Тема 1. Вступ. Історія розвитку мови програмування C++.

Вступ. Мета курсу та його зв'язок з іншими курсами. Структура курсу. Прикладний та теоретичний аспект програмування, їх взаємозв'язок. Екскурс в історію розвитку програмування. Структуроване та об'єктно-орієнтоване програмування.

Програмування і алгоритмічні мови (C++)

Вид контролю: залік.

Мета та завдання дисципліни

Мета - формування теоретичних знань та практичних навиків з розробки програмних додатків на мові програмування високого рівня C++. Розкриття можливостей об'єктно орієнтованого програмування.

Завдання дисципліни – набуття практичних навиків з розробки сучасних програмних додатків на мові програмування C++. Підготовка студентів до курсів «Обчислювальна математика» та «Мікроконтролери», які вимагають знань мови програмування C++.

В результаті вивчення даного курсу студент повинен:

знати: синтаксис мови програмування C++, можливості та засоби об'єктно-орієнтованого програмування;

вміти: самостійно розробляти програмні додатки, застосовувати мову програмування C++ для розв'язання прикладних задач математики і фізики.

Література.

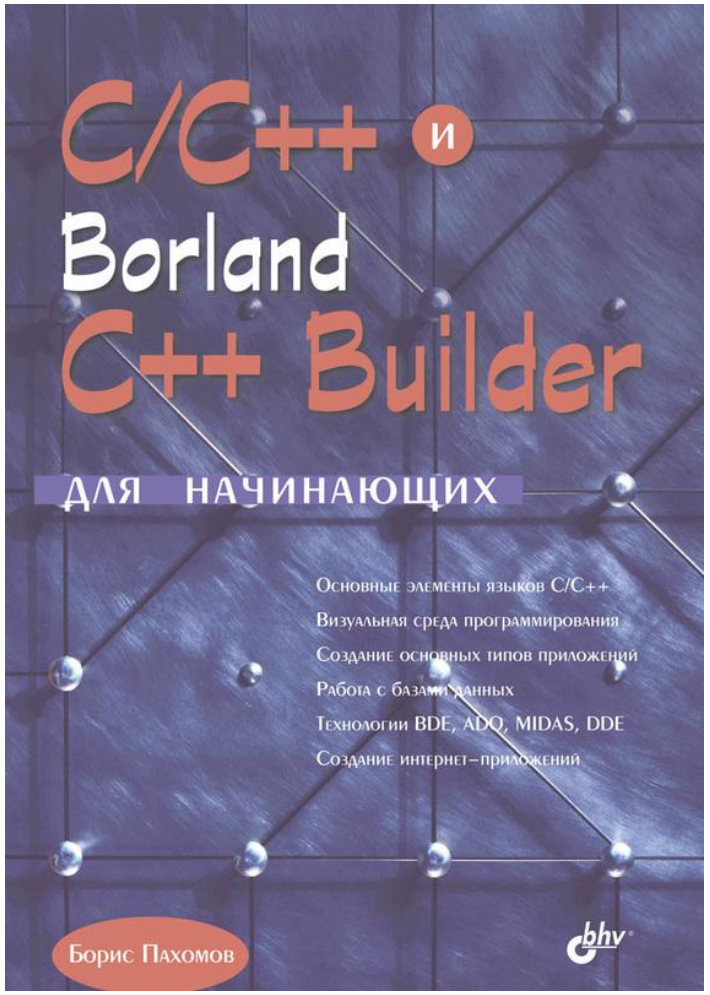
1. Борис Пахомов, C/C++ и Borland C++ Builder для начинающих / - БХВ-Петербург, 2005, 636 с.

2. Архангельский А.Я. Приемы программирования C++ Builder 6 и 2006 / А.Я. Архангельский. – М.: «Бином», 2006. – 991с.

3. Д. Элджер, Библиотека программиста C++ / Элджер Д. - Питер, 1999. – 320 с.

4. William H. Numerical Recipes in C: The Art of Scientific Computing. Cambridge: Cambridge University Press, 1992. ISBN 0-521-43108-5. 994 P.

Література



Достатньо для самостыйного вивчення дисципліни.
Містить реальні робочі приклади.

Зв’язок з іншими дисциплінами.

1. Літня обчислювальна практика (екзамен, оцінка йде у 1 семестр 2 курсу).
2. Чисельні методи (залік, 1 семестр, 2 курс).
3. Мікроконтролери (екзамен, 1 семестр, 2 курс).

Екскурс в історію

Мова С створена на початку 70 х років, коли Кен Томпсон і Денніс Рітчі з Bell Labs розробляли операційну систему UNDC. Спочатку вони створили частину компілятора С, потім використовували її для компіляції решти компілятора С та, нарешті, застосували отриманий в результаті компілятор для компіляції UNIX. Операційна система UNIX спочатку поширювалася у вихідних кодах на С серед університетів і лабораторій, а користувач міг відкомпілювати вихідний код на С в машинний код за допомогою відповідного компілятора С.

