(one_string,pall_string[j]);
            strcpy(pall_string[j],pall_string[j+1]);
            strcpy(pall_string[j+1],one_string);
        }
    }

//виведення всіх рядків
clrscr();
cout<<endl<<"There are all strings, which user have entered: "<<endl;
for(int i=0;i<count_allstring;i++)
    //виведення i-го рядка
    puts(pall_string[i]);
//виведення рядків, які підходять за умовою задачі
cout<<endl<<"There are strings, which we need: "<<endl;

```



```

for(int i=0;i<max_string;i++)
    //виведення i-го рядка
    puts(pstring[i]);
//вивідення найкоротшого рядка
cout<<endl<<endl<<"There is a short string, which we need: "<<endl;
for(int i=0;i<max_string;i++)
    //якщо довжина i-го рядка дорівнює мінімальній довжині
    if (strlen(pstring[i])==min_length)
        //тоді виведення i-го рядка на екран
        puts(pstring[i]);
//виклик функції затримки екрана до натиснення будь-якої клавіші
getch();
}

```

Контрольні питання

- 1 Який тип даних призначений для символьних даних?
- 2 Яким символом повинен закінчуватися будь-який рядок в C/C++?
- 3 Чи однаковий запис констант 'Хс' та "Хс"?
- 4 Яку довжину буде мати рядок "string_max"?
- 5 Що називається рядковою константою?
- 6 Який обсяг пам'яті займає кожний символ у рядковій константі?
- 7 Що потрібно повідомити компілятору при оголошенні масиву символів?
- 8 Чи може компілятор самостійно визначити довжину масиву символів?
Якщо так, в якому випадку?
- 9 Чи можна змінити інформацію, на яку посилається вказівник символьного типу?
- 10 Як оголосити масив рядків?
- 11 Чи можна задати значення рядка за допомогою оператора "="?
- 12 Опишіть призначення функції getchar().
- 13 Опишіть призначення функції putchar().
- 14 Опишіть призначення функції gets().

- 15 Опишіть призначення функції `puts()`.
- 16 Яка головна відмінність між потоком введення `cin`, функцією введення даних `scanf()` та функцією введення рядків `gets()`?
- 17 Яка функція використовується для визначення довжини рядка?
- 18 Яка функція використовується для копіювання рядків?
- 19 Яка функція використовується для порівняння декількох рядків?
- 20 Яка функція використовується для "склеювання" декількох рядків?

Лекція № 11

з навчальної дисципліни «Основи програмування та алгоритмічні мови»

Тема Типи даних, що визначені користувачем. Структури в C/C++. Масиви структур.

Мета заняття Ввести поняття структури як складеного типу даних, що визначається програмістом. Розглянути формати визначення структурного шаблону, приклади ініціалізації структур та операцій над структурами.

Матеріально-технічне забезпечення та дидактичні засоби, ТЗН:
конспект лекції, опорний конспект

Час – 2 години.

План проведення лекції

Структура лекції	Відведений час	Методичні вказівки
1 Організаційна частина	5 хв.	Включає в себе: привітання, яке має на меті привернути увагу, забезпечити контакт лектора з аудиторією; визначення присутності студентів на занятті
2 Актуалізація опорних знань, перевірка вивченого матеріалу та мотивація навчальної діяльності студентів	10 хв.	Включає в себе: вступне зауваження, мотивацію актуальності теми, перевірку попереднього матеріалу, формулювання мети лекції, огляд головних питань теми
3 Основна частина (викладення навчальних питань лекції)	60 хв.	Головне місце приділяється викладенню основного змісту матеріалу теми, його аналізу, узагальненню висунутих положень. Для успіху лекції важливе значення має поділ матеріалу на розділи, основні питання
4 Заключна частина Домашнє завдання: (відповідно до робочої програми)	5 хв.	Включає в себе: узагальнення, теоретичні і фактичні висновки, закріплення вивченого на лекції матеріалу, виставлення оцінок за роботу на занятті

Література

- 1 Шилдт, Г. C++: базовий курс, 3-є видання.: Пер. з англ. [Текст] / Г. Шилдт. – М.: Видавничий дім "Вільямс", 2010. – 624 с.: іл.
- 2 Страуструп, Б. Мова програмування C/C++ [Текст] / Б. Страуструп. – М.: Бі-

ном, 2006. – 501 с.: іл.

3 Паппас, К.Х. Відлагодження в C/C++. Керівництво для розробників [Текст] / К.Х. Паппас, У.Х. Мюррей III. – М.: "Біном", 2009. – 452 с.

Навчальні матеріали лекції

1 Поняття структури

Структура – це об'єднання одного або декількох об'єктів, можливо різного типу, під одним ім'ям, яке є типом структури. В якості об'єктів можуть виступати змінні, масиви, вказівники і інші структури. Елементи структури, які називаються полями, обов'язково повинні мати різні імена. Всі поля однією структури у сукупності називаються записом.

Структури дозволяють трактувати групи пов'язаних між собою об'єктів не як множину окремих елементів, а як єдине ціле.

Елементи структури розташовуються в пам'яті ЕОМ у тому ж порядку у якому вони оголошуються.

Формати визначення структурного шаблону наступні:

1) `struct ім'я_типу // спосіб 1`

<pre>{ тип1 поле1; тип2 поле2; . . . типN полеN; };</pre>	}	<i>всі поля у сукупності називаються</i> записом
---	---	--

В даному способі оголошенні структури після закриваючої фігурної дужки "}" обов'язково ставиться крапка з комою ";".

Приклад.

```
struct Date //визначення структури
```

```

{
    int day;
    int month;
    int year;
};
Date birthday; //змінна типу Date

```

2) struct //спосіб 2

```

{
    тип1 поле1;
    тип2 поле2;
    . . .
} список ідентифікаторів;

```

Приклад.

```

struct
{
    int min;
    int sec;
    int msec;
} time_beg, time_end;

```

У першому випадку опис структури визначає новий тип, ім'я якого можна використовувати разом із стандартними типами.

У другому випадку опис структури служить визначенням змінних.

3) Структурний тип можна також задати за допомогою ключового слова typedef.

```

typedef struct //спосіб 3
{
    тип1 поле1;
    тип2 поле2;
    . . .
} тип_структури;

```

Приклад.

```
typedef struct {
    floar re;
    float im;
} complex;
complex a[100]; //масив з 100 комплексних чисел.
```

2 Ініціалізація структур

Ініціалізація структур може бути здійснена двома способами:

- привласнення значень елементам структури в процесі оголошення змінної, яка відноситься до типу структури;
- привласнення початкових значень елементам структури з використанням потоків введення / виведення `cin`, `cout` та функцій – `scanf()`, `printf()`.

Для ініціалізації структур в процесі її оголошення значення її полів перераховують у фігурних дужках.

Загальний вигляд ініціалізації структури в процесі оголошення.

```
struct тип_структури ім'я_змінної=
{значение_поля_1,
 значение_поля_2,
 . . . ,
 значение_поля_n};
```

Приклади:

```
1 struct Student
{
    char name[20];
    int kurs;
    float rating;
};
```

```
Student s={"Іванов",1,3.5};

2 struct
{ char name[20];
char title[30];
float rate;
} employee = {"Петров", "директор",10000};
```

3 Операції над структурами

Операція привласнення.

Для змінних одного і того ж структурного типу визначена операція привласнення. При цьому відбувається по елементне копіювання.

Приклад.

```
Student s;
...
Student ss=s;
```

Доступ до елементів структур.

Доступ до елементів структур забезпечується за допомогою оператора "." та уточнених імен.

Загальний вигляд доступу до елементів структури.

```
Ім'я_структури.ім'я_поля
```

Приклад.

