Мова програмування. Поняття програми

Як було сказано раніше, ***мова програмування*** **–** фіксована система позначень та правил для опису структур даних та алгоритмів, призначених для виконання обчислювальними машинами.

З формальної точки зору ***мова програмування*** – це набір вихідних символів *(алфавіт)* разом із системою правил утворення з цих символів формальних конструкцій *(синтаксис)* та системою правил їх тлумачення (інтерпретації) *(семантика),* за допомогою яких описуються алгоритми. **Програма –** це алгоритм записаний за допомогою мови програмування.

Алфавіт, синтаксис та семантика — це три основні складові частини будь-якої мови програмування. До **алфавіту** мови програмування, як правило, входять літери латинського алфавіту, арабські цифри, знаки арифметичних операцій, розділові знаки, спеціальні символи. Із символів алфавіту будують послідовності, які називають *словами (лексемами).* Кожне слово у мові програмування має своє змістове призначення. **Правила синтаксису** пояснюють, як потрібно будувати ті чи інші мовні повідомлення для опису всіх понять мови, здійснення описів та запису вказівок. **Правила семантики** пояснюють, яке призначення має кожен опис та які дії повинна виконати обчислювальна машина під час виконання кожної із вказівок. Вказівки на виконання конкретних дій називають ще **командами** або **операторами** мови.

Усі слова або ж лексеми, поділяють на

- *службові* (зарезервовані або ж ключові) слова;
- *стандартні ідентифікатори;*
- *ідентифікатори користувача;*
- *знаки операцій;*
- *літерали;*
- *розділові знаки*.

Службові слова мають наперед визначене призначення і використовуються для формування структури програми, здійснення описів, позначення операцій, формування керуючих конструкцій (вказівок). Наприклад, службовими словами для мови C++ є: abstract, do, in, protected, public, try, else, interface, enum, break, return, new, sizeof, using, class, const, for, operator, continue, struct, while, switch, default, if, this, private.

C++Імена (позначення) для програмних об'єктів (типів даних, констант, змінних, функцій і т. п.) формують у вигляді ідентифікаторів. *Ідентифікатор* — це послідовність латинських літер, цифр і знака підкреслення, яка розпочинається з латинської літери. У мові C++максимальна довжина ідентифікатора є необмеженою.

C++ switch, default,*Стандартні ідентифікатори* використовуються як імена для стандартних (передбачених авторами мови програмування) типів даних, констант, підпрограм (зокрема, стандартних математичних функцій). Приклад: int, long,double, char, bool та ін.

Ідентифікатори користувача є іменами тих програмних об'єктів (констант, змінних, функцій тощо), які створює сам користувач. Службові слова та ідентифікатори користувача не повинні збігатися.

Коментар – невиконувана частина тексту програми, що ігнорується компілятором і служить для вставки деяких поміток у програмі тільки для програміста. Коментарі бувають однорядковими та багаторядковими. Однорядковий коментар починається з

символів «//» і закінчується у кінці рядка. Тобто всі символи до кінця рядка вважаються коментарем.
Приклад.

```
// Це однорядковий коментар
```

Багаторядковий коментар починається з символів «/*» і закінчується символами «*/».
Приглад.

```
/* Це
   багаторядковий
   коментар */
```

Літерал – це явно вказане значення декого типу. Розрізняють такі типи літералів:

тип літерала	*опис*	*приклади*
цілочисловий	у десятковій системі числення – звичне нам ціле число	125, -89, 108;
	у вісімковій системі числення – починається з 0	023, 075, 0416;
	у шістнадцятковій системі числення – поичанається з 0x або 0X	0x4A, 0x6FE2, 0xABC
дійсного типу	у форматі з фіксованою крапкою – у записі числа є крапка, що розділяє цілу і дробову частити	25.69, 2.0, 145.058
	у форматі з плаваючою крапкою – у записі використано символ «e» або «E», що розділяє мантису від порядку	3.5e5, 3e12, 678e2
символьний	заданий у явному вигляді – символ записується у одинарних лапках	‘Z’, ‘a’, ‘E’
	прості ескейп-послідовності – службові символи починаються з символу «\»	\n -перехід на новий рядок, \t -горизонтальна табуляція, \’ - апостороф, і т. д.
	ескейп-послідовності Unicode – символи «\u», за якими вказують код символу з чотирьох цифр у шістнадцятковій системі числення	‘\u0123’ , ‘\u3A58 ’
рядковий	регулярний – у подвійних лапках вказується рядок символів (ескейп-послідовності обробляються)	”Я люблю C++”
логічний	може приймати два значення	true, false

Розділові знаки служать для відокремлення однієї лексеми від іншої. Ними є пробіл, табуляція, символ нового рядка, символ «;» та коментарі.

ВЕЛИЧИНА. ТИП ВЕЛИЧИНИ

У своїй роботі програміст має справу з таким поняттям, як величина. З точки зору програмування ***величини*** – це дані, що обробляються програмами. *Дані* — це інформація, введена у пам'ять комп'ютера або підготовлена до введення.

Носіями даних у програмах є *константи*, *змінні* (значення яких зберігається в оперативній пам'яті) та *файли* (на зовнішніх носіях інформації).

Константи — це величини, значення яких у процесі виконання програми не змінюється. *Змінні* — це величини, значення яких у процесі виконання програми можуть змінюватися. Імена констант і змінних, як і інших програмних об'єктів, записують у формі ідентифікаторів. Кожна змінна і константа належать до визначеного типу.

Тип даних – це сукупність властивостей певного набору даних, від яких залежать: діапазон значень, якого можуть набувати ці дані, а також сукупність операцій, які можна виконувати над цими даними.

З іншого боку **тип даних** – це описання того, яку структуру, розмір мають комірки оперативної пам'яті при зберіганні відповідного елемента даних.

Елемент даних певного типу – це комірка або комірки оперативної пам'яті, що мають фіксовану адресу, розряди яких розшифровуються згідно описання даного типу даних.

З кожним типом даних зв'язано своє унікальне ім'я (ідентифікатор), яке є синонімом певного описання елементу даних відповідного типу. Наприклад, ідентифікатор int є синонімом опису : 32 послідовних розрядів містить ціле значення в діапазоні від (-2^{32}) до $(2^{32}-1)$ (у двійковому вигляді займає 4 байт).

У зв'язку з цим можна дати інше означення константи та змінної.

Якщо елемент даних не може змінювати свого значення, тобто завжди містить одне і те ж саме значення, то відповідний ідентифікатор називається *константою даного типу*. Якщо елемент даних певного типу може змінювати своє значення під час виконання програми, то ідентифікатор, що зв'язаний з цим елементом даних називається змінною відповідного типу. *Значення змінної* – це елемент даних, з якими ця змінна пов'язана.

Отже, у програмах змінна характеризується такими ознаками: *іменем*, *типом* і *значенням*.

Типи даних C++

У мові програмування визначено такі елементарні типи даних

Ім'я типу		Кількість байт	Діапазон
bool	логічний тип	1	true – false
char	символьний тип	1	символи з таблиці кодів ASCII (0 – (2^8-1))
wchar_t	двобайтовий символьний тип	2	0 – $(2^{16}-1)$
int	цілочисловий тип	4	(-2^{31})– $(2^{31}-1)$
float	тип дійсних чисел	4	3.4E-38 – 3.4E+38
double	тип дійсних чисел	8	1.7E-308 – 1.7E+308

Деякі з типів можна модифікувати ключовими словами signed (знаковий), unsigned (беззнаковий), short (короткий) и long (довгий).

[<модифікатор>] [< тип >]

При цьому, якщо тип не вказано, то вважається, що типом даних є тип int.

Цілочислові типи	Кількість байт	Діапазон

signed char	1	$-2^7 - (2^7-1)$
unsigned char	1	$0 - (2^8-1)$
short short int signed short signed short int	2	$(-2^{15}) - (2^{15}-1)$
unsigned short unsigned short int	2	$0 - (2^{16}-1)$
int signed signed int	4	$(-2^{31}) - (2^{31}-1)$
unsigned unsigned int	4	$0 - (2^{32}-1)$
long long int signed long signed long int	8	$(-2^{63}) - (2^{63}-1)$
unsigned long unsigned long int	8	$0 - (2^{64}-1)$
long long long long int signed long lon gsigned long long int	16	$(-2^{127}) - (2^{127}-1)$
unsigned long long unsigned long long int	16	$0 - (2^{128}-1)$
Дійсні типи	Кількість байт	Діапазон
long double	10	3.4E-4932 – 3.4E+4932

Створення псевдонімів типів даних. У C++ є можливість описувати для типів даних псевдоніми. Такий опис здійснюється з використанням оператора typedef.

