Міністерство освіти і науки України

Тернопільський національний технічний університет

імені Івана ПУЛЮЯ

кафедра програмної інженерії

ОСНОВИ ПРОГРАМУВАННЯ

Курс лекцій

для студентів першого рівня вищої освіти

за спеціальністю No 121 Інженерія програмного забезпечення

Тернопіль 2018

Основи програмування. Курс лекцій для студентів першого рівня вищої освіти за спеціальністю No 121 Інженерія програмного забезпечення/ Уклад.: М.Р. Петрик, О.Ю.Петрик - Тернопіль: ТНТУ 2018- 64 с.

Курс лекцій є частиною методичного забезпечення курсу „Основи програмування" і призначений для студентів 1 року навчання за спеціальністю 121 Інженерія програмного забезпечення першого рівня вищої освіти (бакалавр). Складений з урахуванням робочої програми навчальної дисципліни.

Укладачі: М.Р. Петрик, О.Ю.Петрик

Відповідальний за випуск: М.Р.Петрик

Рецензент: к.т.н., доцент О.В.Мацюк

Розглянуто на засіданні кафедри програмної інженерії, протокол №2 від 3.09.2018р.

Схвалено на засіданні методичної ради факультету комп'ютерно-інформаційних систем і програмної інженерії Тернопільського національного технічного університету імені Івана Пулюя, протокол №2 від 24 жовтня 2018р.

ЗМІСТ

Елементи мови

- Історія виникнення і розвитку мови C/C++. Поняття програми. Структура програми мовою C++: поняття препроцесора, функції. Структура функцій.
- Алфавіт, слова. Правила побудови ідентифікаторів. Типи даних. Область видимості.
- Поняття виразу. Арифметичні операції. Класифікація операцій. Пріоритет операцій.
- Оператори присвоєння. Інкремент і декремент. Оператор sizeof.

Структура програми мовою C++ та типи даних

Типова структура програми:

1) Підключення заголовочних файлів бібліотек (директиви препроцесора, функції, константи і інше);

2) Оголошення констант, прототипів функцій, класів, глобальних змінних;

3) Заголовок головної функції;

4) Визначення головної функції – блок, який містить:

- оголошення локальних змінних; констант; їх ініціалізація (присвоєння початкових значень);
- введення вхідних даних (діалог з користувачем);
- опрацювання звертання до функцій, обчислення виразів, виконання операторів;
- виведення результатів;
- повернення коду завершення головної функції.

5) Визначення користувацьких допоміжних функцій; для кожної функції після заголовка розміщується блок, який містить:

- оголошення локальних змінних і їх ініціалізацію;
- опрацювання формальних параметрів з використанням локальних змінних;
- повернення результату.

Препроцесор входить у склад системи програмування C++ і забезпечує опрацювання вхідного тексту програми перед її компіляцією. В ході препроцесорної обробки програми виконуються такі дії:

- підстановка макророзширень замість імен макросів і їх аргументів;
- у вхідний текст програми включаються файли, які містять допоміжну інформацію (описи об'єктів, класів, прототипи функцій і ін.);
- виконується умовна компіляція програми – визначення остаточного вигляду вхідного тексту програми шляхом залишення або виключення окремих її фрагментів.

Завдання препроцесору описується з допомогою директив препроцесора, які починаються зі знаку #, вказаного в першій позиції у рядку.

Функції використовуються для спрощення процесу розробки програм у випадках, коли аналогічні перетворення над різними даними необхідно виконувати в декількох місцях програми.

При визначенні функції рекомендують:

- оголосити і проініціалізувати вхідні і вихідні змінні;
- проміжні змінні оголошувати і ініціалізовувати на початку блоків або перед операторами, в яких вони використовуються.

Структуру функції буде розглянуто пізніше.

Програма на C++ складається з функцій, причому одна з них - main(). Програма завжди починає виконуватись з першого оператора цієї функції. У круглих дужках у загальному випадку міститься інформація, яка передається функції.

При написанні програм на C++ використовуються символи, які складають її алфавіт.

Основними елементами мови є символи (алфавіт мови), лексеми, вирази і оператори.

Алфавіт мови C++

В алфавіт мови C++ входять:

- латинські букви (a-z) – великі та малі;
- цифри;
- спеціальні символи + - * / < > = ! & \ ~ ’ @ # $ % ^ ? _ : ; , . ( )[ ]{ }” . Використовуються для позначення операцій і запису виразів;
- пропуски (знак пропуску, табуляції, переходу на інший рядок).

З окремих символів формуються лексеми.

Лексема - це мінімальна одиниця мови, яка має самостійний зміст.

До лексем відносяться:

- ідентифікатори;
- ключові (зарезервовані) слова;
- знаки операцій;
- константи;
- розділювачі (дужки, крапка, кома, пропуски).

Імена об’єктів - констант, змінних, типів даних, функцій і ряд інших називаються ідентифікаторами.

Ідентифікатори вибираються з урахуванням наступних правил:

- використовуються букви латинського алфавіту, символ підкреслення і цифри (0…9);
- починаються з букви латинського алфавіту (a…z, A…Z) або знаку підкреслення;
- розрізняються великі і малі букви;
- мають довільну довжину (в Borland C++5.02 довжина ідентифікатора може встановлюватися при настроюванні середовища і мати не більше 32 розрядів);
- не співпадають з ключовими словами мови і іменами стандартних функцій з бібліотеки.

Коментарі

Кожен рядок коментарів починається з //. Підтримуються також застарілі формати:

```
/* старий стиль коментарів */
```

або

```
/* багаторядковий
коментар
*/
```

Коментарі можуть бути розміщені в довільному місці програми, але гарний стиль програмування вимагає їх розміщення в окремих рядках.