```
struct employee
{ char name[20];
char title[30];
float rate;
};
employee emp1 = {"Петров", "директор",10000};
```

- empl.name - доступ до поля name (рядок "Петров");
- empl.rate - доступ до поля rate. Змінна цілого типу із значенням 10000.

3 Приклад програми обробки масиву структур

Постановка задачі:

Написати програму, яка буде вводити дані про 5-х студентів та визначати студента з максимальним рейтингом. Дані зберігати у масиві структур. Кожний запис повинен містити: П.І.Б. студента, групу, середній бал успішності (рейтинг).

Блок-схема алгоритму рішення задачі (дивись рисунок 3.1):

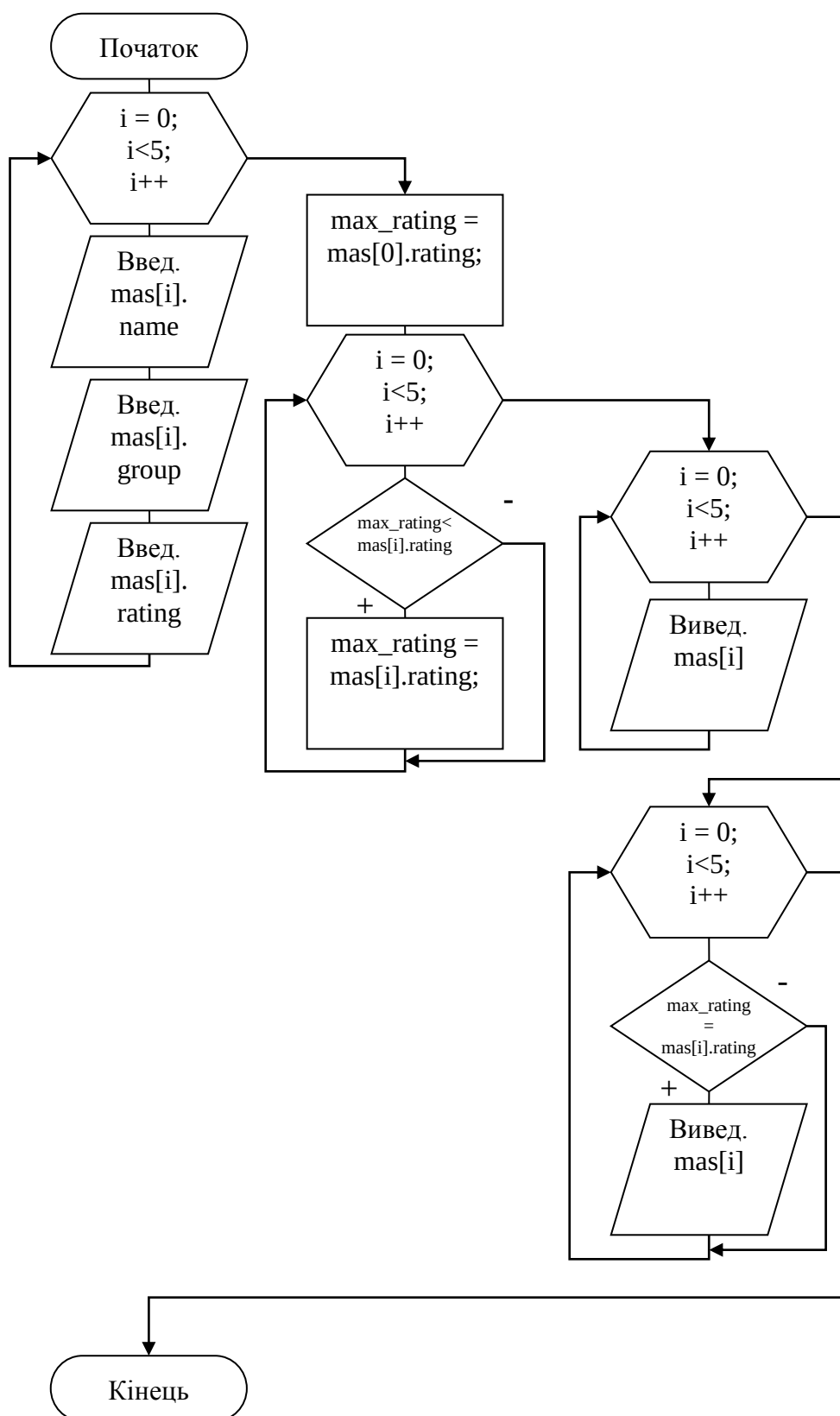


Рисунок 3.1. Блок-схема алгоритму рішення задачі:

Текст програми обробки масиву структур наведено у лістингу 3.1

Лістинг 3.1

```
#include <iostream.h>
#include <conio.h>
struct Student
{char name[30];
char group[10];
float rating;
};
int main()
{
Student mas[5];
float max_rating;
clrscr();
//введення значень масиву
for(int i=0;i<5;i++)
{
cout<<endl<<"Enter name:";
cin>>mas[i].name;
cout<<endl<<"Enter group:";
cin>>mas[i].group;
cout<<endl<<"Enter rating:";
cin>>mas[i].rating;
}
//визначення максимального рейтингу студента
max_rating=mas[0].rating;
for(int i=0;i<5;i++)
{
if(max_rating<mas[i].rating)
{
max_rating=mas[i].rating;
}
}

//виведення введених студентів на екран
clrscr();
for(int i=0;i<5;i++)
{
cout<<endl<<endl<<"Student # "<<i+1;
cout<<endl<<"\t Name:"<<mas[i].name;
cout<<endl<<"\t Group:"<<mas[i].group;
cout<<endl<<"\t Rating:"<<mas[i].rating;
}

//виведення студента (-ів) з максимальним рейтингом
cout<<"Students have a max rating:";
for(int i=0;i<5;i++)
{
if(max_rating==mas[i].rating)
{
```