Загальний вигляд	Приклад

typedef <Тип даних> <Псевдонім типу даних>;	typedef long int lint;
	typedef double Myd;
	
	//Зараз опис
	long int m;
	//і опис
	lint m;
	//є еквівалентними

Консольні програми у C++

Консольна програма – це програма, яка дозволяє виводити символьну інформацію на екран та вводити символьну інформацію з клавіатури. Точкою входу у будь-яку консольну програму є функція main.

Структура консольної програми

Загальний вигляд	Приклад

<Підключення заготовочних файлів>	#include "stdafx.h" #include <iostream>; using namespace std;
<Оголошення глобальних змінних> <Оголошення функцій> <Оголошення класів > ……	int d=35; int sum(int x, int y);

int main ()
{
<оператори >
return 0;
}
	int main()
	{
	cout<<"Hello!!"<<endl;
	cout<<"d="<<sum(d,5)<<endl;
	system("pause");
	return 0;
	}

<Реалізація функцій і методів класів>	int sum(int x, int y)
	{
	return x+y;
	}

Програма у C++ , як і у інших мовах програмування, може складатися з багатьох файлів. Серед них можна виділити так звані *файли заголовків*, які можуть містити описи різного роду програмних об'єктів: функцій, констант, типів даних та ін. Розширення файлів заголовків – « .h ». Якщо при написанні програми необхідно використати деякі чи стандартні, чи розроблені програмістом програмні об'єкти, що розміщено в іншому файлі заголовка, то необхідно підключити цей файл заголовка з використанням директиви include

include <ім'я файлу заголовка >;

Зауважимо, що при підключенні стандартних заготовочних файлів розширення «.h» не вказують. Так у наведеному прикладі було підключено файл заголовка iostream, що містить опис стандартних об'єктів cout та cin, які дають можливість здійснювати введення та виведення даних

#include <iostream>;

Опис змінних

Змінна – це іменована ділянка оперативної пам'яті, що використовується у програмі для збереження даних. Перш ніж використати змінну, її необхідно попередньо описати

Загальний вигляд	Приклад
[<Специфікатор>] [<Модифікатор типу>] < **Тип** > <**Ім'я змінної**>;	int x; double d;

При описі змінної можна одразу надавати їй початкове значення, тобто ініціалізувати цю змінну

Загальний вигляд	Приклад
[<Специфікатор>] [<Модифікатор типу>] < **Тип** > <**Ім'я змінної**>=<**Значення**>;	int x = 12; double d=2.5;

Описуючи змінну, можна також вказувати *специфікатори* зберігання, які впливають на місце розташування змінної.

Специфікатор	Призначення
auto	тип змінної визначається на основі значення, яким ця змінна ініціалізується
register	значення змінної буде зберігатися у регістрах процесора (як правило, такі змінні дуже часто використовуються)
extern	пояснює компілятору, що повний опис змінної знаходиться в іншому місці (наприклад, у іншому файлі)
static	використовується при описі змінних у середині функцій, що дозволяє зберігати значення між викликами цих функцій

Визначення розміру змінних та типів. Кількість байтів, які виділяються для змінної можна визначити з використанням оператора sizeof, який має два формати

Загальний вигляд	Приклад
sizeof(<Змінна>)	int x; cout<<sizeof(x);
sizeof(<Тип даних>)	cout<<sizeof(int);

Динамічне визначення типів змінних. Тип змінної у процесі роботи програми можна визначити з використанням оператора typeid, який описано у файлі заголовка typeinfo. Оператор typeid повертає об'єкт класу type_info, який дозволяє отримати ім'я типу, за допомогою функції name.

Загальний вигляд	Приклад
typeid (<Змінна>).name()	#include < typeinfo > int x; typeid (x).name(); // int

Константи

Константи – це величини, значення яких не можуть змінюватися у процесі роботи програми. Їх описують з використанням ключового слова const

Загальний вигляд	Приклад
const <тип> <ім'я> =<значення>;	const int x=55; const double d=3.7;

Області видимості

Перш ніж змінну використати, її обов'язково необхідно описати. Описати змінну можна як глобальну (описану поза межами функцій), так і локальну (описану у середині якоїсь функції чи блоку).

Глобальна змінна описується поза межами функцій і може бути використана у будь-якій частині програми після її опису.

Локальна змінна описується у середині якогось блоку (всередині фігурних дужок) чи функції. Вона може бути використана тільки в межах блоку чи функції, у яких вона описана.

Якщо всередині блоку ім'я локальної змінної співпадає з іменем глобальної змінної, то кажуть, що локальна змінна приховує глобальну. Для доступу до глобальної змінної у межах цього блоку необхідно використати оператор « **: :** »

Загальний вигляд	Приклад

: : <Змінна>	int x=20; //Опис глобальної змінної x
	int func(){
	int x=3.5; //Опис локальної змінної x
	cout<<"Локальна x=" << x <<endl; //Локальна x=3.5
	cout<<"Глобальна x="<< **: : x**<<endl; //Глобальна x=20
	}

Простори імен

При розробці великих і складних програм декількома програмістами часто виникає ситуація, коли одними і тими ж іменами позначають величини різних типів. Але ж у програмі не можна описати дві різні змінні з однаковим іменем. Для розв'язання цієї проблеми використовують так звані простори імен. *Простір імен* – це частина програми, іменування змінних і типів у якій ніяк не залежить від іншої частини програми. Розглянемо формат опису простору імен.

Загальний вигляд	Приклад

namespace <Назва простору імен>{
 	
<опис змінних, типів та ін.>
 	
}
	namespace nsp1{
	
	int g=20;
	
	}
	namespace nsp2{
	
	double g=83.9;
	}

Якщо у межах одного простору імен необхідно здійснити доступ до змінної чи типу, описаних у іншому просторі імен, то необхідно використати оператор « : : ».

Загальний вигляд	Приклад
<Назва простору імен>::<ідентифікатор>	cout<<"nsp1::g="<<nsp1::g<<endl; //20 cout<<"nsp2::g="<<nsp2::g<<endl; //83.9

Для спрощення доступу до програмних об'єктів, описаних у іншому просторі імен можна здійснити імпортування всіх описів або окремих ідентифікаторів цього простору імен з використанням оператора using.

Загальний вигляд	Приклад
//Імпортування всіх описів using namespace <Назва простору>;	using namespace nsp1; cout<<"nsp1::g="<< g <<endl;
//Імпортування окремих ідентифікаторів **using** <Назва простору>::<ідентифікатор>;	**using** nsp1::**g**; cout<<"nsp1::g="<< g <<endl;

При описі розглянутих просторів імен вказувались назви цих просторів (*іменовані простори імен*), але у С++ є також можливість описувати так звані *неіменовані (анонімні) простори*, при описі яких не вказують імен

```
namespace {

  <опис змінних, типів та ін.>

}
```

Ідентифікатори, описані у неіменованому просторі імен, доступні тільки у межах файлу, де цей простір імен описано. Звертання до ідентифікаторів цього простору імен здійснюється так само як і для глобальних змінних.

Виведення даних

Виведення даних може бути здійснено з використанням об'єкта cout з простору імен std, що описано у файлі заголовку iostream

Загальний вигляд	Приклад

#include <iostream>;
…….
 	
std: :cout<< <Дані1> << <Дані2><<…
	#include <iostream>;
	…….
	int x=36;
	std : : cout << "x=" << x << endl;
	std : : cout << "Hello" << endl;

Спростити доступ до cout можна, виконавши імпорт описів з простору імен std.

Загальний вигляд	Приклад

#include <iostream>;
using namespace std;
…….
 	
cout << <Дані> << <Дані><<…
	#include <iostream>;
	using namespace std;
	…….
	int x=36;
	cout << "x=" << x << endl; // x=36
	cout << "Hello" << endl; // Hello

Введення даних

Введення даних може бути здійснено з використанням об'єкта cin з простору імен std, що описано у файлі заголовку iostream.

Загальний вигляд	
#include <iostream>; ……. std: :cin >> <Змінна> ;	#include <iostream>; ……. int x; std : : cout << "x="; std : : cin >> x;

Спростити доступ до cin можна, виконавши імпорт описів з простору імен std

Загальний вигляд	
#include <iostream>; **using namespace std**; ……. cin >> <Змінна>	#include <iostream>; **using namespace std**; ……. int x; cout << "x="; cin >> x;

Оператор присвоєння. Перетворення типів

Оператор – це логічно завершена конструкція (вказівка), призначена для виконання конкретних дій.