Пропуски, знаки табуляції і переходу на інший рядок ігноруються. Це дозволяє записувати текст програми в довільній формі і рядки починати з довільної позиції, один оператор може бути записаний в декількох рядках.

Типи даних

Тип даних - це множина значень, яких можуть набувати об’єкти програми, та сукупність операцій, допустимих над цими значеннями.

Розрізняють поняття „специфікатор” типу даних і „модифікатор” типу

Специфікатори типів	Розмір (байт)
char (символьний)	1
bool (логічний)	1
int (цілочисельний)	4
float (дійсний)	4
double (дійсний подвійної точності)	8
void (порожній, для нейтралізації значення об'єкта)	-

Модифікатори: signed (знаковий), unsigned (беззнаковий), short (короткий), long (довгий). Після модифікатора слідує специфікатор. Якщо специфікатор опущений - за замовчуванням int.

Область видимості змінних

Змінна – це поіменована область пам'яті, в якій зберігаються дані визначеного типу. Змінна визначається ідентифікатором і значенням. Змінні оголошуються в довільному місці програми до моменту їх використання.

Оголошення змінних всередині коду C++ затрудняє їх пошук, але доцільніше оголошувати змінну тоді, коли вона вперше потрібна, а не на початку програми.

Оператор опису змінних має загальний формат:

> *[<клас пам'яті>][<const>] <тип> <ім'я> [<ініціалізатор>];*

Клас пам'яті може приймати значення auto, extern, static, register. Модифікатор const показує, що значення змінної змінювати не можна. При оголошенні змінній можна присвоїти початкове значення (ініціалізувати). Ініціалізатор можна записувати у двох формах:

> *= <значення> або (<значення>).*

Константа має бути ініціалізована при оголошенні. В одному операторі можна описати декілька змінних, наприклад:

```
short int a=1;
const char c='c';
char s, sf='f', t(54);
float c=0.22, x(3), sum;
```

Оголошення змінної, крім типу і класу пам'яті, явно або по замовчуванню, задає її *область дії*. Клас пам'яті і область дії залежать не тільки від оголошення, але й від місця його розміщення в програмі.

Залежно від області дії змінна може бути локальною або глобальною. Якщо змінна оголошена всередині блоку (у фігурних дужках), вона називається *локальною.* Область її дії – від моменту описання до кінця блоку, включаючи всі вкладені блоки.

Якщо змінна оголошена поза межами будь-якого блоку, вона називається глобальною і областю її дії є файл, в якому вона оголошена.

Клас пам'яті визначає *час життя* і *область видимості* змінної. Час життя може бути постійним (протягом виконання програми) і тимчасовим (під час виконання блоку).

Областю видимості змінної називається частина тексту програми, з якої допустимий звичайний доступ до пов'язаної з ідентифікатором області пам'яті. Область видимості співпадає з областю дії, крім ситуації, коли у вкладеному блоці описана змінна з тим же ім'ям, що й глобальна. В цьому випадку зовнішня змінна у вкладеному блоці не видима, хоча цей блок і входить в область її дії. Але, якщо ця змінна глобальна, то до неї можна звернутись, використовуючи операцію доступу (::).

Наприклад,

```
int a;              //глобальна змінна a
int main()
{
int b;              //локальна автоматична змінна b
```

```
extern int x;        // зовнішня змінна x
static int c;        // локальна статистична змінна c
a=1;                 // присвоєння значення глобальній змінній a
int a   ;            // оголошення локальної змінної з тим же ім'ям
a=2;                 // присвоєння значення локальній змінній a
::a=3;               /* доступ до глобальної змінної, оскільки її
                        область видимості обмежена локальною
                     змінною з тим же ім'ям */
}
int x=4;         // ініціалізація зовнішньої змінної x
```

Операції і вирази

За кількістю операндів, які беруть участь в операції, операції поділяються на унарні, бінарні і тернарні. В мові C++ розрізняють наступні унарні операції:

Знак операції	Операція
-	арифметичне заперечення (заперечення і доповнення);
~	побітове логічне заперечення (доповнення);
!	логічне заперечення;
*	непряма адресація;
&	взяття адреси;
+	унарний плюс;
++	інкремент (збільшення на 1);
--	декремент (зменшення на 1);
sizeof	визначення розміру .

Унарні операції виконуються справа наліво.

Бінарні операції, список яких наведено нижче, виконуються зліва направо.

Знак операції	Операція	Група операцій
*	Множення	Мультиплікативні
/	Ділення	
%	Остача від ділення	
+	Додавання	Адитивні
-	Віднімання	
<<	Зсув вліво	Операції зсуву
>>	Зсув вправо	
<	Менше	Операції відношення
<=	Менше або рівне	
>=	Більше або рівне	
==	Рівне	
!=	Не рівне	
&	Порозрядне і	Порозрядні операції
|	Порозрядне або	
^	Порозрядне виключне або	

&&	Логічне і	Логічні операції
||	Логічне або	
,	Послідовне обчислення	Послідовного обчислення
=	Присвоєння	Операції присвоєння
*=	Множення з присвоєнням	
/=	Ділення з присвоєнням	
%=	Остача від ділення з присвоєнням	
-=	Обчислення з присвоєнням	
+=	Додавання з присвоєнням	
<<=	Зсув вліво з присвоєнням	
>>=	Зсув вправо присвоєнням	
&=	Порозрядне „і" з присвоєнням	
|=	Порозрядне „або" з присвоєнням	
^=	Порозрядне „виключне або" з присвоєнням	

Приведення типів при обчисленні виразів

При виконанні операцій відбувається автоматичне приведення типів операндів до спільного типу. Виконання приведення залежить від специфіки операцій і від типу операнда чи операндів, але в загальному випадку, при обчисленні виразів операнди будуть мати тип того операнда, що займає найбільший розмір пам'яті.