```

cout<<endl<<endl<<"Student # "<<i+1;
cout<<endl<<"\t Name:"<<mas[i].name;
cout<<endl<<"\t Group:"<<mas[i].group;
cout<<endl<<"\t Rating:"<<mas[i].rating;
}
}
getch();
return 0;
}

```

Питання та завдання до контролю знань студентів

Питання для узагальнення та перевірки засвоєного матеріалу на лекції

- 1 Дати визначення структури як типу даних.
- 2 Які формати визначення структурного шаблону ви знаєте?
- 3 Як здійснюється ініціалізація структур?
- 4 Як виконується операція привласнення для структур?
- 5 Як здійснюється доступ до елементів структури?

Завдання для самоперевірки засвоєного матеріалу на лекції

- 1 Розробити конструкцію структури, що дозволить містити наступну інформацію про банківських клієнтів: Номер, Прізвище, Баланс.
- 2 Описати тип Tdate – структуру з полями цілого типу: Day (день), Month (місяць), Year (рік) і функцію логічного типу з параметром типу Tdate, що повертає значення true, якщо рік в даті високосний, і false – якщо ні.
- 3 Скласти програму , яка забезпечить введення з клавіатури даних мінімум про 5 студентів коледжу(ПІБ,рік народження, адреса, курс, група) та виведення на екран даних про студентів, які народились у 1997 році. В програмі повинні бути присутні такі поля запису : `rib`, `rik_n`, `adres`, `curs`, `grup`.

Лекція № 12-13

з навчальної дисципліни «Основи програмування та алгоритмічні мови»

Тема Файлова система в мові C. Основні функції роботи з файлами.

Мета заняття Ввести поняття моделі потоків введення-виведення в стилі C програм. Охарактеризувати роботу стандартних потоків бібліотеки stdio. Пояснити призначення основних функцій роботи з файлами.

Матеріально-технічне забезпечення та дидактичні засоби, ТЗН:
конспект лекції, опорний конспект

Час – 4 години

План проведення лекції

Структура лекції	Відведений час	Методичні вказівки
1 Організаційна частина	5 хв.	Включає в себе: привітання, яке має на меті привернути увагу, забезпечити контакт лектора з аудиторією; визначення присутності студентів на занятті
2 Актуалізація опорних знань, перевірка вивченого матеріалу та мотивація навчальної діяльності студентів	10 хв.	Включає в себе: вступне зауваження, мотивацію актуальності теми, перевірку попереднього матеріалу, формулювання мети лекції, огляд головних питань теми
3 Основна частина (викладення навчальних питань лекції)	60 хв.	Головне місце приділяється викладенню основного змісту матеріалу теми, його аналізу, узагальненню висунутих положень. Для успіху лекції важливе значення має поділ матеріалу на розділи, основні питання
4 Заклучна частина Домашнє завдання: (відповідно до робочої програми)	5 хв.	Включає в себе: узагальнення, теоретичні і фактичні висновки, закріплення вивченого на лекції матеріалу, виставлення оцінок за роботу на занятті

Література

- 1 Шилдт, Г. С++: базовий курс, 3-є видання.: Пер. з англ. [Текст] / Г. Шилдт. – М.: Видавничий дім "Вільямс", 2010. – 624 с.: іл.
- 2 Страуструп, Б. Мова програмування С/С++ [Текст] / Б. Страуструп. – М.: Біном, 2006. – 501 с.: іл.
- 3 Паппас, К.Х. Відлагодження в С/С++. Керівництво для розробників [Текст] / К.Х. Паппас, У.Х. Мюррей III. – М.: "Біном", 2009. – 452 с.

Навчальні матеріали лекції

1 Модель потоку введення – виведення

Більшість сучасних операційних систем доручають керування пристроями введення – виведення спеціалізованим драйверам, а потім програми звертаються до них за допомогою засобів бібліотеки введення - виведення, що забезпечують максимально однаковий зв'язок з різними джерелами й адресатами даних. Загалом, драйвери пристроїв глибоко впроваджені в операційну систему й недоступні для більшості користувачів, а бібліотечні засоби введення - виведення забезпечують абстракцію введення – виведення, так що програміст не повинен думати про пристрої і їхні драйвери.

Коли використовується така модель, вся вхідна й вихідна інформація може розглядатися як **потоки** байтів (символи), оброблювані засобами бібліотеки введення - виведення. Наша робота як програмістів зводиться до наступного: (див. рис. 1.1)

- 1 Налаштувати потоки введення - виведення на відповідні джерела й адресати даних.

- 2 Прочитати й записати їхні потоки.

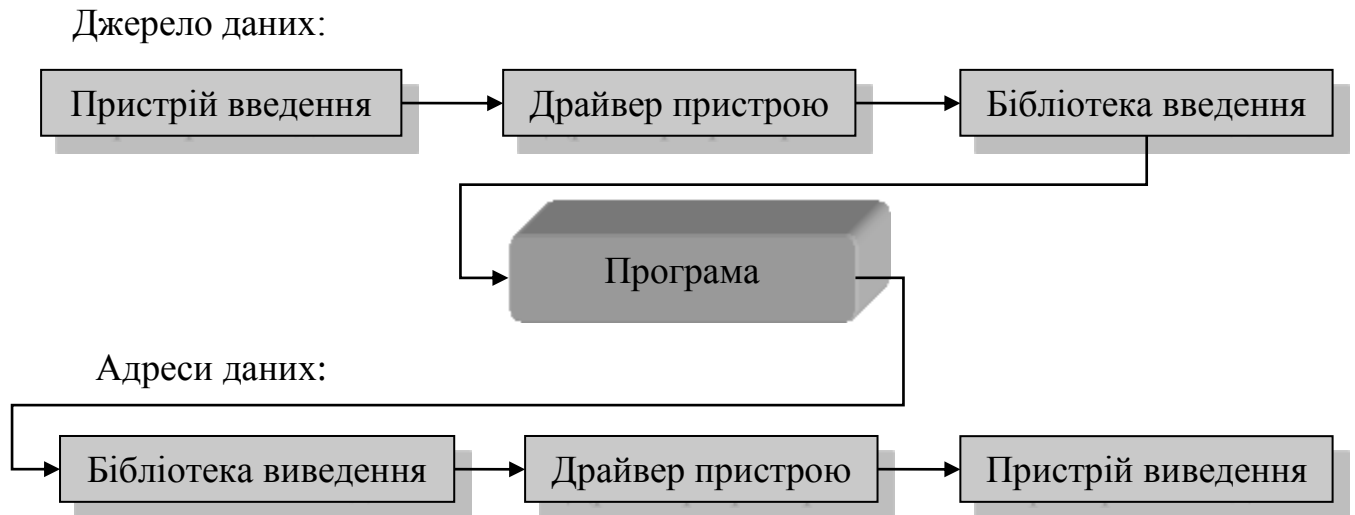


Рисунок 1.1. Модель потоків введення – виведення.

Як й інші сучасні мови, C++ не має вбудованих операцій введення - виведення. Вони винесені із самої мови в бібліотеку. Програми C++ можуть використати дві бібліотеки: стандартну бібліотеку введення – виведення `stdio`, успадковану з мови C, і більше нову бібліотеку `iostream`, розроблену спеціально для C++.

Обидві бібліотеки підтримують файлове введення й виведення. Бібліотека C складна й породжує помилки. Це важливо знати програмістам, що використовують успадковані програми на C.

Бібліотека C++ сприяють зниженню кількості помилок у програмах, але більше складна й громіздка. Часто одна й та ж дія може виконуватись двома способами. Щоб зрозуміти, як працює бібліотека C++ , потрібно знати про використання класів C++ , про спадкування, множинне спадкування й інші концепції, які поки не обговорювалися. От чому в даній лекції описуються лише базові засоби, що дозволяють зчитувати дані з файлів на диску й записувати їх у файл.

Стандартну бібліотеку введення – виведення `stdio`

Стандартна бібліотека мови C містить визначення типів `stdin` – стандартний потік введення, `stdout` – стандартний потік виведення та `stderr` – стандартний потік помилок.

Потік `stdin` робить наступне:

- 1 Одержує послідовність символів "звідкись" (наприклад, з консолі, з файлу, з оперативної пам'яті або від іншого комп'ютера).
- 2 Перетворює послідовності символів у значення різних типів.

Для взаємодії з операційною системою потік `stdin` використовує буфер. Буфер - це структура даних, яку потік `stdin` використає для зберігання інформації, отриманої від вас у ході взаємодії з операційною системою. При цьому буферизація може виявитися візуально помітною для користувача. Коли ви використаєте потік `stdin` пов'язаний із клавіатурою, усе, що ви введете, залишиться в буфері, поки ви не натиснете клавішу Enter (увести й перейти на новий рядок), і якщо ви передумали, то можете видалити символи за допомогою клавіші <Backspace> (поки не нажали клавішу <Enter>).

Потік `stdin` можна зобразити наступним чином (див. рис. 1.2):

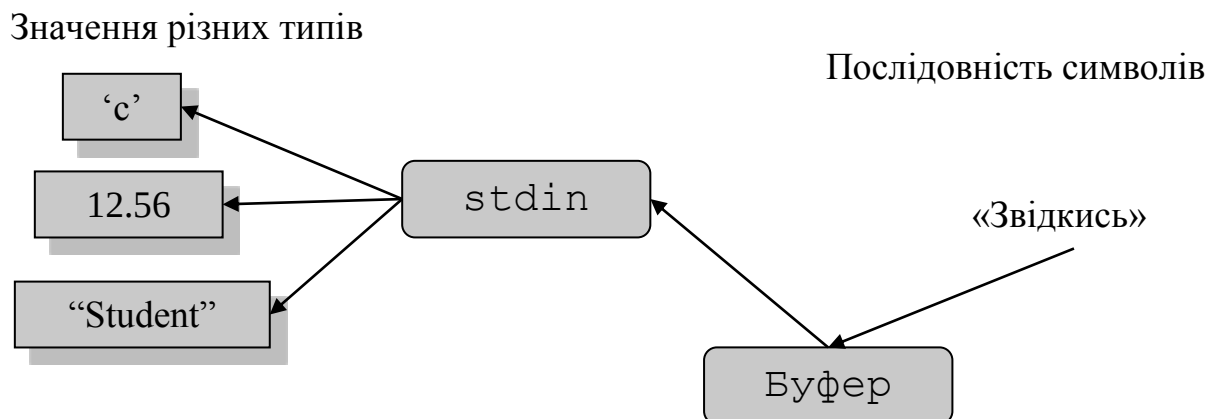


Рисунок 1.2. Модель потоку `stdin`.

Потік `stdout` робить наступне:

- 1 Перетворює значення різних типів у послідовності символів.
- 2 Посилає ці символи "кудись" (наприклад, на консоль, у файл, основну пам'ять або на інший комп'ютер).

Потік `stdout` можна зобразити наступним чином (див. рис. 1.3):

Значення різних типів

Послідовність символів

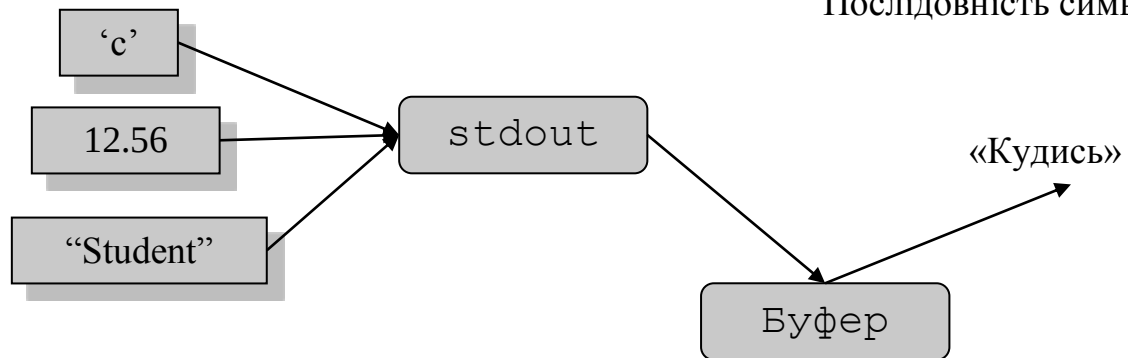


Рисунок 1.2. Модель потоку `stdout`.

2 Робота з файлами

Звичайно ми маємо набагато більше даних, чим здатна вмістити основна пам'ять нашого комп'ютера, тому більша частина інформації зберігається на дисках або інших засобах зберігання даних високої ємності. Такі пристрої також запобігають зникненню даних при вимиканні комп'ютера – такі дані є персистентними.

На самому нижньому рівні файл просто являє собою послідовність байтів, пронумерованих починаючи з нуля.

Файл має формат, інакше кажучи, набір правил, що визначають зміст байтів. Наприклад, якщо файл є текстовим, то перші чотири байти являють собою чотири символи. З іншого боку, якщо файл зберігає бінарне представлення цілих чисел, то перші чотири байти використовуються для бінарного представлення одного цілого числа. Формат стосовно файлів на диску грає ту ж роль, що й типи стосовно об'єктів в основній пам'яті. Ми можемо приписати бітам, записаним у файлі, певний зміст тоді й тільки тоді, коли відомий його формат.

При роботі з файлами потік `stdout` перетворює об'єкти, що зберігаються в основній пам'яті, у потоки байтів і записує їх на диск. Потік `stdin` діє на-

впаки; інакше кажучи, він зчитує потік байтів з диска й становить із них об'єкт (див. рис. 2.1).

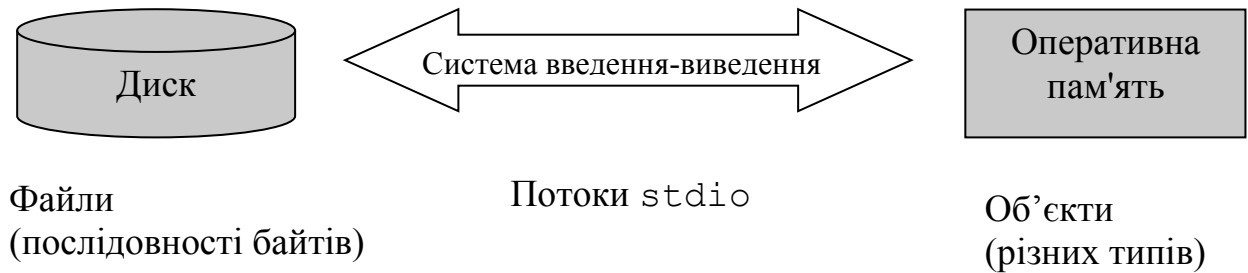
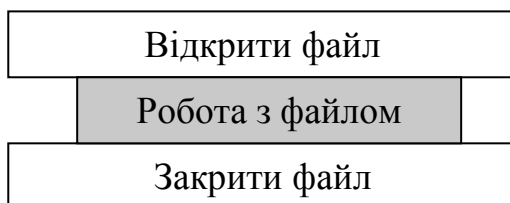


Рисунок 2.1. Файловий обмін даними.

Найчастіше ми припускаємо, що байти на диску є символами з звичайного набору символів. Це не завжди так, але, оскільки інші подання обробити нескладно, ми, як правило, будемо дотримуватися цього припущення. Крім того, будемо вважати, що всі файли перебувають на дисках (тобто на магнітних пристроях зберігання даних). І знов-таки це не завжди так (згадаєте про флеш-пам'ять), але на даному рівні програмування фактичний пристрій зберігання не має значення. Це одне з головних перевага абстракцій файлу й потоку.

Робота з файлами будується за принципом сандвіча:



Поняття "відкрити файл" означає "почати з ним роботу", зробити його активним і заблокувати обіг інших програм до цього файлу. При закритті файлу він звільняється (тепер з ним можуть працювати інші програми) і всі ваші зміни вносяться на диск.

Доступ до файлу здійснюється за допомогою покажчика. Покажчик на файл описується в такий спосіб:

```
FILE *fp;
```

Тип `FILE` - це структура, певна в `<stdio.h>` й утримує деяку інформацію про файл: наприклад, прапори стану файлу, розмір буфера, покажчик на буфер й ін. Описаний покажчик можна зв'язати з певним файлом у момент відкриття даного файлу. Це здійснюється за допомогою функції:

```
fp = fopen ("шлях до файлу", "тип доступу"),
```

що повертає покажчик на файл або `NULL` у випадку помилки.

Наприклад, у результаті виконання оператора

```
fp = fopen ("ex1.txt", "w"),
```

файл `ex1.txt` відкритий для запису, а в програмі на цей файл можна послати-ся за допомогою покажчика `fp` (тобто функція `fopen()` бере зовнішнє представлення файлу – його фізичне ім'я – і ставить йому у відповідність внутрішнє логічне ім'я, що далі й буде використатися в програмі).

Як тип доступу можуть бути зазначені наступні параметри:

"w" - існуючий файл відкривається для запису, при цьому його старий вміст затирається. Маркер файлу вказує на його початок. Якщо файл ще не існував, то він створюється (якщо це можливо);

"r" - існуючий файл відкривається для читання (з початку). Якщо файл не існує - це помилка;

"a" - файл відкривається для дозапису в кінець. Якщо файл ще не існує, він буде створений;

"w+" - відкривається порожній файл для читання й запису, при цьому його старий вміст затирається. Якщо файл ще не існував, то він створюється;

"r+" - існуючий файл відкривається для читання й запису. Якщо файл не існує - це помилка;

"a+" - файл відкривається для читання й додавання інформації в кінець.

Файл можна відкрити у двійковому (b) або текстовому (t) режимі. Ці символи записують у другому параметрі, наприклад, "rb" або "rt". Двійковий режим означає, що символи перекладу рядка й повернення каретки (0x13 й 0x10) обробляються точно так само, як й інші. У текстовому режимі ці символи

перетворюються в одиночний символ перекладу рядка. За замовчуванням файли відкриваються в текстовому режимі.

Приклад:

```
FILE *f = fopen ("d:\\cpp\\data", "rb+");
```

(відкривається файл d:\cpp\data для читання й запису у двійковому режимі й пов'язує з покажчиком f).

Показчик f використовується в подальших операціях з потоком (файлом).

По закінченні роботи з файлом він повинен бути закритий. Для цього використовується функція

```
fclose (показчик_файлу)
```

При закритті файлу очищаються буфери.

fclose(f) повертає 0 при успішному закритті й EOF в інших випадках:

```
if (fclose(f)!=0) printf("Ошибка при закрытии файла %s\n");
```

EOF - константа, що повідомляє про кінець файлу (ціле негативне) або про помилку при роботі з функціями читання.

Іноді програма не може відкрити файл. Якщо файл відкривається на читання, це можливо в наступних випадках:

- невірно задане ім'я файлу або файлу немає на диску;
- файл використовується іншою програмою й заблокований.

Якщо файл відкривається на запис, операція може закінчитися невдало, якщо:

- на диску немає місця;
- файл захищений від запису;
- невірно задане ім'я файлу (наприклад, воно містить дві крапки, знак питання й т.п.).

Якщо файл не вдалося відкрити, функція fopen повертає спеціальне нульове значення (нульовий показчик), що позначається NULL. Тому треба завжди перевірити правильність відкриття файлу, особливо в режимі читання. Якщо файл не є відкритий, треба вивести повідомлення про помилку й вийти із програми.