Усі оператори можна поділити на **прості та складні**. **Прості** оператори не містять всередині себе інших операторів. **Складні** або **структурні** оператори представляють собою конструкції, що містять інші оператори (як складні так і прості). До простих операторів як правило відносять оператор присвоєння та виклик процедури.

Зрозуміло, якщо у програмі буде обчислене значення виразу, то його необхідно десь запам'ятати для подальшого використання. Для цього існує оператор присвоювання.

Загальний вигляд	Приклад

<ім'я змінної> = <вираз>;	int n;
	n=12;
	int i=n+2;

Під час виконання цього оператора спочатку обчислюється значення виразу в правій частині при поточних значеннях змінних, що входять до нього, а потім отриманим результатом замінюється попереднє значення змінної, що вказана зліва.

Оператор присвоєння можна застосовувати до числових, логічних та символьних даних.

Приклад.

```
char c='d';
int n=9;
double d=2.7;
string s="перетворення типів";
bool b=true;
```

При використанні оператора присвоєння необхідно слідкувати, щоб тип виразу у правій частині відповідав типу даних змінної в лівій. Але часто трапляються випадки коли змінній необіхно присвоїти значення виразу, тип якого не співпадає із типом змінної. Наприклад, коли змінній дійсного типу необхідно присвоїти значення цілого типу. В цьому випадку необхідно виконати перетворення (приведення) виразу до необхідного типу (типу змінної). У мові розрізняють два види перетворень змінних – *явне* та *неявне*.

Неявне перетворення типів використовується в тому випадку, коли дане перетворення є «природним». Тобто, наприклад, перетворюється величина типу float в величину типу double.

Приклад.

```
float f = 1.23;
double d = f;       //Неявне перетворення
```

Таке перетворення є природним, оскільки обидва типи використовуються для представлення дійсних типів, причому цільовий тип має більший діапазон представлення і більшу точність. При такому перетворення не відбувається втрата інформації. При проведенні неявного переторення немає необхідності вказувати цільовий тип (тип до якого здійснюється перетворення).

Явне перетворення типів вимагає явного задання цільового типу, до якого здійснюється перетворення. Цільовий тип вказується в дужках перед значенням тип якого перетворюється.

Загальний вигляд	Приклад
<змінна> = (<цільовий тип>) <вираз>;	double d = 2.9; float f = (float) d;

Для цього типу повинно існувати неявне перетворення до типу змінної, в якій буде збережено результат.

Арифметичні та логічні вирази

Вирази

Виразом називають послідовність операцій, операндів і розділових знаків, що задають деякі обчислення. В залежності від значення, яке одержується в результаті цих обчислень, вираз поділяють на арифметичні та логічні вирази.

Арифметичні вирази

Арифметичним виразом називають вираз, в результаті обчислення якого одержуємо числове значення. При цьому можуть використовуватися бінарні та унарні операції арифметичні операції.

Конструктивно означення **арифметичного виразу (АВ)** можна дати за допомогою правил його побудови:

1. Довільна числова константа – АВ
2. Довільна числова змінна – АВ
3. Довільний виклик підпрограми (процедури або функції) – АВ
4. Якщо А – **АВ,** то (А) – АВ
5. Якщо А, В – АВ, то А+В, А-В, А*В, А/В – арифметичні вирази

При складанні арифметичного виразу необхідно використовувати **наступні правила**:

1. Усі складові частини виразу записують в один рядок

$\frac{a+b}{2}$ як $(a+b)/2$, $2a+\frac{4b-c}{(a+b)}$ як $2*a+(4*b-c)/(a+b)$

2. У виразах використовується тільки круглі дужки. При цьому кількість відкриваючих дужок має дорівнювати кількості закриваючих.
3. Обчислення арифметичного виразу здійснюється зліва направо, згідно із пріоритетом операцій.

У програмуванні, так само як і в математиці, існує пріоритет виконання арифметичних дій, тобто визначається, яким діям надається перевага перед іншими під час обчислення значення арифметичного виразу. Наведемо арифметичні операції мови С++ саме в порядку зменшення їх пріоритетності:

*, / – множення і ділення;

% – остача від ділення націло двох цілих чисел;

+ , – – додавання і віднімання.

Якщо операції, які йдуть безпосередньо одна за одною мають однаковий пріоритет , то вони виконуються в такому порядку в якому записані.

Приклад: $A*2*T+R/T*N-S$

1) $A*2$ 2) $A*2*T$ 3) R/T 4) $R/T*N$ 5) $A*2*T+R/T+N$ 6)$...-S$

Якщо з яких-небудь причин необхідно змінити порядок виконання операцій, то для цього використовуються круглі дужки (**Наприклад:** $S/(Q+T)$). Якщо у виразі є декілька вкладених дужок і вони вкладені одна в одну, то спочатку обчислюється вираз в самих внутрішніх дужках, а потім у зовнішніх.

$A*(B+C*(D+E))$ 1) $D+E$ 2)$C*(D+E)$ .....

В арифметичних виразах можуть використовуватися і **стандартні функції.** Пріоритетність обчислення функцій найвища. Отже, якщо в арифметичному виразі

використовуються функції, то спочатку буде обчислене їх значення, а потім над цими результатами будуть виконані інші дії.

Якщо аргумент функції є арифметичним виразом, то спочатку обчислюється цей вираз, а потім значення функції.

У мовах програмування аргументи функції вказуються в дужках. Тобто в математиці ви пишете sin *x*, а в C++ треба писати sin *(x)*.

Наведемо список функцій, які описано в файлі заголовку ***math.h***

Математичний запис	Запис на C++	Призначення
cos *x*	cos(x)	косинус x – радіани
sin *x*	sin(x)	синус x – радіани
tg *x*	tan(x)	тангенс x – радиан
ch*x*	cosh(x)	гіперболічний косинус x – радіани
sh*x*	sinh(x)	гиперболический синус x – радиани
th *x*	tanh(x)	гіперболічний тангенс x – радіани
arccos*x*	acos(x)	арккосинус числа x
arcsin*x*	asin(x)	арксинус числа x
arctg *x*	atan(x)	арктангенс числа x
e^x	exp(x)	значение e в степени x
x^y	pow(x,y)	число x в степені y
|*x*|	fabs(x)	модуль числа x
$\sqrt{}$	sqrt(x)	квадратний корінь числа x
ln *x*	log(x)	натуральный логарифм x
log10*x*	log10(x)	десятковий логарифм x

Бінарні операції. Бінарні операції – це операції, для виконання яких необхідно два операнди.

Операція	Позначення	Приклад
+	додавання	`z=x+y`
–	віднімання	`z=x-y`
*	множення	`z=x*y`
/	ділення	`z=x/y`
%	остача від ділення	`z=x%y`

Окрім бінарних операцій в арифметичному виразі можуть бути присутні також унарні операції «+», «–» та операції інкременту «++» і декременту «– –». Унарний мінус використовується для зміни знаку. Операції інкременту і декрменту використовують для збільшення та зменшення значення змінної на одиницю. Ці операції можуть вживатися у префіксній та постфіксній формі.

Операція	Аналог
`i++` або `++i`	`i=i+1`
`i--` або `--i`	`i=i-1`

Якщо унарну операцію вжито у префіксній формі у виразі, то вона виконується до використання значення змінної у виразі. Якщо ж унарну операцію вжито у постфіксній формі, то операція виконується після використання значення змінної у виразі.

Операція	Аналог з бінарними операціями

`int i = 5;`
 	
`int j = ++i;  //  j=6   i=6`
	`int i = 5;`
	`i=i+1;`
	`int j = i;  //  j=6   i=6`

`int i = 5;`
 	
`int j = i++;  //  j=5   i=6`
	`int i = 5;`
	`int j = i;`
	`i=i+1;  //  j=5   i=6`

`int i = 5;`
 	
`int j = --i;  //  j=4   i=4`
	`int i = 5;`
	`i=i-1;`
	`int j = i;  //  j=4   i=4`

`int i = 5;`
 	
`int j = i--;  //    j=5   i=4`
	`int i = 5;`
	`int j = i;`
	`i=i-1;  //    j=5   i=4`

Побітові операції

У мові C# є можливість здійснювати побітові операції над розрядами аргументів

Операція	Позначення	Приклад

&	побітове «і»	`int x=10; //x=1010(2)`
		`int y=7;  //y=0111(2)`
		`z=x&y  // z=2=0010(2)`

|	побітове «або»	`int x=10; //x=1010(2)`
		`int y=7;  //y=0111(2)`
		`z=x|y  //z=15=1111(2)`

^	Побітове «виключаюче або»	`int x=10; //x=1010(2)`
		`int y=7;  //y=0111(2)`
		`z=x^y  //z=13=1101(2)`

Бінарні операції зсуву

Операція	Позначення	Приклад

>>	зсув розрядів вправо
	(змінна) =(змінна)>>(кільк. розрядів)
		`int i=4;  //i=100(2)`
		`i=i>>1;  //i=2=10(2)`

<<	зсув розрядів вліво
	(змінна) =(змінна)<<(кільк. розрядів)
		`int i=4;  //i=100(2)`
		`i=i<<2; // i=16=10000(2)`

Операції присвоєння

Якщо при виконанні бінарної операції результат зберігається у змінній, що є першим аргументом, то можна використати так звані операції присвоювання.