Приклад:

```
double        ft,sd;
unsigned char  ch;
unsigned long  in;
int           i;
sd=ft*(i+ch/in);        //double
```

Операція sizeof

За допомогою операції sizeof можна визначити розмір пам'яті, яка відповідає вказаному типу. Операція sizeof має наступний формат:

```
sizeof (вираз) ;
```

В якості виразу може бути використаний будь-який ідентифікатор або ім'я типу, крім void, а ідентифікатор не може бути іменем бітового поля чи ім'ям функції.

Якщо як вираз указане ім'я масиву, то результатом є розмір усього масиву (тобто добуток числа елементів на довжину типу), а не розмір вказівника, що відповідає ідентифікатору масиву.

Коли sizeof застосовується до імені типу структури чи об'єднання, то результатом є сума розмірів типів елементів структури чи об'єднання.

Мультиплікативні операції

До цього класу операцій відносяться операції множення (*), ділення (/) і одержання остачі від ділення (%). Відзначимо, що типи операндів операцій множення і ділення можуть відрізнятися, і для них справедливі правила приведення типів. Типом результату є тип операндів після приведення.

Операція ділення (/) виконує ділення першого операнда на другий. Якщо дві цілі величини не діляться націло, то результат округляється до цілого.

При спробі ділення на нуль видається повідомлення про помилку.

```
int i=49, j=10,  n, m;
```

```
n = i/j;                    // n= 4
m = i/(-j);                 // m=-4
```

Операція (%) дає остачу від ділення першого операнда на другий. Операнди операції (%) повинні бути цілими змінними. Знак результату залежить тільки від знаку першого операнда.

```
int n = 49, m = 10, i, j, k, l;
i = n % m;                   //  9
j = n % (-m);                //  9
k = (-n) % m;                // -9
l = (-m) % (-n);             // -10
```

Операція послідовного обчислення

Операція послідовного обчислення позначається комою (,) і використовується для обчислення двох і більше виразів там, де за синтаксисом допустимий тільки один вираз, наприклад, for(int i=0, j=0; i<10; i++, j++).

Умовна операція

У мові С є одна тернарна операція - умовна операція, що має наступний формат:

```
Операнд_1 ? операнд_2 : операнд_3;
```

Операнд_1 повинен бути цілого, дійсного типу, чи вказівником. Якщо операнд_1 не дорівнює 0, то обчислюється операнд_2 і його значення є результатом операції. Якщо операнд_1 дорівнює 0, то обчислюється операнд_3 і його значення є результатом операції.

Приклад:

```
max = (d<=b) ? b : d;
/*Змінній max присвоюється  максимальне значення змінних d і b */
```

Операції збільшення і зменшення

Операції інкременту (++) і декременту (--) є унарними операціями присвоєння. Вони відповідно збільшують чи зменшують значення операнда на одиницю. Операнд може бути цілого, дійсного типу чи типу вказівник. Операнд цілого чи дійсного типу збільшується (зменшується) на одиницю. Тип результату відповідає типу операнда. Операнд адресного типу збільшується (зменшується) на розмір об'єкта, який він адресує. Допускається префіксна та постфіксна форми операцій збільшення (зменшення).

Якщо знак операції стоїть перед операндом (префіксна форма запису), то зміна операнда відбувається до його використання у виразі і результатом операції є збільшене (зменшене) значення операнда. У випадку, якщо знак операції стоїть після операнда (постфіксна форма запису), то операнд спочатку використовується для обчислення виразу, а потім відбувається зміна операнда.

Приклади:

```
int t=1, s=2, z, f;
z=(t++)*5;
```

Спочатку відбувається множення t*5, а потім збільшення t на 1. У результаті одержимо z=5, t=2.

```
f=(++s)/3;
```

Спочатку значення s збільшується на 1, а потім використовується в операції ділення. У результаті одержимо s=3, f=1.

У випадку, якщо операції збільшення і зменшення використовуються як самостійні оператори, префіксна і постфіксна форми запису стають еквівалентними.

```
z++; /* еквівалентне */ ++z;
```

Просте присвоєння

Операція звичайного присвоєння використовується для заміни значення лівого операнда значенням правого операнда. При присвоюванні відбувається приведення типу правого операнда до типу лівого операнда за правилами, згаданими раніше. Лівий операнд має бути змінною.

Приклад:

```
int t; char f;
long z; t=f+z;
```

Значення змінної f приведеться до типу long, обчислюється f+z, результат приведеться до типу int і потім присвоїться змінній t.

Складене присвоєння

Крім простого присвоєння є ціла група операцій присвоєння, що поєднують просте присвоєння з однією з бінарних операцій. Такі операції називаються складеними операціями присвоєння і мають вигляд:

(операнд_1) (бінарна операція) = (операнд_2).

Складене присвоєння за результатом еквівалентне наступному простому присвоюванню:

(операнд_1) = (операнд_1) (бінарна операція) (операнд_2).

Відзначимо, що вираз складеного присвоєння з погляду реалізації не еквівалентний простому присвоюванню, тому що в останньому операнд_1 обчислюється двічі.

Приклад:

```
double  arr[4]={ 2.0, 3.3, 5.2, 7.5 };
double  b=3.0;
b+=arr[2];      // b=b+arr[2]
arr[3]/=b+1;    // arr[3]=arr[3]/(b+1)
```

При другому присвоєнні використання складеного присвоєння дає виграш у часі виконання, тому що лівий операнд є індексним виразом.