```

if ( fp == NULL )
{
printf("Ошибка открытия файла");
return;
}

```

Питання та завдання до контролю знань студентів

Питання для узагальнення та перевірки засвоєного матеріалу на лекції

- 1 Охарактеризуйте модель потоку введення – виведення, що використовується в мові C/C++.
- 2 Опишіть призначення та принцип дії основних функцій стандартної бібліотеки введення – виведення stdio
- 3 Назвати два види файлових потоків. Пояснити різницю між ними.
- 4 Сформулювати порядок роботи з файлами в C програмі.
- 5 Що таке файловий покажчик? Як описується файловий покажчик?
- 6 Сформулювати призначення та описати прототип функції fopen().
- 7 Перерахувати та пояснити відмінність між основними типами доступу до файлів.
- 8 Сформулювати призначення та описати прототип функції fclose().

Завдання для самоперевірки засвоєного матеріалу на лекції

- 1 Обрати правильну відповідь:
 - 1.1 Стандартний потік введення stdin бібліотеки stdio.h робить наступне:
 - А. одержує послідовність символів з консолі, з файлу, з оперативної пам'яті або від іншого комп'ютера та перетворює послідовності символів у значення різних типів;
 - В. тільки одержує послідовність символів з консолі, з файлу, з оперативної пам'яті або від іншого комп'ютера;

С. тільки перетворює послідовності символів у значення різних типів.

1.2 Стандартний потік введення `stdout` бібліо-теки `stdio.h` робить наступне:

- А. перетворює значення різних типів у послідовності символів;
- В. посилає символи вихідного потоку на консоль, у файл, в основну пам'ять або на інший комп'ютер;
- С. перетворює значення різних типів у послідовності символів та посилає ці сим-воли на консоль, у файл, в основну па-м'ять або на інший комп'ютер.

2 Написати функцію відкриття файлу `data.txt`:

- на читання в текстовому режим;
- на запис в текстовому режимі;
- як порожній на читання й запис в текстовому режимі;
- на запис в кінець в текстовому режимі.

Лекція № 14-15

з навчальної дисципліни «Основи програмування та алгоритмічні мови»

Тема інформації до файлу, читання з файлу. Створення та видалення файлу.
Пошук кінця або початку файлу.

Мета заняття Пояснити роботу функцій введення-виведення fread(), fwrite(), функції довільного доступу fseek(). Показати приклад роботи з файлом структур.

Матеріально-технічне забезпечення та дидактичні засоби, ТЗН:
конспект лекції, опорний конспект

Час – 4 години.

План проведення лекції

Структура лекції	Відведений час	Методичні вказівки
1 Організаційна частина	5 хв.	Включає в себе: привітання, яке має на меті привернути увагу, забезпечити контакт лектора з аудиторією; визначення присутності студентів на занятті
2 Актуалізація опорних знань, перевірка вивченого матеріалу та мотивація навчальної діяльності студентів	10 хв.	Включає в себе: вступне зауваження, мотивацію актуальності теми, перевірку попереднього матеріалу, формулювання мети лекції, огляд головних питань теми
3 Основна частина (викладення навчальних питань лекції)	60 хв.	Головне місце приділяється викладенню основного змісту матеріалу теми, його аналізу, узагальненню висунутих положень. Для успіху лекції важливе значення має поділ матеріалу на розділи, основні питання
4 Заключна частина Домашнє завдання: (відповідно до робочої програми)	5 хв.	Включає в себе: узагальнення, теоретичні і фактичні висновки, закріплення вивченого на лекції матеріалу, виставлення оцінок за роботу на занятті

Література

1 Шилдт, Г. С++: базовий курс, 3-е видання.: Пер. з англ. [Текст] / Г. Шилдт. –

М.: Видавничий дім "Вільямс", 2010. – 624 с.: іл.

2 Страуструп, Б. Мова програмування C/C++ [Текст] / Б. Страуструп. – М.: Біном, 2006. – 501 с.: іл.

3 Паппас, К.Х. Відлагодження в C/C++. Керівництво для розробників [Текст] / К.Х. Паппас, У.Х. Мюррей III. – М.: "Біном", 2009. – 452 с.

Навчальні матеріали лекції

1 Читання / запис двійкових даних. Функції fread(), fwrite()

Функція

```
size_t fread (void *buffer, size_t size, size_t count, FILE *stream);
```

зчитує count елементів довжиною size байтів в область, задану покажчиком buffer з потоку stream;

Функція

```
size_t fwrite (const void buffer, size_t size, size_t n, file stream);
```

записує n елементів довжиною size байтів з буфера, заданого покажчиком buffer у потік stream. Повертає число успішно записаних елементів звичайно воно = size, але при помилці воно може бути менше.

Тип size_t визначений у файлі stdio.h, означає «будь-який цілий». У байт, зайнятих цим «будь-яким цілим» можна одержати за допомогою функції sizeof(). Наприклад sizeof(int) - скільки байтів займає тип int.

Приклад:

```
char buf[256]; // масив символів
fwrite(buf, 256, 1, fp); // перед buf не ставиться &, тому що buf = &buf[0]
int k[10]; // масив цілих
fwrite(k, sizeof(int), 10, fp);
```

У прототипах цих функцій використовується void *buffer. Він діє у якості «пастки» для всіх типів даних.

2 Довільний доступ до фалів. Функції `fseek()`, `ftell()`

Функція

```
int fseek (file *f, long off, int org);
```

переміщає поточну позицію у файлі, пов'язаному з потоком `f` (відкритим за допомогою `fopen()`) на позицію `off`, відлічувану від значення `org`, що повинне бути дорівнює однієї з 3х констант, певних в `stdio.h`:

`seek_cur` - від поточної позиції,

`seek_end` - від кінця файлу,

`seek_set` - від початку файлу.

Функція повертає значення 0 при успішному завершенні та 1 – якщо помилка.

Функція

```
Long int ftell(file *f);
```

визначає поточну позицію у файлі `f` і повертає, як довге ціле позитивне число байтів, відлічуване від початку файлу. Якщо виникла помилка, функція поверне значення -1.

Як правило, довільний доступ потрібен лише при роботі з бінарними файлами. Оскільки текстові файли можуть вміщати керуючі символи, кількість байтів, на яке слід виконати переміщення файлового курсору, може не відповідати очікуваному символу. Функція `fseek()` коректно працює з текстовим файлом лише тоді, коли позиція курсора була заздалегідь визначена функцією `ftell()` та макросом `seek_set`.

Приклад програми посимвольного виводу текстового файлу відкритого в бінарному режимі у зворотному порядку (див. лістинг 2.1).

Лістинг 2.1. Посимвольна обробка файлу

```
#include <stdio.h>
#include <stdlib.h>
#define ctrl_z '0_32' // це ознака eof у текстовому файлі MS-DOS
```



```

#define n[50]

int main()
{char file[n]; // для введення імені оброблюваного файлу
char ch;
file *fp;
long c,k;
puts("Уведіть ім'я оброблюваного файлу");
gets(file);
if ((fp=fopen(file,"rb"))==null)
/* якщо помилка при відкритті файлу (тільки для читання в бінарно-
му режимі)*/
{printf("Не можу відкрити файл %s\n",file);
exit(1);
}
fseek(fp,0L,seek_end); /* перехід до кінця файлу. 0L - це довга
ціла константа «нуль»*/
r=tell(fp); // у цьому випадку це в байтів у файлі
for (c=1L; c<=r; c++)
    {fseek(fp,-c,seek_end); //рух назад
    ch=getc(fp); // читання чергового символу
    if ((ch!=ctrl_z)&&(ch!='\r'))
/* для MS-DOS якщо це не символ кінця файлу (^Z) і не символ '\r'
(див. самий початок теми) */
        putchar(ch); // вивід на екран поточного символу
    }
putchar('\n');
fclose(fp);
return 0;
}

```

3 Приклад програми роботи з файлом структур

Нехай файл містить наступну інформацію про співробітників патрульної служби ДАІ: ПІБ співробітника, стаж роботи, табельний номер зброї, стать, номер ланки.

Програма дозволяє:

- виконувати коректне відкриття файлу;
- додавання нових записів в файл;
- перегляд вмісту файлу;
- редагування запису.

Програма містить користувацьке меню.

Текст програми наведено в лістингу 3.1.

Лістинг 3.1. програма обробки файлу структур.