Операція	Операція	Аналог з бінарними операціями
+=	`int i = 5;` `i += 3;  // i=8`	`int i = 5;` `i=i+3;  //  i=8`
-=	`int i = 5;` `i -= 3;  // i=2`	`int i = 5;` `i=i-3;  //  i=2`
*=	`int i = 5;` `i *= 3;  // i=15`	`int i = 5;` `i=i*3;  //  i=15`
/=	`int i = 6;`	`int i = 6;`

	`i /= 3;  // i=2`	`i=i/3;  //  i=2`
>>=	`int i = 5;` `i >>= 1;  // i=2`	`int i = 5;` `i=i>>1;  //  i=2`
<<=	`int i = 5;` `i <<= 1;  // i=10`	`int i = 5;` `i=i<<1;  //  i=10`

&=	`int i = 5;`
	`int j = 7;`
	`i &= j;  // i=5`
		`int i = 5;`
		`int j = 7;`
		`i=i&j;  //  i=5`

^=	`int i = 5;`
	`int j = 7;`
	`i ^= 1;  //  i=2`
		`int i = 5;`
		`int j = 7;`
		`i=i^j;  //  i=2`

|=	`int i = 5;`
	`int j = 7;`
	`i |= 1;  //  i=7`
		`int i = 5;`
		`int j = 7;`
		`i=i|j;  //  i=7`

Логічні вирази

Логічним виразом називається такий вираз, внаслідок обчислення якого одержується логічне значення типу `bool` (*true* або *false*).

Прикладом логічного виразу є вираз, що містить операції порівняння

Операція	Позначення	Приклад
==	рівність	`x==y`
>	Більше	`x>y`
<	Менше	`X<y`
>=	більше або рівно	`x>=y`
<=	менше або рівно	`X<=y`
!=	не рівно	`x!=y`

У арифметичному виразі можуть також використовуватися логічні операції

x	y	x&&y (логічне «і»)	x || y (логічне «або»)	x ^ y (виключаюче «або»)	!x (заперечення)
false	false	false	false	false	true
false	true	false	true	true	true
true	false	false	true	true	false
true	true	true	true	false	false

Конструктивно логічний вираз можна означити так:

1. Логічна константа (*true* або *false*) – Логічний Вираз (ЛВ).
2. Логічна змінна – ЛВ.
3. Якщо L – ЛВ то (L) – ЛВ.
4. Якщо L – ЛВ то $!L$ – ЛВ.
5. Якщо A_1, A_2 – арифм. вирази то
   $A_1 < A_2, A_1 > A_2, A_1 <= A_2, A_1 >= A_2, A_1 == A_2, A_1 != A_2,$ – ЛВ
6. Якщо L_1, L_2 – ЛВ то $L_1 \&\& L_2, L_1 \| R\ L_2, L_1 \wedge L_2,$ – ЛВ

Пріоритет операцій

Пріоритет	Операції
1	() [] . (постфікс)++ (постфікс)-- new sizeof typeof unchecked
2	! ~ (ім'я типу) +(унарний) -(унарний) ++(префікс) --(префікс)
3	* / %
4	+ -
5	<< >>
6	< > <= => is
7	== !=
8	&
9	^
10	|
11	&&
12	||
13	?:
14	= += -= *= /= %= &= |= ^= <<= >>=

Складений оператор.

Складений оператор – це складний оператор, який об'єднує декілька операторів в одну групу.

Форма його запису наступна:

```
{
    оператор 1;
    оператор 2;
    ……..
    оператор n;
}
```

В даній конструкції дужки “{“ та “}” мають назву операторних дужок. “{“ – відкриваюча операторна дужка; “}” – закриваюча операторна дужка. Складений оператор визначається як єдиний оператор. Його можна вставляти в довільне місце програми, де дозволено використання одного простого оператора.

Керуючі конструкції мови С++

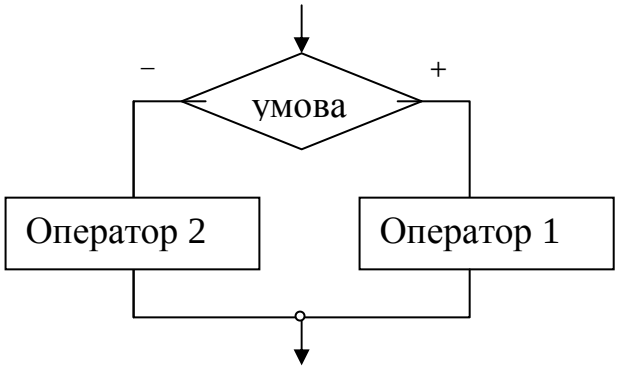
Конструкція мови С++ називається керуючою, якщо вона може змінювати послідовність виконання дій. До керуючих конструкцій відносяться умовний та циклічний оператори.

Умовний оператор

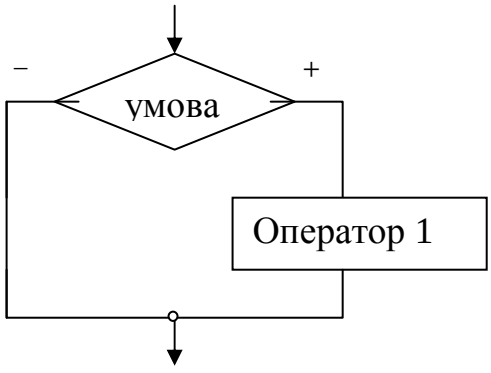
Дозволяє вибрати оператор, який буде виконуватися в залежності виконання чи невиконання деякої умови. Існують дві форми для даного оператора повна та скорочена.

Повна форма

Програмна структура	**Аналог на мові блок-схем**	**Приклад**

if (<умова>)
  <оператор1>;
else
  <оператор2>;
		`if(x>y)`
		`  max=x;`
		`else`
		`  max=y;`

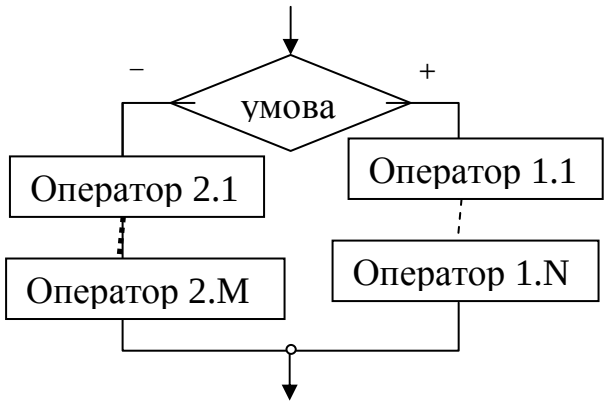
Скорочена форма

Програмна структура	**Аналог на мові блок-схем**	**Приклад**
if (<умова>) <оператор1>;		`if(x!=0)` `  z=1/x;`

Якщо в умовному операторі при виконанні чи невиконанні умови необхідно виконати декілька операторів, то необхідно ці оператори помістити в складений оператор.