Пріоритети операцій і порядок виконання обчислень

Пріоритет	Знак операції	Тип операції	Порядок виконання
1	- ~ ! * & ++ -- sizeof приведення типів	Унарні	Справа наліво
2	( ) [ ] . ->	Вираз	Зліва направо
3	* / %	Мультиплікативні	Зліва направо
4	+ -	Адитивні	
5	<< >>	Зсуву	
6	< > <= >=	Відношення	
7	== !=	Відношення (рівність)	
8	&	Порозрядне І	
9	^	Порозрядне виключні АБО	
10	|	Порозрядне АБО	
11	&&	Логічне І	
12	||	Логічне АБО	
13	? :	Умовна	
14	= *= /= %= += -= & = |= >>= <<= ^=	Просте і складене присвоєння	Справа наліво
15	,	Послідовне обчислення	Зліва направо

Варто не використовувати операції присвоєння змінної у виклику функції, якщо ця змінна бере участь у формуванні інших аргументів функції.

Організація введення-виведення в C/C++

- Форматне введення-виведення з допомогою функцій бібліотеки stdio.
- Введення-виведення символів і рядків.
- Поняття потокового введення-виведення з допомогою об'єктів cin, cout класу ios. Деякі маніпулятори форматування потоку виведення

Форматоване виведення

Функція printf() виконує форматований вивід на консоль. Прототип функції (бібліотека stdio.h) має загальний вигляд:

> *int printf (char *управляючий рядок, аргумент1, аргумент2, ...);*

Управляючий рядок містить два типи інформації: символи, які безпосередньо виводяться на екран, і команди формату (специфікатори формату) які визначають, як саме виводити аргументи. Команда формату починається з символу % і закінчується деяким символом, який задає перетворення. Між знаком % і символом перетворення можуть зустрічатись інші знаки у відповідності з наступним форматом

> *%[прапорець][ширина_поля][точність][F|N|h|L] символ_перетворення*

Символи перетворення наступні:

c	значенням аргумента є символ;
d або i	ціле десяткове число;
e	дійсне десяткове число у експоненціальній формі x.xx e+xx;
E	дійсне десяткове число у вигляді x.xx E+xx;
f	дійсне десяткове число з плаваючою крапкою xx.xxxx;
g	використовується як f чи e, і виключає вивід незначущих нулів;
o	вісімкове ціле число;
s	рядок символів;
u	беззнакове десяткове число (символи рядка виводяться до того часу, поки не зустрінеться символ кінця рядка чи не буде виведене число символів, задане точністю);
x	16-кове число з цифрами 0..9, a, b, c, d, e, f;
X	16-кове число з цифрами 0..9, A, B, C, D, E, F;
p	значенням аргументу є вказівник.

Компілятор не перевіряє ні кількість аргументів, ні тип даних. Тому за всі помилки форматування несе відповідальність програміст.

Необов'язкові параметри в специфікаціях перетворення:

- прапорець – задає правила вирівнювання, знаки + і -, десяткову точку, "хвостові" нулі і префікси для вісімкових і 16-кових значень;

  Прапорці для функції printf():

  - текст, що виводиться, вирівнюється ліворуч. Все залишене справа порожнє місце заповнюється пропусками. За замовчуванням текст вирівнюється праворуч;

  + виводиться знак + чи - перед числовим значенням.

- ширина поля – задає розмір поля виведення, в якому об'єкт виведення доповнюється символами пропуску або цифрами 0;
- точність - починається з крапки і визначає точність числа – для чисел з плаваючою крапкою або мінімальне значення цифр – для цілих чисел.

Один або декілька *модифікаторів* дозволяє вказати характеристики, пов'язані з розміром значення, що виводиться: F – (far - дальний вказівник), N – (near – ближній вказівник), h – значення типу short int, l - long, L – long double. Тип даних вибирається відповідним символом перетворення.

- Управляючі константи:

  \a – звуковий сигнал;

  \b – зміщення курсора вліво на одну позицію;

  \n – перехід на новий рядок;

  \r – повернення каретки на початок рядка;

  \t – горизонтальна табуляція;

  \\ - вивід символа «\»;

  \' - вивід символа «‘»;

  \'' - вивід символа «”»;

  \?- вивід символа «?».

Форматоване введення даних

Функція scanf() призначена для зчитування даних з клавіатури, форматування введених даних згідно формату, заданого в управляючому рядку, і присвоєння аргументам зі списку. Кожен аргумент має бути вказівником.

Прототип із файла stdio.h має вигляд

int scanf(“управляючий рядок”, аргумент 1, аргумент 2, ...);

Управляючий рядок містить три види символів: специфікатори формату, пропуски й інші символи. Команди або специфікатори формату починаються із символу % і перераховані нижче: %c - читання символу,

%d, %i - читання десяткового цілого,

%e - читання числа типу float,

%h - читання short int,

%o - читання вісімкового числа,

%s - читання рядка,

%x - читання 16-кового числа,

%p - читання вказівника,

%n - читання вказівника в збільшеному форматі.

Символ пропуску в управляючому рядку дає команду пропустити один або більше пропусків з потоку введення. Крім пропуску може сприйматися символ табуляції або нового рядка. Ненульовий символ указує на читання і відкидання (discard) цього символу.

Розділювачами між двома числами, що вводяться, є символи пропуску, табуляції або нового рядка. Знак * після % і перед кодом формату дає команду прочитати дані зазначеного типу, але не присвоювати це значення.

Так, scanf("%d%-c%d", &i, &j); -при введенні 50+20 присвоїть змінній і значення 50, змінній j - значення 20, а символ + буде прочитаний і проігнорований.

У команді формату може бути зазначена найбільша ширина поля, що підлягає зчитуванню.