```
#include <iostream.h>
#include <conio.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>

// описание структуры данных

struct Record
{
    char fio[25]; //ФИО сотрудника
    int staz;
    int tab; // табельный номер оружия
    char gen; //пол: F - женский, M - мужской
    int zveno; //номер звена
};

// Прототипы функций

void Create_F(); //корректное открытие файла
void Intake(); //добавление данных
void View(); //просмотр файла
void Edit(); //редактирование записи

void main()
{
    int punkt;
    char c = 'y';

    // Организация меню программы

    while(c=='y')
    {
        clrscr();
        puts("Программа учета сотрудников патрульной службы ГАИ Жовт-
невого р-на");
        cout<<endl;
        puts("Главное меню:");
        puts("1. Корректное открытие файла");
        puts("2. Добавление сотрудников в файл");
        puts("3. Просмотр данных о сотрудниках");
        puts("4. Редактирование записей");
        puts("5. Выход из программы");
        puts("Выберите пункт меню: ");
        cin>>punkt;
        // cin.sync();
        switch(punkt)
        {
            case 1: Create_F(); break;
            case 2: Intake(); break;
            case 3: clrscr();View(); break;
            case 4: Edit(); break;
        }
    }
}
```

```

        case 5: c='n'; puts("Нажмите Enter завершить работу");
getch(); break;

    }

}

return;
}

```

//функция корректного открытия файла

```

void Create_F ()
{
    int i;
    FILE *fp;
    if((fp = fopen("d:\\pat.dan","r+b"))==NULL)
    { puts("Файл не существует. Хотите создать? (1 - да):");
      cin>>i;
      if(i==1)
      {
          fp = fopen("d:\\pat.dan","wb"); //создание нового файла взамен
старого
          fclose(fp);
          puts("Создан новый файл. Для возврата в главное меню нажмите
Enter");
          getch();
      }
      else
      {
          puts("Вы отказались создавать новый файл. Для возврата в
главное меню нажмите Enter ");
          getch();
      }
    }
    else
    {
        puts("Файл существует! Вы хотите удалить старый файл и создать
новый? (1 - да): ");
        cin>>i;
        if(i==1)
        {
            fclose(fp);
            fp = fopen("d:\\pat.dat","wb"); //создание нового файла взамен
старого
            fclose(fp);
            puts("Создан новый файл. Для возврата в главное меню нажмите
Enter ");
            getch();
        }
        else
        {
            puts("Вы отказались создавать новый файл. Для возврата в глав-
ное меню нажмите Enter ");

```

```

        fclose(fp);
        getch();
    }

}
return;
}

//функция добавления записей в файл

void Intake()
{
    clrscr();
    FILE * fp;
    Record PATR;
    char c='y';

    if((fp = fopen("d:\pat.dan", "a+b"))==NULL)
    {
        fputs("Ошибка открытия файла pat.dan", stderr);
        puts("Для возврата в главное меню нажмите Enter");
        getch();
        return;
    }
    else
    {
        while (c=='y')
        {
            cout<<endl<<"Введите данные о  сотруднике\n";
            cout<<endl<<"ФИО сотрудника:  ";
            cin>>PATR.fio;
            cout<<endl<<"Стаж работы:  ";
            cin>>PATR.staz;
            cout<<endl<<"Табельный номер:  ";
            cin>>PATR.tab;
            cout<<endl<<"Пол сотрудника:  ";
            cin>>PATR.gen;
            cout<<endl<<"Номер звена:  ";
            cin>>PATR.zveno;

            c=fwrite(&PATR, sizeof(PATR), 1, fp);
            cout<<endl<<"Продолжить ввод? (y/n):  ";
            cin.sync();
            cin>>c;
            getch();
        }
        cout<<endl<<"Для возврата в главное меню нажмите Enter  ";
        getch();
        fclose(fp);
        return;
    }
}

//функция просмотра содержимого файла

```

```

void View()
{

int i;
FILE * fp;
Record PATR;
if((fp = fopen("pat.dan", "r+b"))==NULL)
    { fputs("Ошибка открытия файла patrol.dat", stderr);
      puts("Для возврата в главное меню нажмите любую клави-
шу.Enter");
      getch(); return;
    }
else
    {
rewind(fp); //установка маркера в начало файла
i = 0;
while(fread(&PATR, sizeof(PATR), 1, fp)==1) i++;
if(i==0)
    { puts("Файл pat.dan пуст!");
      fclose(fp);
      puts("Для возврата в главное меню нажмите любую клави-
шу.Enter..");
      getch(); return;
    }
else
    {
puts("Текущее содержимое файла pat.dan");
cout<<"Сотрудников в файле: "<<i<<endl;
rewind(fp);
while (!(feof(fp)))
{ int n=1;
while(!(feof(fp))&&(n<=18))
{
if (fread(&PATR, sizeof(PATR), 1, fp)==1)
{
cout<<endl<<n<<"          "<<PATR.fio<<"          "<<PATR.staz<<"
"<<PATR.tab<<" "<< PATR.gen<<" "<< PATR.zveno<<endl;
n++;
}
}

getch();
}
cout<<endl<<"Для возврата нажмите Enter "<<endl;

getch();
fclose(fp); return ;

}
}
}

//функция редактирования записи

```

```

void Edit()
{
    clrscr();
    int i, punkt;
    FILE * fp;
    Record PATR;
    View();
    cout<<"Выберите номер записи"<<endl;
    cin>>i;
    clrscr();
    if((fp = fopen("d:\pat.dan", "r+b"))==NULL)
    {
        fputs("Ошибка открытия файла patrol.dat", stderr);
        puts("Для возврата в главное меню нажмите Enter");
        getch();
        return;
    }
    else
    {
        fseek(fp, (i-1)*sizeof(PATR), SEEK_SET);
        if (fread(&PATR, sizeof(PATR), 1, fp)==1)
        {
            cout<<endl<<PATR.fio<<"    "<<PATR.staz<<"    "<<PATR.tab<<"    "<<
PATR.gen<<"    "<< PATR.zveno<<endl;
            cout<<endl;
            cout<<"Выберите поле редактирования"<<endl;
            cout<<"1. ФИО сотрудника"<<endl;
            cout<<"2. Стаж работы"<<endl;
            cout<<"3. Табельный номер"<<endl;
            cout<<"4. Пол"<<endl;
            cout<<"5. Номер звена"<<endl;
            puts("Выберите пункт меню: ");
            cin>>punkt;
            cout<<endl;
            switch(punkt)
            {
                case 1: cin>>PATR.fio; break;
                case 2: cin>>PATR.staz; break;
                case 3: cin>>PATR.tab; break;
                case 4: cin>>PATR.gen; break;
                case 5: cin>>PATR.zveno; break;
            }
            fseek(fp, (i-1)*sizeof(PATR), SEEK_SET);
            fwrite(&PATR, sizeof(PATR), 1, fp);
        }
        else
        {
            cout<<"Ошибка выбора номера записи"<<endl;
        }
        cout<<endl<<"Для возврата нажмите Enter "<<endl;