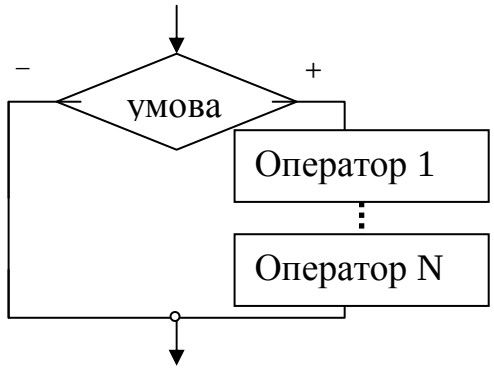
Повна форма

Програмна структура	**Аналог на мові блок-схем**	**Приклад**

if (<умова>)
{
<оператор1.1>;
. . . . . .
<оператор1.N>;
}
else
{
<оператор2.1>;
. . . . . .
<оператор2.M>;
}
		`if(x>y)`
		`{`
		`  max=x;`
		`  min=y;`
		`}`
		`else`
		`{`
		`  max=y;`
		`  min=x;`
		`}`

Скорочена форма

Програмна структура	**Аналог на мові блок-схем**	**Приклад**

if (<умова>)
{
<оператор 1>;
. . . . . .
<оператор N>;
}
		`if(x>0)`
		`{`
		`  z=1/x;`
		`  l=y/x;`
		`}`

Приклад. Знайти максимальне та мінімальне із двох дійсних чисел.

```
#include <vcl.h>
#include <iostream>
using namespace std;
int main(int argc, char* argv[])
{  double a,b;
   cout<<"a=";
   cin>>a;
   cout<<"b=";
   cin>>b;
   double max,min;
   if (a>b)
    {
      max=a;
      min=b;
    }
   else
    {
      max=b;
      min=a;
    }
```

```
    cout<<"max="<<max<<endl;
    cout<<"min="<<min<<endl;
    system("pause");
        return 0;
}
```

Результати роботи програми

```
a=3
b=8
max=8
min=3
```

Умовний оператор (?:)

Якщо значення виразу може дорівнювати одному із двох значень в залежності від виконання чи невиконання деякої умови, то можна скористатися умовним оператором **(?:)** .

<умова>?<значення1>:<значення 2>

Якщо умова виконується (логічний вираз має значення true), то результатом виразу є **значення1**, інакше – **значення2.**

Часто цей оператор використовують у правій частині оператора присвоювання.

Загальний вигляд умовного оператора **(?:)**	Аналог з використанням умовного оператора if

<змінна>=**<умова>?<значення1>:<значення 2>;**	if (<умова>)
	  <змінна>=**<значення 1>;**
	else
	 <змінна>=**<значення 2>;**

Приклад. Знайти максимальне із двох дійсних чисел.

З використанням умовного оператора **(?:)**	З використанням умовного оператора if

max=(x>y)? x : y ;	if (x>y)
	  max=x;
	else
	  max=y;

Оператор вибору switch

Оператор switch дозволяє передавати керування одному з декількох операторів в залежності від значення виразу, який називають селектором вибору. У якості селектора вибору може бути вираз цілого типу, типу char, типу або типу enum.

Загальна форма	**Приклад.** Вводиться оцінка – цифра, вивести оцінку прописом (селектор вибору цілого типу).

switch (<селектор вибору>)
{
case <константа 1> : <оператор 1>;
	#include <vcl.h>
	#include <iostream>
	using namespace std;
	int main(int argc, char* argv[])
	{
	  int mark;
	  cout<<"mark=";
	  cin>>mark;
	  switch (mark)
	  {
	  case 2: cout<<"Незадовільно";

```
                        break;
   case <константа 2> : <оператор 2>;
                        break;
   ……..
   case <константа N> : <оператор N>;
                        break;
  default : <оператор N+1>;
            break;
}
```

```
            break;
       case 3: cout<<"Задовільно.";
            break;
       case 4: cout<<"Добре";
            break;
       case 5: cout<<"Відмінно";
            break;
      default: cout<<"Неправильна оцінка.";
            break;
      }

    system("pause");
      return 0;
}
```

Якщо для декількох варіантів необхідно виконати одні і ті ж оператори, то ці оператори вказують тільки для одного з варіантів, а для всіх інших не вказуємо ні необхідних операторів, ні операторів `break.`

Приклад. З клавіатури вводиться оцінка у національній шкалі, необхідно вивести повідомлення про те, чи зараховано студенту залік.

```
#include <vcl.h>
#include <iostream>
using namespace std;
int main(int argc, char* argv[])
{
   int mark;
   cout<<"mark=";
   cin>>mark;
       switch (mark)
       {
        case 1:
        case 2: cout<<"незараховано ";
                break;
        case 3:
        case 4:
        case 5: cout<<"зараховано";
                break;
       }

    system("pause");
        return 0;
}
```

Приклад. З клавіатури вводиться буква у нижньому регістрі, з'ясувати, чи є буква голосною. При розв'язанні цього завдання використаємо оператор switch, у якому селектор вибору та константи вибору типу char.

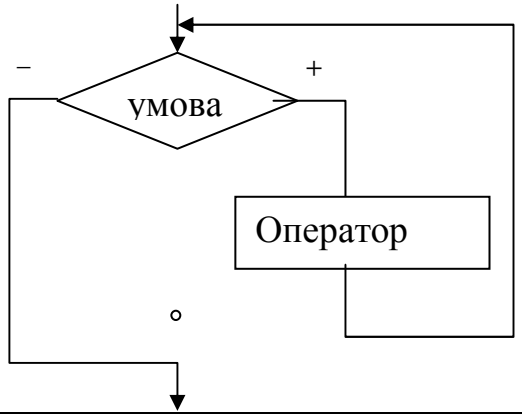
```
#include <vcl.h>
#include <iostream>
using namespace std;
int main(int argc, char* argv[])
{  char c;
   cout<<"с=";
   cin >> c;
       switch (c)
       {
        case 'а':
        case 'о':
        case 'у':
        case 'и':
        case 'і':
        case 'е': cout<<"голосна";
                  break;
        default: cout<<"приголосна";
                  break;
       }
    system("pause");
        return 0;
}
```

Оператори циклу

Оператор циклу while

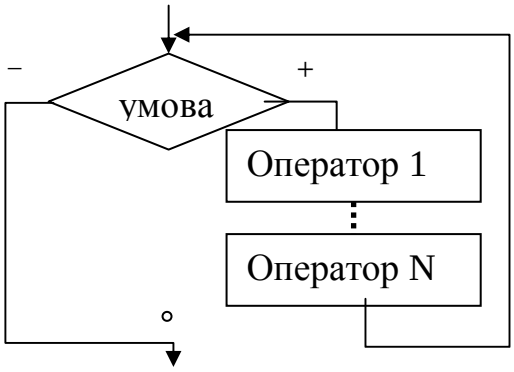
Оператор while циклічно виконує свою інструкцію до тих пір, поки умова виконується (логічний вираз приймає значення true).

Програмна структура	**Аналог на мові блок-схем**	**Приклад.** Знайти суму перших n чисел.

while (<умова>)
<оператор>;
		int sum=0;
		int i=1;
		while(i<=n)
		sum=sum+i++;

Якщо тіло циклу складається з більше ніж одного оператора, то необхідно використати складений оператор.

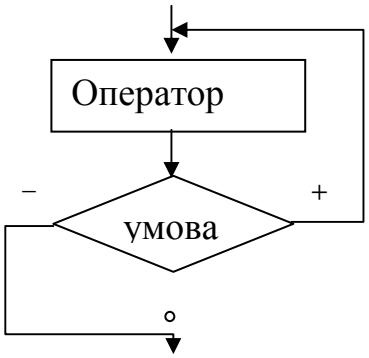
Програмна структура	**Аналог на мові блок-схем**	**Приклад.** Знайти суму і добуток перших n чисел.

while (<умова>)
{
<оператор 1>;
. . . . . .
<оператор N>;
}
		int sum=0;
		int mult=1;
		int i=1;
		while(i<=n)
		{
		sum=sum+(i++);
		mult=mult*(i++);
		}

Оператор циклу do-while

Оператор циклу do-while відрізняється від оператора while тим, що перевірка умови виконується не до, а після виконання інструкції (оператора).