Наприклад, scanf("%5s", str); указує необхідність прочитати з потоку введення перші 5 символів. При введенні 123456789 масив str буде містити тільки 12345, інші символи будуть проігноровані. Розділювачі: пропуск, символ табуляції і символ нового рядка - при введенні символу сприймаються, як і всі інші символи.

Функції для роботи з символами і рядками:

Функція	Заголовочний файл	Прототип	Результат

getch()	conio.h	int getch(void);	ввід одного символа з клавіатури без копіювання на екран
getche()	conio.h	int getche(void);	ввід одного символа з клавіатури з копіюванням на екран
getchar()	stdio.h	int getchar(void);	ввід одного символа в стандартний потік
gets()	stdio.h	char *gets(char *string)	ввід рядка
putch()	conio.h	int putch(int ch);	вивід символа з кодом ch на екран
putchar()	stdio.h	int putchar(int ch);	виводить символ, який є її аргументом, на екран в поточну позицію курсора.
puts()	stdio.h	int puts(char *string)	вивід в стандартний вихідний файл послідовності символів, заданої параметром string і доповненої символом нового рядка

Потокове введення-виведення

Управляти вводом / виводом можна з допомогою функцій і з допомогою об'єктів.

Для введення/ виведення програма направляє дані в потік cin або cout.

Для використання в програмі об'єктів cin або cout для введення чи виведення інформації підключають заголовочний файл iostream.h.

Потоковий ввід/ вивід опрацьовує всі вбудовані типи даних C++: char, int, float, double. Масиви виводяться поелементно, структури – по окремих полях, якщо вони відносяться до вбудованого типу даних. Рядки виводяться повністю.

Після того, як заголовочний файл включено в програму, автоматично оголошуються і стають доступними об'єкти cin і cout.

Потоковий вивід:

> *cout << ідентифікатор_змінної;*

Наприклад,

> *cout<< "x= "<<x;*

При виведенні на екран декількох змінних пропуски автоматично не виводяться. Щоб вставити пропуски, необхідно задати їх явно. Щоб вивести дані з нового рядка, використовують управляючий символ \n

Потоковий ввід:

> *cin >> <ім'я змінної>;*

Наприклад,

> *cin>>x>>y>>z;*

При цьому значення можна вводити через пропуск, табуляцію чи Enter.

Форматувати потік можна з допомогою маніпуляторів потоку бібліотеки iomanip.h. Деякі маніпулятори потоку наведено в таблиці.

Маніпулятор	Призначення
endl	початок нового рядка
ends	вставка символа null в потік виведення
flush	очистка потоку
setiosflags(long flag)	установка прапорців введення / виведення
resetiosflags(long flag)	очистка прапорців введення / виведення

setfill(char fillchar)	установка символа-заповнювача
setprecision (int places)	установка точності
setw(int width)	установка повної ширини поля

Маніпулятори setiosflags, resetiosflags, setfill, setprecision зберігають свою дію до того часу, поки не будуть перевизначені (наприклад, якщо в якості символа-заповнювача визначити “.” – то він буде таким, поки не буде визначений інший).

Наприклад,

```
float a=4.567; b=1.4;
cout<<setw(6)<<setprecision(2)<<a<<b;
//в результаті буде виведено:  4.57   1.4
```

Приклад1. Написати програму, яка пропонує користувачу ввести довжину відстані у футах, переводить одержане значення в метри і сантиметри і виводить на друк.

```
#include<stdio>
int main()
{
   float feet, meters, centimeters;
   printf (“Введіть кількість футів: ”);
   scanf ( “%f” , &feet);
   centimeters = feet*12*2.54;
   meters = centimeters / 100;
   printf(“%8.2f  (фути) дорівнює \n”, feet);
   printf(“%8.2f  (метри)  \n”, meters);
   printf(“%8.2f  (сантиметри)  \n”, centimeters);

    return 0; }
```

Результати виконання роботи:

```
Введіть кількість футів: 10
10.00 (фути) дорівнює
3.05 (метри)
304.80 (сантиметри)
```

Наприклад, ввести рядок з клавіатури і вивести його на екран, використовуючи функції введення-виведення символів і рядків

```
#include <stdio.h>
main()
{
char str[80];       // оголошення симв. змінної
  puts ("Введіть Ваше ім’я:"); // виведення запиту на екран
  gets(str);              // зчитування рядка
  printf("\n Я знаю Ваше ім’я, Ваше ім’я %s", str);
  return 0;
}
```

Результати виконання роботи:

```
Введіть Ваше ім’я:     Джонатан Свіфт
Я знаю Ваше ім’я, Ваше ім’я Джонатан Свіфт
```

Файлові структури даних

- Файли: структура логічного і фізичного файлів, класифікація файлів за внутрішньою організацією.
- Оголошення даних файлового типу. Функції відкриття і закриття файлу. Функції запису до файлу; створення нового файлу; функції читання з файлу.
- Реалізація основних кроків роботи з файлами засобами C/C++.

Оголошення даних файлового типу

Файли призначені для постійного зберігання великих об'ємів даних на зовнішніх носіях(магнітних дисках, оптичних дисках, магнітних стрічках).

Ввід / вивід файлів може здійснюватись:

- за допомогою стандартних функцій бібліотеки C;
- за допомогою потоків введення/ виведення старого стилю C++;
- за допомогою класів введення – виведення бібліотеки стандарту ANSI/ISO для C++.

Файловий ввід/ вивід у C++ складається з п'яти основних кроків:

a. Включення файла заголовків stdio і оголошення вказівника файла FILE *f1, *f2, *fk;.

b. Відкриття файла (функції open(), fopen, freopen()) і присвоєння значення-результату функції вказівнику файла.Параметрами функції є дві стрічки - ім'я файла і тип доступу до файла.

c. Перевірка правильності відкриття файла (вказівник файла після кроку 2 не повинен дорівнювати NULL).

d. Читання з файла і/або запис у файл за допомогою функцій fprintf, fscanf, fputs, fgets і інших, оголошених у бібліотеці stdio. Вони аналогічні функціям printf, scant, puts, gets, за винятком того, що мають ще один параметр - вказівник файла.

e. Закриття файла (функція fclose із вказівником файла в якості параметра).