        getch();
        fclose(fp); return ;
    }
}

```

}

Питання та завдання до контролю знань студентів

Питання для узагальнення та перевірки засвоєного матеріалу на лекції

- 1 Пояснити роботу функції `fread()` блокового зчитування з бінарного файлу.
- 2 Пояснити роботу функції `fwrite()` блокового запису в файл.
- 3 Пояснити роботу функції `fseek()` довільного доступу до даних бінарного файлу.
- 4 Пояснити роботу функції `ftell()`.

Завдання для самоперевірки засвоєного матеріалу на лекції

- 1 Дані про книги зберегти у файлі `book.dat`. Якщо файл уже існує, то програма відображає його вміст, а потім дозволяє додати до нього дані про нові книги. Уведіть дані про нові книги вводяться в масив структур `bibl`, а потім весь масив (потрібне в записів без обліку того, що вже в ньому було) записується у файл. Вивід повного переліку книг походить із масиву структур, а не з файлу.

Лекція № 16

з навчальної дисципліни «Основи програмування та алгоритмічні мови»

Тема Обробка виключних ситуацій. Загальні механізми обробки виключних ситуацій. Блок try, catch. Синтаксис і семантика генерації виключень.
Синтаксис і семантика обробки виключень

Мета заняття Ознайомити студентів з поняттям виключної ситуації, механізмами їх обробки, блоком обробки виключених ситуацій try-catch, продемонструвати приклад реалізації програми з обробкою виключних ситуацій

Матеріально-технічне забезпечення та дидактичні засоби, ТЗН:
конспект лекцій, опорний конспект

Час – 2 годин.

План проведення лекції

Структура лекції	Відведений час	Методичні вказівки
1 Організаційна частина	5 хв.	Включає в себе: привітання, яке має на меті привернути увагу, забезпечити контакт лектора з аудиторією; визначення присутності студентів на занятті
2 Актуалізація опорних знань, перевірка вивченого матеріалу та мотивація навчальної діяльності студентів	10 хв.	Включає в себе: вступне зауваження, мотивацію актуальності теми, перевірку попереднього матеріалу, формулювання мети лекції, огляд головних питань теми
3 Основна частина (викладення навчальних питань лекції)	60 хв.	Головне місце приділяється викладенню основного змісту матеріалу теми, його аналізу, узагальненню висунутих положень. Для успіху лекції важливе значення має поділ матеріалу на розділи, основні питання
4 Заключна частина Домашнє завдання: (відповідно до робочої програми)	5 хв.	Включає в себе: узагальнення, теоретичні і фактичні висновки, закріплення вивченого на лекції матеріалу, виставлення оцінок за роботу на занятті

Література

1 Шилдт, Г. С++: базовий курс, 3-е видання.: Пер. з англ. [Текст] / Г. Шилдт. – М.: Видавничий дім "Вільямс", 2010. – 624 с.: іл.

- 2 Страуструп, Б. Мова програмування C/C++ [Текст] / Б. Страуструп. – М.: Біном, 2006. – 501 с.: іл.
- 3 Паппас, К.Х. Відлагодження в C/C++. Керівництво для розробників [Текст] / К.Х. Паппас, У.Х. Мюррей III. – М.: "Біном", 2009. – 452 с.

Навчальні матеріали лекції

ВСТУП

У ході виконання програми можуть виникнути різні помилки. Вони можуть бути пов'язані з неправильним програмуванням (наприклад, вихід індексу масиву за межі припустимого чи переповнення пам'яті), а іноді їх причина не залежить від програміста (наприклад, розрив зв'язку при мережевому з'єднанні). У кожній з цих ситуацій реакція програми непередбачена. Іноді вона завершує виконання, і лише після закінчення деякого інтервалу часу починають з'являтися наслідки помилки, а частіше програма негайно припиняє роботу, піддаючи ризику дані, що знаходяться в пам'яті чи у файлі. Якщо не передбачити кероване завершення роботи, використовуючи обробку виключних ситуацій, результати можуть виявитися неприємними.

В подальшому будемо називати виключною Ця задача покладається на оброблювача виняткової ситуації.

1 Загальні поняття виключних ситуацій

Виключна ситуація – будь-яка подія, що вимагає особливої обробки. При цьому зовсім неважливо, чи є ця подія фатальною чи простою помилкою. Перевірка умов, що описують виключну ситуацію, і реакція на її виникнення називається обробкою виняткової ситуації. Використовуючи C++-підсистему обробки виключних ситуацій, такі події (помилки) можна використовувати програмісту в своїх цілях (наприклад, для їх усунення). При їх виникненні під час роботи програми автоматично викликається обробник виключень.

Керування C++-механізмом обробки виключень виконується з використанням блоку `try-catch` та оператора `throw`.

Програмні інструкції (в тому числі функції, які викликаються з `try-catch` блока, також перевіряються), які необхідно проконтролювати на предмет виключень, поміщуються в `try`-блок. Якщо виключення (тобто помилка) виникає у цьому блоці, воно дає про себе знати генерацією визначеного роду інформації за допомогою оператора `throw`. Це згенероване виключення може бути перехоплене програмним шляхом за допомогою `catch`-блока і оброблено відповідним чином.

Загальний вигляд блоку `try-catch`:

```
try
{
    //try-блок (блок коду, який повинен перевірятися
    //на предмет помилок)
}
catch (type1 arg)
{
    //catch-блок (обробник виключень типу type1)
}
catch (type2 arg)
{
    //catch-блок (обробник виключень типу type2)
}
catch (typeN arg)
{
    //catch-блок (обробник виключень типу typeN)
}
```

де *type* – один з програмних типів даних (наприклад, `int`, `float`, `char`, `double`, `bool`), об'єкт класу, літерал або користувацький тип;

arg – ім'я змінної для передачі значення виключення, яке формується за правилом ідентифікаторів.

Блок `try` повинен вміщувати код, який повинен бути перевірений на предмет виникнення помилок. Цей блок може включати або декілька інструкцій деякої функції, або охоплювати весь код функції `main()`.

Після генерації виключення перехоплюється відповідним блоком `catch`, який виконує його обробку. З одним блоком `try` може бути пов'язаний не один, а декілька `catch`-блоків. Який саме з них буде виконуватись буде визначатися типом виключення. Тобто буде виконаний той блок `catch`, тип виключення якого (тип даних заданий параметром *type*) співпадає з типом згенерованого виключення, а всі інші блоки будуть проігноровані. Після перехвату виключення параметр `arg` прийме його значення.

Загальний вигляд оператора `throw`:

```
throw exception;
```

За допомогою елемента *exception* задається виключення, згенероване оператором `throw`. Якщо це виключення повинне бути перехоплене, то оператор `throw` повинен бути виконаний в самому блоці `try`, або в будь-якій функції, яка викликається з нього.

Якщо в програмі забезпечується генерація виключення, для якого не передбачено відповідного `catch`-блоку, програма буде аварійно завершена, тому програміст повинен передбачити перехоплення всіх можливих виключень в програмі.

Приклад програми обробки виключених ситуацій засобами мови C++

```
#include <iostream.h>
#include <conio.h>

int main()
{
    clrscr();
    cout << endl << "ПОЧАТОК";
    try //початок try-блоку
    {
        cout << endl << "У блоці try";
        throw 99; //генерування виключення (із значенням 99)
```

```

        cout << "Ця інструкція не буде виконана";
    }
    catch (int i) //перехоплення виключення
    {
        cout << "Перехоплення виключення. Його значення дорівнює: ";
        cout << i << endl;
    }
    cout << "КІНЕЦЬ";

    getch();
    return 0;
}

```

Результат виконання програми:

ПОЧАТОК
 У блоці try
 Перехоплення виключення. Його значення дорівнює: 99
 КІНЕЦЬ

У прикладі, розглянуто вище, блок try вміщує три інструкції, а блок catch(int i) призначений для обробки виключення цілочисельного типу. У блоці try виконуються тільки 2 інструкції з трьох: cout та throw. Після генерації виключення керування передається блоку catch, при цьому виконання try-блоку припиняється.

Необхідно пам'ятати, що блок catch не викликається, а з нього просто продовжується виконання програми після генерації виключення.

Тому cout-інструкція, яка йде після throw-інструкції, ніколи не виконається.

Зазвичай при виконанні catch-блоку виконується спроба виправити помилку шляхом виконання відповідних дій. Якщо помилку можна виправити, то після виконання catch-блоку керування програмою передається інструкції, яка слідує за цим блоком. В протилежному випадку виконання програми повинно бути перервано.

Приклад.

Запитати у користувача розмір масиву. Для виконання завдання використовувати виключні ситуації та оператори для роботи з ними.

Знайти максимум парних елементів масиву.