Програмна структура	**Аналог на мові блок-схем**	**Приклад.** Знайти суму перших n чисел.

do
{
<оператор>;
}
while (<умова>)
		int sum=0;
		int i=1;
		do
		{
		sum=sum+i++;
		}
		while(i<=n)

Оператор циклу for

Загальна форма оператора наступна

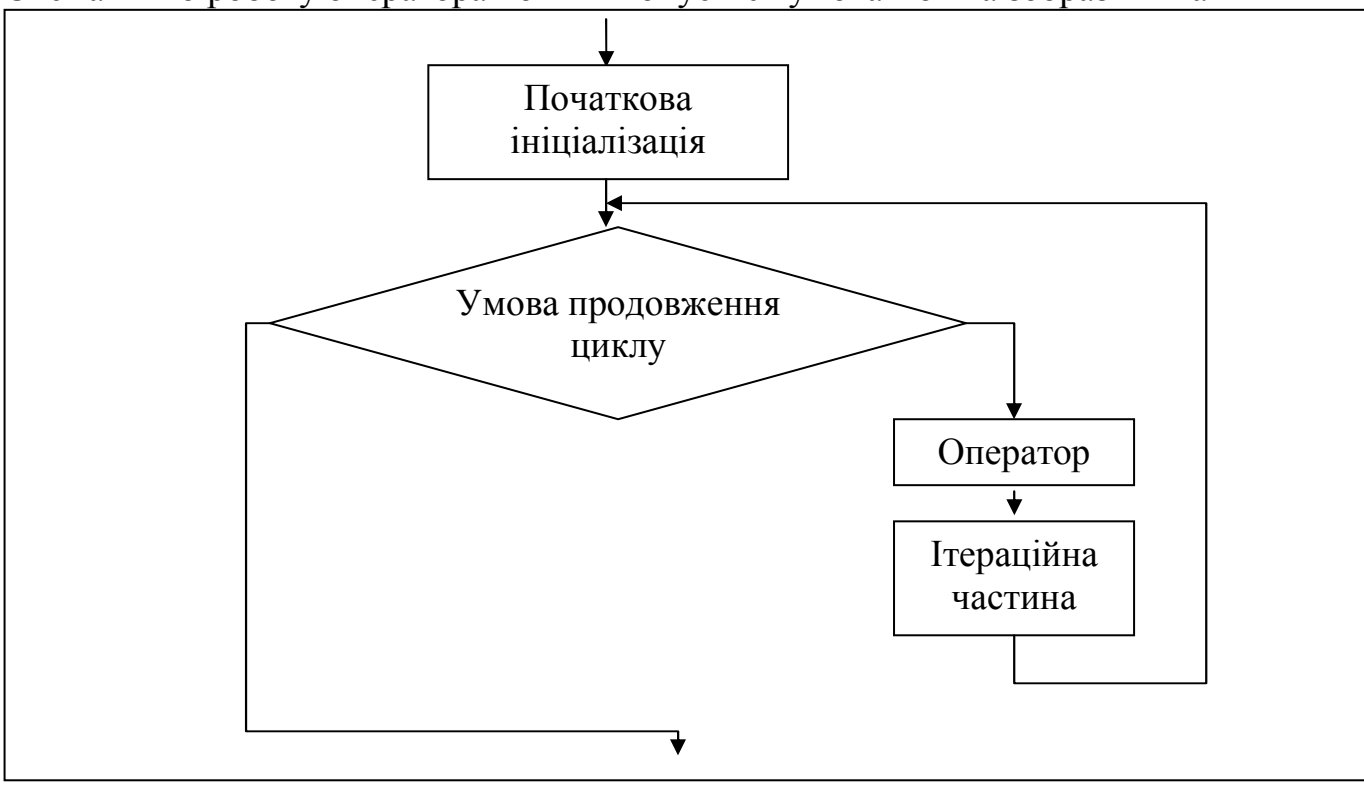
for (<вираз ініціалізації>;<логічна умова>;<ітераційна частина>)
<оператор>;

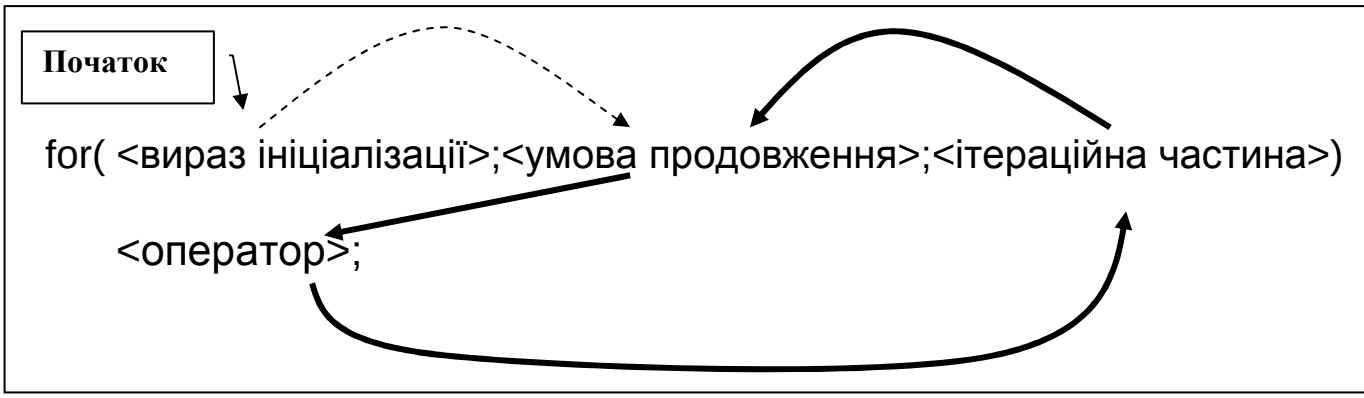
Оператор for виконує наступні дії:

1. Обчислює вираз ініціалізації. У цій частині допустима ініціалізація декількох лічильників циклу.
2. Перевіряється логічна умова. Якщо умова невірна то робота циклу завершується і передається управління наступному оператору.

3. Якщо умова істинна, виконується інструкція даного оператора.
4. Виконується приріст одного або декількох лічильників цикл (або виконується довільна інша операція.
5. Здійснюється перехід до кроку 2.

Схематично роботу оператора поки виконується умова можна зобразити так





Приклад. Знайти суму перших n натуральних чисел.

```
#include <vcl.h>
#include <iostream>
using namespace std;
int main(int argc, char* argv[])
{
   int n;
   cout<<"n=";
   cin >> n;
   int s=0;

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

```
    s=s+i;

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

    system("pause");
        return 0;
}
```

Проілюструємо випадок, коли частина ініціалізації та ітераційна частина мість декілька виразів, розділених комою.

Приклад. Обчислити значення виразу

$$\frac{1}{0,3}+\frac{2}{0,4}+...+\frac{n}{0,3+0.1*(n-1)}$$

```
#include <vcl.h>
#include <iostream>
using namespace std;
int main(int argc, char* argv[])
{

   int n;
   cout<<"n=";
   cin>>n;
   double d;
   int c;
   double sum=0;

   for (c = 1, d=0.3 ; c <= n ; c++, d+=0.1)
    sum +=c/d;

   cout<<"sum="<<sum<<endl;

    system("pause");
        return 0;
}
```

Кожна з частин оператора циклу може бути відсутньою, але розділові знаки “ ; ” є обов’язковими. Так нескінчений цикл може бути задано так

```
    for (  ;  ;  )
        <оператор>;
```

Оператори break та continue

Оператор continue може бути використаний у будь-якому із циклів у випадку, коли немає потреби виконувати всі оператори у тілі циклу у поточній ітерації, а необхідно одразу перейти до наступної ітерації.

Приклад. Знайти добуток непарних натуральних чисел, що менші за K.

```
#include <vcl.h>
#include <iostream>
using namespace std;
int main(int argc, char* argv[])
{

   int K;
   cout<<"K=";
   cin>>K;

   int mult=1;
   for (int i = 1;i<=K ; i++)
   {
     if ((i % 2) == 0) continue;
       mult *= i;
   }

   cout<<"mult="<<mult<<endl;

    system("pause");
        return 0;
}
```

Оператор break у циклах використовують для негайного завершення самого внутрішнього циклу, у тілі якого він знаходиться.

Приклад. Знайти найменше значення факторіалу натурального числа, що перевищує число K.

```
#include <vcl.h>
#include <iostream>
using namespace std;
int main(int argc, char* argv[])
{
    int K;
    cout<<"K=";
    cin>>K;

    int fakt=1;
    for (int i = 1; ; i++)
    {
      fakt *= i;
      if (fakt > K) break;
    }
    cout<<"fakt="<<fakt<<endl;
    system("pause");
        return 0;
```

```
}
```

Приклад. Вивести на екран усі натуральні двоцифрові числа, у яких друга цифра не перевищує першу.

При розв'язанні цього завдання використаємо оператор break , який здійснює вихід тільки із вкладеного циклу (робота зовнішнього циклу продовжується).