Оголошення файла

FILE * ідентифікатор;

Ідентифікатор - це логічне ім'я файла, використовується в програмі для звертання до файла, а не ім'я файла на диску. Відповідність між ідентифікатором і ім'ям файла на диску визначається функцією open().

Після оголошення файла операції над ним можна виконувати за допомогою функцій бібліотеки stdio.

Функції відкриття і закриття файла, запису до файла, створення нового файла, читання з файла

Прототипи функцій відкриття і закриття файлів:

```
int  fclose (FILE  *f);
FILE *fopen(const char *fname, const char *mode);
FILE *freopen(const char *fname, const char *m, FILE *f);
FILE *tmpfile(void);
```

fopen (fname, mode) відкриває файл з ім'ям fname для роботи в режимі, вказаному рядком mode. При успішному відкритті повертається вказівник на потік (у такий спосіб отриманий потік зв'язується з файлом), у противному випадку - NULL ;

У параметрі fname можна вказувати шлях до файла (через \\).

Параметр mode є рядком, що містить одне зі значень, перерахованих у табл.

Режими відкриття файла:

Режим	Дія
“r”	Відкриття текстового файла на читання
“w”	Створення нового текстового файла на запис
"a"	Відкриття текстового файла на дозапис
"rb"	Відкриття двійкового файла на читання
"wb"	Створення нового двійкового файла на запис
"ab"	Відкриття двійкового файла для дозапис
"r+"	Відкриття текстового файла на читання і запис
"w+"	Створення нового текстового файла на читання і запис
"a+"	Відкриття текстового файла на читання і запис
"rb+"	Відкриття двійкового файла на читання і запис
"wb+"	Створення нового двійкового файла на читання і запис
"ab+"	Відкриття двійкового файла на читання і запис

fclose(f) закриває потік введення/ виведення f;

freopen (fname, m, f) аналогічна fopen з тими ж режимами відкриття файла, але попередньо пробує закрити відкритийфайл;

tmpfile() відкриває тимчасовий файл у режимі двійкового введення/ виведення і повертає вказівник на потік .

Після того як отримано вказівник на потік, він може використовуватися як аргумент f у функціях введення/ виведення.

Функції запису у файл

Прототипи функцій виведення у файл:

```
int fflush(FILE *f);
int fprintf(FILE *f, const char *fmt,...);
int fputc(int ch, FILE *f);
int fputs(const char *s, FILE *f);
size_t fwrite(const void *p, size_t size, size_t n, FILE *f);
int putc(int ch, FILE *f);
```

fflush(f) - вмістиме буферу виведення фізично записується у файл. При успішному завершенні повертає значення 0, інакше повертається значення EOF; Буфер автоматично очищається.

fprintf (f, fmt, список аргументів) записує в потік, адресований параметром f, значення аргументів зі списку у відповідності з форматом, заданим рядком fmt. Повертає число записаних символів . Операції перетворення, задані у рядку управління форматом, і команди виведення аналогічні операціям і командам, що використовуються функцією printf().

fputc (ch, f) записує символ ch у потік f (в поточну позицію файла), а потім зміщує індикатор позиції файла. При помилці повертає значення EOF, інакше - записаний символ;

fputs (s, f) записує рядок символів s у потік f. Символ кінця рядка у файл не записується. При помилці повертає значення EOF, інакше – невід’ємне число;

fwrite (p, size, n, f) записує в потік f n елементів довжиною size байт із буфера, заданого вказівником p, у потік f. Повертає число реально записаних елементів.

Функції читання з файла

Прототипи функцій читання з файла:

```
int fgetc(FILE *f);
char *fgets(char *s, int n, FILE *r);
size_t fread(void *p, slze_t size, size_t n, FILE *f);
int fscanf(FILE *f, const char *fmt,...);
int getc(FILE *f);
```

fgetc (f) зчитує черговий символ із потоку f. Якщо символ не може бути зчитаний, те повертається значення EOF. Символ читається як unsigned char, яке перетворюється в int; При досягненні кінця файла функція fgetc() повертає значення EOF.

fgets (s, n, f) читає не більше n -1 байт із потоку f у рядок, адресований вказівником s, припиняючи читання при виявленні символу нового рядка , кінця файла, або досягнення заданої межі. Символ нового рядка зберігається і залишається частигою рядка, адресованого s. При виявленні помилки повертається NULL, у противному випадку - вказівник на рядок s (кінець файла в даному контексті – не помилка);

fread (p, size, n, f) читає з потоку f n елементів, кожний довжиною size байт, у буфер, заданий вказівником p. Повертає число прочитаних елементів;

fscanf (f, fat,...) відповідно до формату, показаного рядком fat, читає з потоку f змінні, список адрес яких позначений (...). Повертає число зчитаних елементів або EOF у випадку невдалого читання.

getc (f) - те ж, що і fgetc;

ungetc (ch,f) повертає символ ch назад у вхідний потік f. Повертає EOF у випадку помилки, у противному випадку повертає ch. Гарантується повернення тільки одного символу.