```
#include <iostream.h>

int main()
{
    try
    {
        clrscr();
        int *iMas;
        int iAmount;
        cout << "Please enter an amount of elements: ";
        //введення кількості елементів в масиві
        cin >> iAmount;
        //виділення пам'яті під масив, розміром amount елементів
        iMas = new int[iAmount];
        cout << endl << "Please enter elements:" << endl;
        //введення елементів масиву
        for (int i = 0; i < iAmount; i++)
        {
            cout << "Enter " << i + 1 << " element: ";
            //введення i-го елемента масиву
            cin >> iMas[i];
        }
        clrscr();
        int iMax = 1;
        for (int i = 0; i < iAmount; i++)
        {
            cout << endl << i + 1 << " element of array = " << iMas[i];

            //якщо елемент масиву парний і не дорівнює 0
            if (iMas[i]%2==0 && iMas[i]!=0)
            {
                iMax = iMas[i];
            }
        }
        for (int i = 0; i < iAmount; i++)
        {
            //якщо i-елемент масиву парний, не дорівнює 0 та i-елемент
            //масиву менше за iMax
            if (iMas[i] % 2 == 0 && iMas[i] != 0 && iMas[i] > iMax)
            {
                //привласнюємо iMax значення нового максимуму
                iMax = iMas[i];
            }
        }
        //якщо за всю роботу програми значення iMax не змінилось
        //то максимум не було знайдено
    }
}
```

```

if (iMax == 1)
{
    //генеруємо виключення з типом char*
    throw "A maximum hasn't been found";
}

//виведення максимум з парних елементів масиву
cout << endl << endl<< "Maximum of element's array = " << iMax;
}
catch (char* cMessage)
{
    //виведення перехопленого виключення
    cout << endl << endl<< cMessage << endl;
}
getch();
return 0;
}

```

Блок-схема алгоритму рішення зазначеного прикладу дивись на
 рисунку 1.1.

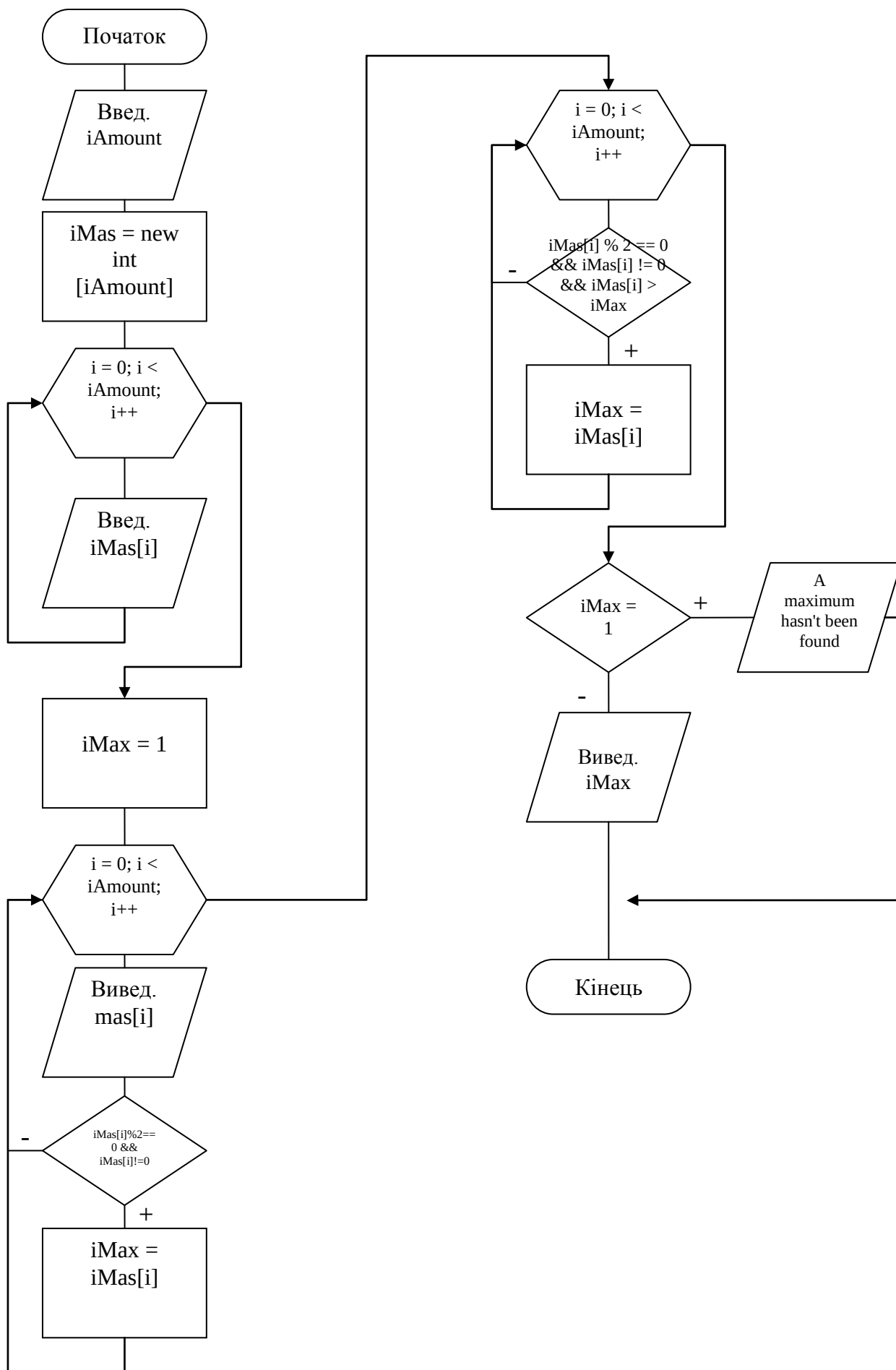


Рисунок 1.1 – Блок-схема алгоритму рішення задачі

ВИСНОВКИ

До основних тверджень лекції можна віднести наступні:

- виключною ситуацією називається будь-яка подія, що вимагає особливої обробки;
- перевірка умов, що описують виключну ситуацію, і реакція на її виникнення називається обробкою виключної ситуації. Ця задача покладається на обробника виключної ситуації;
- обробка виключних ситуацій у мові C++ є об'єктно-орієнтованою. Це значить, що виключна ситуація є об'єктом, що генерується при виникненні незвичайних умов, передбачених програмістом, і передається обробнику, що його перехоплює. Об'єктом, що описує природу виключної ситуації, може бути будь-яка сутність – літерал, рядок, об'єкт класу, число і т.д.;
- основі обробки виключних ситуацій лежать три ключових слова: `try`, `catch` і `throw`. Блок `try` визначає область програми, де може виникнути виключна ситуація, оператор `throw` генерує об'єкт-виключну ситуацію, а блок `catch` перехоплює цей об'єкт;
- розмір блоку `try` необмежений. У нього можна включити як один оператор, так і цілу програму;
- кожен блок `catch` відповідає окремому типу виключної ситуації. Програма сама визначає, який з них виконати. У цьому випадку інші блоки `catch` не виконуються. Кожен блок `catch` має аргумент, що приймає визначене значення. Цей аргумент може бути об'єктом будь-якого типу. Якщо програма виконана правильно і у блоці `try` не виникло жодної виключної ситуації, усі блоки `catch` будуть проігноровані;
- якщо в програмі виникла подія, яку програміст вважає небажаною, оператор `throw` генерує виключну ситуацію. Для цього оператор `throw`

повинний знаходитися усередині блоку `try` або усередині функції, яка викликається всередині блоку `try`;

- блоки `try` і `catch` нерозривні. Не можна помістити блок `try` у функцію, залишивши блок `catch` у функції `main()`. Необхідно або обробити виключну ситуацію усередині функції, або перенести обробку в модуль виклику. У першому випадку функція, завершивши обробку, повертає визначене її специфікацією значення, а в другому – виключну ситуацію.

Контрольні питання:

- 1 Дайте визначення що таке виключна ситуація?
- 2 Які блоки та оператори використовуються при обробці виключних ситуацій?
- 3 Опишіть призначення блоку `try`.
- 4 Опишіть призначення блоку `catch`.
- 5 Опишіть призначення оператора `throw`.
- 6 Наведіть загальний вигляд блоку `try`.
- 7 Наведіть загальний вигляд блоку `catch`.
- 8 Наведіть загальний вигляд оператора `throw`.
- 9 Чи може з одним блоком `try` бути пов'язано декілька блоків `catch`?
- 10 Чи виконуються інструкції програми, які слідують після оператора `throw`.