```
#include <vcl.h>
#include <iostream>
using namespace std;
int main(int argc, char* argv[])
{
    for (int i = 1; i <= 9; i++)
    {
      for (int j = 1; j <= 9; j++)
      {
        if (j > i) break;
          cout<<i<<j<<endl;
      }
    }

    system("pause");
        return 0;
}
```

Результат роботи програми

```
11
21
22
31
32
33
..
99
```

Масиви

Можна дати декілька означень масиву.

Масив – це структура даних, що являє собою однорідну (за типом), фіксовану (за розміром і конфігурацією) сукупність елементів, упорядкованих за номерами.

Масив – це нумерована послідовність елементів одного типу.

Масиви відносяться до структур з прямим або довільним доступом. Щоб визначити окремий елемент масиву потрібно вказати його *індекси*. У якості індексів використовуються значення цілого типу. В одновимірному масиві індекс – це порядковий номер елементу масиву. Нумерація елементів завжди починається з нуля. Кількість індексів називають *розмірністю*, кількість дозволених значень кожного індексу – *діапазоном*, а сукупність розмірності та діапазону – *формою масиву*.

Класифікація масивів може здійснюватися за багатьма ознаками:

1) можливістю зміни значень елементів масиву (масиви-константи та масиви-змінні);

2) кількістю індексів, які необхідно вказати для того, щоб ідентифікувати елемент у масиві (масиви одновимірні, двовимірні, тривимірні і т.д.);

3) можливістю зміни кількості елементів масиву в процесі виконання програми (статичні та динамічні масиви).

Статичні масиви

Одновимірні масиви

Описуючи статичний одновимірний масив, наперед необхідно вказати максимально допустиму кількість елементів.

Загальний вигляд	Приклад
<Тип> <Ім'я масиву> [<Кількість елементів>]	int a [3]; double b[7];

Описуючи масив його можна одразу ініціалізувати, тобто надати елементам масиву початкові значення. Для цього використовують *ініціалізатор масиву* – у фігурних дужках через кому наводиться список початкових значень елементів.

Загальний вигляд	Приклад
<Тип><Ім'я масиву>[<Кількість елементів>]= { <список значень> };	int a [3]={2,8,5};

Якщо кількість значень є меншою ніж кількість елементів масиву, то всі інші елементи до кінця масиву заповнюються нульовими значеннями.

Зауважимо, що у випадку, якщо ініціалізація елементів масиву здійснюється одразу після його опису і кількість значень співпадає з кількістю елементів масиву, то кількість елементів масиву під час його опису вказувати не обов'язково.

Загальний вигляд	Приклад
<Тип><Ім'я масиву>[ ]= { <список значень> };	int a [ ]={2,8,5};

Для ідентифікації елемента масиву необхідно вказати ім’я масиву та індекс елемента, який записується у квадратних дужках після імені масиву

```
<ім’я масиву>  [ <індекс> ]
```

Приклад.

```
a [ 0 ] = 2;
a [ 1 ] = 9;
a [ 2 ] = a[ 0 ]+3;
```

Наведемо декілька важливих зауважень:

1) елементи масиву можуть бути довільного, але тільки одного типу;

2) діапазон не може змінюватися під час виконання програми;

3) значення індексу не повинно виходити за межі описаного діапазону;

4) оскільки процедури введення/виведення розраховані на введення та виведення значень простих типів, то масив потрібно вводити/виводити поелементно, тобто кожен елемент окремо.

Як було зауважено, межі зміни діапазону індексів повинні бути константами, і діапазон не може змінюватися під час виконання програми. Але ж при розв’язанні задач, як правило, кількість елементів масиву вводить користувач під час виконання програми. Тому в залежності від задачі при описі масиву необхідно вказати гранично можливий діапазон зміни індексу. Для збереження ж справжнього діапазону зміни індексу описують додаткові змінні, значення яких не можуть виходити за вказані гранично можливі.

Приклад. Якщо необхідно зберігати роки народження учнів у класі (учнів не більше 40), то масив описують так

```
int Y[40];
```

Для збереження дійсної кількості учнів у класі описуємо додатково змінну цілого типу n, значення якої буде задавати користувач під час виконання програми. Зрозуміло, що значення змінної n у даному випадку не може бути більшим за 40.

```
int Y[40];        // Y  – масив років народження
int n;            // n – кількість учнів у класі ( n ≤ 40 )
```

Тут

Y[0] – рік народження 1-го учня;

Y[1] – рік народження 2-го учня;

……

$Y[n-1]$ – рік народження n-го учня.

Наведемо текст програми, що знаходить рік народження найстаршого учня.

```
#include <vcl.h>
#pragma hdrstop
#pragma argsused
#include <iostream>
using namespace std;
int main(int argc, char* argv[])
{
```

```
   int Y[40];        //Опис масиву
   int n;            //Кількість елементів масиву

   cout<<"n=";       //Введення кількості елементів масиву
   cin>>n;
                     //Введення елементів масиву
   for(int i=0;i<n;i++)
   {
      cout<<"Y["<<i<<"]=";
      cin>>Y[i];
   }
   int min=Y[0];          //Знаходження наймешого елемента масиву
   for(int i=1;i<n;i++)
   {
      if (Y[i]<min) min=Y[i];
   }

   cout<<"Рік нар. найстаршого учня ="<<min<<endl;
   system("pause");
        return 0;
}
```

Приклад. Необхідно знайти загальну масу автомобілів у парку.

Будемо вважати, що кількість автомобілів не перевищує 100.

```
double C[100];          // C–масив для збереження маси автомобілів
int n;                  // n – кількість автомобілів (n ≤ 100)
```

Тут

C[0] – маса 1-го автомобіля;

C[1] – маса 2-го автомобіля;

……

C[$n-1$] – маса n-го автомобіля.

Приклад. Дано $a,b \in R^n$, знайти $c = a + b$.

Розв'язання

Для збереження векторів використаємо однойменні одновимірні масиви a, b і c. Будемо вважати, що $n \le 50$.

```
#include <vcl.h>
#pragma hdrstop
#pragma argsused
#include <iostream>
using namespace std;
int main(int argc, char* argv[])
{  int a[50];          //Опис масивів
   int b[50];
   int c[50];
   int n;
```

```
   cout<<"n=";
   cin>>n;
                        //Введення масиву a
   cout<<"------- Input a ----------"<<endl;
   for(int i=0;i<n;i++)
   {
      cout<<"a["<<i<<"]=";
      cin>>a[i];
   }
                        //Введення масиву b
   cout<<"------- Input b ----------"<<endl;
   for(int i=0;i<n;i++)
   {
      cout<<"b["<<i<<"]=";
      cin>>b[i];
   }
                        //Знаходження масиву c
   for(int i=0;i<n;i++)
   {
      c[i]=a[i]+b[i];
   }
                        //Виведення масиву c
   cout<<"------- Output c ----------"<<endl;
   for(int i=0;i<n;i++)
   {
      cout<<"c["<<i<<"]="<<c[i]<<endl;
   }
   system("pause");
        return 0;
}
```

Представлення одновимірного масиву в пам'яті ЕОМ

Для збереження масивів в пам'яті ЕОМ використовується послідовне представлення. Тобто для збереження масиву наперед виділяється ділянка оперативної пам'яті величиною

<кількість елементів> * <кількість байтів для збереження одного елемента>

і елементи розміщуються послідовно один за одним.

Приклад.

```
int A[10];
```

Масив A в пам'яті ЕОМ буде розміщено наступним чином

…	A[0]	A[1]	A[2]	…	A[9]	…	**Елементи масиву**
	Область масиву A						

Одновимірні масиви-константи

У мові С++ є можливість описання масиву-константи. При цьому необхідно вказати всі елементи описуваного масиву.

const <тип елементів> <ім'я масиву> **[**<кільк. елем.>**]** = {<елемент 0>,<елемент 1 >,…,<елемент N -1>};

Приклад.

```
const int v[3]={10,21,37};
```

Спроба змінити значння масиву-константи призведе до помилки компіляції.

Багатовимірні масиви

У С++ також є можливість описувати багатовимірні масиви

<Тип><Ім'я масиву>[<Кількість елементів 1>][<Кількість елементів 2>] … [<Кількість елементів N>];

Найчастіше використовують двовимірні масиви, які асоціюють з матрицями (що складаються з рядків і стовпців).

Загальний вигляд	Приклад
<Тип><Ім'я масиву>[<Кількість рядків>][<Кількість стовпців>]	int a [3][2];

Як і одновимірні масиви, двовимірні масиви можна ініціалізувати одразу після їх опису.

Загальний вигляд	Приклад
<Тип><Ім'я масиву>[<Кільк. рядків>][<Кільк.стовпців>]= { <Список значень> };	int a [3][2]={2,8, 7,4, 7,5};
<Тип><Ім'я масиву>[<Кільк. рядків>][<Кільк.стовпців>]= { {<Список значень рядка1>},{<Список значень рядка2>}, … };	int a [3][2]={ {2,8}, {7,4}, {7,5} };
<Тип><Ім'я масиву> [ ] [<Кількість стовпців>]= { {<список значень рядка1>},{<список значень рядка2>}, … }; // Можна не вказувати кількість рядків	int a [ ] [2]={ {2,8}, {7,4}, {7,5} };

Для ідентифікації елементу масиву необхідно вказати ім'я та індекси

<ім'я масиву> [<індекс 1>][<індекс2>] …[<індекс N>]

Приклад.

a[2][1] – елемент двивимірного масиву a, що знаходиться у рядку під номером 2, і стовпці під номером 1.

Представлення статичного двовимірного масиву в пам'яті ЕОМ

Для збереження статичних двовимірних масивів, як і для одновимірних, в пам'яті ЕОМ використовується послідовне представлення. При цьому наперед виділяється ділянка оперативної пам'яті величиною

<кількість рядків> ***<кількість стовпців>** * < **розмір одного елемента масиву** >

і елементи розміщуються послідовно, один за одним. Спочатку елементи першого рядка, потім другого, третього і т.д.

Приклад.