Функції позиціювання файла

Прототипи функції для роботи з позицією у файлі:

```
int feof(FILE *f);
int fgetpos(FILE *f, fpos_t *pos);
int fseek(FILE *f, long off, int org);
int fsetpos(FILE *f, const fpos_t *pos);
long int ftell(FILE *f);
```

Ці функції визначають позицію у файлі. Вони особливо зручні для організації довільного доступу до файлів. Аргумент pos має тип fpos_t, що визначається у файлі stdio.h, fpos_t - це або ціле, або структура достатньо велика для того, щоб записати або зберегти будь-яку позицію у файлі:

feof (f) перевіряє індикатор позиції файла, щоб вияснити, чи досягнуто кінець файла, зв’язаного з потоком f, і повертає ненульове (істинне) значення при виявленні кінця файла (макрос) і відмінне від нуля значення в противному випадку.

fgetpos (f, *pos) повертає поточну позицію у файлі, пов'язаному з потоком f, і копіює значення за адресою pos. Це значення може використовуватися функцією fsetpos().

fseek (f, off, org) переміщає індикатор позиції файла, пов'язаного з потоком f, на позицію off, що відраховується від значення org (ціле); призначення цієї функції – підтримка операцій введення – виведення у файлах з довільним доступом.

fsetpos (f, pos) переміщає поточну позицію у файлі, пов'язаному з потоком f, на позицію *pos, попередньо отриману за допомогою функції fgetpos;

ftell (f) повертає поточну позицію у файлі, пов'язаному з потоком f, як довге ціле.

Для функції fseek аргумент org може приймати одне з таких значень: SEEK_SET (встановити позицію відносно початку файла), SEEK_CUR (установити позицію щодо поточної позиції) і SEEK_END (установити позицію щодо кінця файла).

Функції опрацювання помилок

Прототипи функцій опрацювання помилок:

```
void clearerr(FILE *f);
int ferror(FILE *f);
void rewind(FILE *f);
void setbuf(FILE *f,char *p);
setvbuf(FILE *f,char *p,int mode,size_t size);
```

Файл заголовків: stdio. h або cstdio.

clearerr (f) очищає прапори помилок для f. Після виникнення помилки робота з файлами може бути продовжена тільки після скидання помилки за допомогою функцій clearerr або rewind;

ferror (f) повертає ціле, що означає код помилки (0 означає відсутність помилки);

rewind (f) очищає прапори помилок і встановлює поточну позицію на початок файла;

setbuf (f, p) задає буфер введення/ виведення для потоку f, або, якщо параметр p рівниу нулю, відключає буферизацію. Якщо необхідно задати розмір буфера, визначений програмістом, то

його довжину треба встановити рівною BUFSIZ символам. Ідентифікатор BUFSIZ визначається у файлі <stdio>/

setvbuf (f, p, mode, size) аналогічна setbuf, за винятком того, що mode указує режим, а size - розмір буфера. Повертається значення 0 при успішному завершенні і ненульове значення у випадку помилки. Параметр mode є цілим, що приймає одне зі значень: _IOFBF (повна буферизация), _IONBF (заборона буферизації) і _IOLBF (для виведення - порядкова буферизація, тобто очищення буфера при записі в буфер символу нового рядка).

Опрацювання файлів

Читання і запис текстових файлів

Засобами C++ опрацьовують два види файлів: текстові і бінарні.

Текстові файли організовані в рядки, кожен з яких позначений символом кінця рядка (endl). При цьому всі дані зберігаються в символьному вигляді. Поля даних розділені пропусками. Бінарні файли не організовані в рядки. Інформація заноситься підряд, числа зберігаються у двійковому форматі.

Перед використанням файлів їх оголошують.

Перед читанням або записом файл необхідно відкрити. Це можна зробити за допомогою функції fopen. При успішному завершенні вона повертає вказівник типу FILE*, що надалі використовується в якості параметра в інших функціях:

```
FILE *fp;
fp = fopen("test. txt","w");
```

Тут рядок "w" є параметром, що визначає режим, у якому буде використовуватися потік і пов'язаний із ним файл. У даному випадку рядок "w" указує, що потік використовується для виведення в текстовому режимі (див. опис функції fopen).

Відмінність між текстовим і двійковим режимами достатньо мала. У текстовому режимі кожна пара символів carriage return/line feed (повернення каретки/перевід рядка) при читанні з файла в пам'ять заміняється на один символ line feed (символ нового рядка), і навпаки, при виведенні символ нового рядка перетворюється в пару символів.

Головною відмінністю між текстовим і двійковим режимами є спосіб інтепретації інформації. У текстовому файлі числа представляються символами цифр, а в двійковому використовується внутрішнє представлення числа, що залежить від типу комп'ютера:. .

```
void test_files(void)
{
FILE *fp;
int i = 12;
fp = fopen("test. txt","w");
if (fp ! = NULL)
   fprintf(fp,"Ось число: %d\n",i);
else
   fputs("Помилка відкриття файла\n", stderr);
```

При роботі з текстовими файлами гарним інструментом є функції fgets і fputs - вони узгоджено співпрацюють між собою. Функція fgets читає рядок із файла і включає в нього символ нового рядка і нуль-термінатор. Функція fputs виводить такий рядок без додавання символу нового рядка і нуль-термінатора.

Розглянемо приклад програми, що копіює текстовий файл. Кожний рядок у файлі-копії починається символом «>»:

```
#define LINESZ 256
 void main(void)
{
FILE *inf, *outf;
char s[LINESZ]:
inf = fopen("test. txt","r");
outf = fopen("test1. txt","w");
if (inf == NULL || outf == NULL) {
    puts("Не вдалося відкрити файл. \n");
    return 1;
                                    }
```

```
fgets(s,LINESZ,inf);
while (!feof(inf)) {
fputs(">",outf);
fputs(s,outf);
fgets(s, LINESZ, inf);
}
```

Читання і запис двійкових файлів

Читання і запис двійкових файлів, як правило, провадяться за допомогою функцій fread і fwrite.

Щоб працювати з двійковим файлом, необхідно використовувати символ b у параметрі, що визначає режим відкриття (див. також опис функції fopen). У двійковому режимі ніяких перетворень даних не провадиться, тобто нуль-термінатор при читанні не добавляється:

```
#include <stdio>
 // . . .
FILE *fb;
fb = fopen("stuff. dat","wb");
```

Нижче наведено текст програми виведення вмісту файла в 16-ковому форматі (дампа файла). Ця програма запитує й одержує ім'я файла, а потім виводить дамп на стандартний пристрій виведення:

```
#define BYTES_PER_ROW 16
int main() {
FILE *fb;
char filespec[256], buf[BYTES_PER_ROW];
int i, n;
printf("Введіть ім'я файла:");
gets(filespec);
fb = fopen(filespec,"rb");
if (fb == NULL) {
   printf("Не удалося відкрити файл %s\n",filespec);
   return 1;          }
do   {
     n = fread(buf,1,BYTES_PER_ROW,fb);
     for (i = 0; i < n; i++) // 16-ковий дамп
         printf("%02X ",buf[i]);
     for (i=0; i < n; i++) // символьний дамп
         if (iscntrl(buf[i])) putchar('.' );
            else
               putchar(buf[I]);
     putchar('\n'); }
while (n == BYTES_PER_ROW);
return 0; }
Ключова інструкція цієї програми:
n = fread(buf,1,BYTES_PER_ROW,fb);
```

Параметр buf визначає масив, у який заносяться зчитані дані. Другий параметр задає розмір кожного зчитаного елемента. Так як цей розмір дорівнює 1, то функція зчитує байти по одному. Третій параметр задає число елементів (у нашому випадку байтів) читання. Тут він визначений константою BYTES_PER_ROW, рівною 16, тобто довжині одного рядка дампа. Після читання програма виводить 16-кові значення байтів і їхнє символьне представлення.

Оскільки функція tread повертає число реально зчитаних елементів, то при числі зчитаних символів менше 16 байт (що можливо тільки при досягненні кінця файла) програма виводить ці символи і завершується:

```
do {
     n = fread(buf,1,BYTES_PER_ROW,fb);
     // . .. }
 while (n == BYTES_PER_ROW);
```

Функція fwrite працює аналогічно і має той же список параметрів - вона намагається записати у файл задане число елементів. Якщо розмір елемента дорівнює 1, то записуються окремі байти:

```
n = fwrite(buf,1,BYTES_PER_ROW,fb);
```

ЛІТЕРАТУРА

1) Страуструп Б. Дизайн и эволюция C++: Пер. с англ.- М.:ДМК Пресс; СПб: Питер, 2016 - 446 с.

2) Айвор Хортон. Visual C++ 2010: полный курс. Изд-во: Диалектика-Вильямс, 2011. – 1216 с.

3) О. Петрик, М. Петрик. Технологія програмування в системі С++. Лабораторний практикум. Тернопіль, видавництво ТДТУ ім. Ів. Пулюя, 2005 - 107с.

4) «Основи програмування» для студентів спеціальності 121 Інженерія прграмного забезпечення. / Уклад. Петрик О.Ю. –Тернопіль: ТНТУ імені Івана Пулюя, 2014. [Електронний ресурс]. - Режим доступу: URL: https://dl.tntu.edu.ua/content.php? course=1060

5) Шпак З.Я. Програмування мовою С. – Львів: Видавництво львівської політехніки, 2011.-436с. [Електронний ресурс]. - Режим доступу: URL:

http://programming.in.ua/programming/basisprogramming/275-programming-c-book-shpak.html

6) Шилдт Г. C++: базовый курс, 3-е изд. : Пер. с. англ. – М.:Изд. дом «Вильямс», 2010. – 624с.

7) Глушаков С.В., Коваль А.В., Смирнов С.В. Язык программирования С++: Учебный курс.- Харьков: Фолио; М.:ООО "Издательство АСТ", 2001.-500 с.

8) Дейтел Х., Дейтел П. Как программировать на С++. 5-е изд. - М.:ООО „Бином-Пресс”, 2008г.–1456с

9) Романов Е.Л. Практикум по программированию на С++: Учебное пособие. СПб: БХВ-Петербург, Новосибирск: Изд-во НГТУ, 2004. – 432 с.

10) Кнут Д. Искусство программирования. т.2. Получисленные алгоритмы.- М., СПб., К.:Вильямс, 2000.- 832 с.

11) Прата С. Язык программирования С++. Лекции и упражнения. 5-е изд. –М.: ООО «И. Д. Вильямс», 2007 – 1184с.

12) Шилдт Г. Полный справочник по C++. 4-е издание. - М.: Изд.дом «Вильямс», 2006г.–800с.

13) Шилдт Г. C++: руководство для начинающих, 2-е изд.: Пер. с. англ. – М.:Изд.дом «Вильямс», 2005. – 672с.

14) Основи програмування на мовах С та С++ для початківців [Електронний ресурс]. - Режим доступу: URL: http://cppstudio.com/uk/

15) Довідник по C++ [Електронний ресурс]. - Режим доступу: URL: https://msdn.microsoft.com/uk-ua/library/3bstk3k5.aspx

16) Система для проведення дистанційних олімпіад та змагань зі спортивного програмування [Електронний ресурс]. - Режим доступу: URL: https://www.e-olymp.com/uk/

17) Система онлайн-тестування [Електронний ресурс]. - Режим доступу: URL: http://www.quizful.net/category/cpp

18) IDE Visual C++ [Електронний ресурс]. - Режим доступу: URL https://support.microsoft.com/uk-ua/help/2977003/the-latest-supported-visual-c-downloads