```
int A[3][2];
```

Змінну A

$$A = \begin{pmatrix} a_{0,0} & a_{0,1} \\ a_{1,0} & a_{1,1} \\ a_{2,0} & a_{2,1} \end{pmatrix}.$$

у пам'яті ЕОМ буде розміщено наступним чином:

								Комірки
…	A[0,0]	A[0,1]	A[1,0]	A[1,1]	A[2,0]	A[2,1]	…	**Елементи масиву**
	Рядок 1		Рядок 2		Рядок 3			
	Область масиву A							

Покажчики

Покажчик – це змінна, яка може містити адресу оперативної пам'яті. Як правило, покажчики використовують для роботи з динамічною пам'яттю та описанні формальних параметрів підпрограм, що можуть змінюватися у цій підпрограмі. Опис покажчика містить символ «*».

Загальний вигляд	Приклад
<Тип> ***** <Покажчик>;	int *p;

Знак «*» можна вказувати як біля типу даних, так і біля покажчика. Як правило, його вказують біля покажчика.

Покажчик може містити адресу іншої змінної (для взяття адреси використовується оператор «&»). При цьому тип змінної повинен співпадати з типом, який використано при описі покажчика покажчика.

Загальний вигляд	Приклад

< Покажчик > = &<Змінна >;	int *p; //Опис покажчика на тип int
	int d; //Опис змінної типу int
	p=&d; //Покажчик p одержує адресу змінної d

Доступ до ділянки пам'яті, адресу якої містить покажчик здійснюється через операцію розіменування (символ «*» ставиться перед покажчиком).

Загальний вигляд	Приклад

* < Покажчик > = <значення >;	int k;
	int *m=&k;
	*m=27;
	double r;
	double *d=&r;

	*d=2.5;

Адресна арифметика. При збільшенні (зменшенні) покажчика на деяке ціле число count значення покажчика (адреса) збільшується (зменшується) на величину, що дорівнює добутку

count*sizeof (<Тип покажчика>)

Оператор	Результат
int *p; p=p+1; // Або ж p++	Значення покажчика (адресу) p збільшили на 1*sizeof(int)=4 байти
int *p; p=p+3; // Або ж p++	Значення покажчика (адресу) p збільшили на 3*sizeof(int)=12 байтів
double *d; d=d+3;	Значення покажчика (адресу) d збільшили на 3*sizeof(double)=24

Виділення динамічної пам’яті

Динамічна пам'ять може бути виділена за допомогою оператора new і звільнена за допомогою delete, або ж виділена за допомогою malloc і звільнена за допомогою оператора free.

Загальний вигляд (new – delete)	Приклад

<покажчик> = new <тип покажчика>; //Виділення пам'яті
……. //Використання пам’яті
delete <покажчик>; //Звільнення пам'яті
	int *p;
	p= new int;
	…….
	delete p;

Функції malloc та free описано у файлі заголовку cstdlib

void *malloc(size_t Size);

void free (void *Memory);

тому для їх використання необхідно цей файл заголовка підключити. Оскільки функція malloc типу void*, то при її використанні необхідно виконувати приведення типу до необхідного типу даних. Зауважимо, що у різних операційних системах кількість байтів, що виділяється для типів даних може бути різною, тому, щоб програма була незалежною від операційної системи, при виділенні пам’яті необхідно використовувати функцію sizeof.

Загальний вигляд (malloc – free)	Приклад

//Виділення пам'яті
<Покажчик>=(Тип *) malloc(<Кількість байтів>);
……. //Використання пам’яті
//Звільнення пам'яті
free ((void *) <Покажчик>);
	#include <cstdlib>
	int *p;
	
	p=(int*)malloc(sizeof(int));
	…….
	
	free((void *) p);

Посилання

Посилання – це змінні, які можуть бути псевдонімами для інших змінних. При цьому змінні та посилання на них (їх псевдоніми) звертаються до однієї і тієї ж комірки пам’яті. Вони, як правило, використовуються при

описі формальних параметрів функцій, щоб уникнути виділення додаткової пам'яті для збереження копії фактичних параметрів(значення цих змінних у функції можуть змінюватися). Змінній-посиланню можна присвоїти деяке значення тільки при її описі. Тип змінної, для якої посилання є псевдонімом, повинен співпадати з типом змінної-посилання. При описі посилань використовується символ «&».

Загальний вигляд	Приклад

<Тип> &<Посилання> = <Змінна >;	int d;
	int &n=d; //n є посиланням на змінну d
	
	n=2; // Еквівалентно d=2

Динамічні масиви

Динамічні масиви – це масиви, пам'ять для яких виділяється у процесі роботи програми. Динамічне виділення пам'яті здійснюється з використанням покажчиків.

Загальний вигляд (new-delete)	Приклад

<Тип елемента масиву> * <Покажчик>;
 	
<Покажчик>=**new** <Тип елем. масиву>[<К-сть елементів>]
……..
delete [ ] <Покажчик>;
	int *p;
	int Size; //Кільк. елем.
	cout<<"Size=";
	cin>>Size;
	p=new int [Size];
	……
	delete [ ] p;

Загальний вигляд (malloc-free)	Приклад

<Тип елементів масиву> * <Покажчик>;
 	
//Виділення пам'яті для масиву
<Покажчик>=(Тип елем.*)**malloc**(<Кільк. елем.>*sizeof(<Тип елем.>));
 	
…….. //Використання масиву
free((void *) <Покажчик>); //Звільнення пам'яті
	#include <cstdlib>
	…….
	int *p;
	int Size; //Кільк. елем.
	cout<<"Size=";
	cin>>Size;
	p=(int*)**malloc**(count *
	sizeof(int));
	……
	free((void *) p);

Доступ до елементів динамічного масиву можна здійснювати по різному.

Загальний вигляд	Приклад
<Покажчик> [<Номер елемента>]	p[i] =8;

<Номер елемента> [<Покажчик>]	i [p] = 8;
*(<Покажчик> + <Номер елемента>)	*(p+i) =8;
*(<Номер елемента> + <Покажчик>)	*(i+p) =8;

Наведемо приклад використання динамічних масивів.

Приклад. Задано послідовність цілих чисел $a_1, a_2, ..., a_n$. Знайти найбільший елемент цієї послідовності.

new-delete

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

using namespace std;
int main(int argc, char* argv[])
{       int n;
                //Введення кількості елементів
        cout<<"n= ";        cin>> n;
                        //Виділення пам'яті
        int *p=new int[n];

                        //Введення елементів
        for(int i=0;i<n;i++)
        {       cout<<"a["<<i<<"]=";
                cin>>p[i];
        }
                //Знаходження максимального
        int max=p[0];
        for(i=0;i<n;i++)
        {     if(max<p[ i ])
                  max=p[ i ];
        }
        cout<<"max="<<max<<endl;
                        //Звільнення пам'яті
        delete[ ] p;

       system("pause");
        return 0;
}
```

malloc-free

```
#include "stdafx.h"
#include <iostream>
#include <cstdlib>
using namespace std;
int main(int argc, char* argv[])
{       int n;
                //Введення кількості елементів
        cout<<"n= ";        cin>> n;
                        //Виділення пам'яті
      int *p=(int*) malloc(n*sizeof( int ));

                        //Введення елементів
        for(int i=0;i< n;i++)
       {   cout<<"a["<<i<<"]=";
            cin>>p[i];
        }
                //Знаходження максимального
        int max=p[0];
        for(i=0;i<n;i++)
        {       if(max<p[ i ])
                    max= p[ i ];
         }
        cout<<"max="<<max<<endl;
                        //Звільнення пам'яті
        free((void *) p);
       system("pause");
        return 0;
}
```