Поширення вихідного коду зробило операційну систему UNIX унікальною; програміст міг змінити операційну систему, а вихідний код міг бути перенесений з одного апаратної платформи на іншу. С був третьою мовою, яку розробили Томсон і Рітчі в процесі створення UNIX; першими двома були, зрозуміло, A і B.

http://www.interestprograms.ru/articles/historyprogramming/historycpp

Екскурс в історію

У порівнянні з більш ранньою мовою - BCPL, **C** була поліпшена шляхом додавання типів даних певної довжини. Наприклад, тип даних *int* міг застосовуватися для створення змінної з певною кількістю бітів (зазвичай 16), в той час як тип даних *long* міг використовуватися для створення цілої змінної з великим числом бітів (зазвичай 32). На відміну від інших мов високого рівня, C міг працювати з адресами пам'яті безпосередньо за допомогою покажчиків і посилань. Оскільки **C** зберегла здатність прямого доступу до апаратного забезпечення, її часто відносять до мов середнього рівня або в жарт називають "мобільною мовою асемблера".

Що стосується граматики і синтаксису, то **C** є структурною мовою програмування. У той час як сучасні програмісти мислять категоріями класів і об'єктів. В **C** можна визначити власні абстрактні типи даних, використовуючи ключове слово *struct*. Аналогічно, можна описувати власні цілі типи (перерахування) і давати інші назви існуючих типів даних за допомогою ключового слова *typedef*. У цьому сенсі C є структурним мовою з зародками об'єктно-орієнтованого програмування.

Екскурс в історію (стадартизація)

Широке поширення мови **C** на різних типах комп'ютерів привело, на жаль, до багатьох варіацій мови. Вони були схожі, але несумісні один з одним. Це було серйозною проблемою для розробників програм, які потребували написання сумісних програм, які можна було б виконувати на різних платформах. Стало зрозуміло, що необхідна стандартна версія **C**. У 1983 році ANSI (Американський Національний Комітет Стандартів) сформував технічний комітет X3J11 для створення стандарту мови **C** (щоб "забезпечити недвозначне і машинно-незалежне визначення мови"). У 1989 стандарт був затверджений. ANSI скооперувався з ISO (Міжнародна Організація Стандартів), щоб стандартизувати C в міжнародному масштабі; спільний стандарт був опублікований в 1990 році і названий ANSI / ISO 9899: 1990. Цей стандарт вдосконалюється досі і підтримується більшістю фірм розробників компіляторів.

Екскурс в історію

Бйорн Страуструп вивільнив об'єктно-орієнтована потенціал **C** шляхом перенесення можливостей класів Simula 67 в **C**. Спочатку нова мова носила ім'я "C з класами" і тільки потім стала називатися **C++**. Мова **C++** досягла популярності в Bell Labs, пізніше вона була перенесена в інші індустрії і корпорації.

Сьогодні це одна з найбільш популярних мов програмування в світі.

Екскурс в історію (цитати)

Бйорн Страуструп: "Я придумав **C++**, записав його первинне визначення і виконав першу реалізацію. Я вибрав і сформулював критерії проектування **C++**, розробив її основні можливості і відповідав за долю пропозицій по розширенню мови в комітеті по стандартизації **C++**", - пише автор найпопулярнішого мови програмування.

"Мова **C++** багато чим зобов'язана мові **C**, і мова **C** залишається підмножиною мови **C++** (але в **C++** усунені кілька серйозних недоліків системи типів **C**). Я також зберіг засоби **C**, які є досить низькорівневими, щоб справлятися з самими критичними системними завданнями. Мова **C**, в свою чергу, багато чим зобов'язана своєму попередникові, BCPL; до речі, стиль коментарів «//» був узятий до C++ з BCPL. Іншим основним джерелом натхнення була мова Simula 67. Концепція класів (з похідними класами і віртуальними функціями) була запозичена з нього. Засоби перевантаження операторів і можливість приміщення об`явлень в будь-якому місці, де може бути записана інструкція, нагадує Algol 68. "

Екскурс в історію

Назву **C++** вигадав Рік Масіті. Назва вказує на еволюційну природу переходу до нього від **C**. "++" - це операція збільшення в **C**.

int C = 0;

C++; //постінкримент

Спочатку мова програмування **C++** була розроблена, щоб автору і його друзям не доводилося програмувати на асемблері, **C** або інших мовах високого рівня. Основним її призначенням було зробити написання хороших програм більш простим і приємним для окремого програміста.

Перед тим як починати освоювати **C++**, Страуструп і більшість інших програмістів, що використовують **C++** вважають вивчення мови **C** необов'язковим.

Мова програмування **C++** на даний час вважається панівною мовою, що використовується для розробки комерційних продуктів, 90% ігор пишуться на **C++** з використанням DirectX або OpenGL.

Тема 2. Елементи й основні поняття мови С ++. Типи даних.

Символи, що використовуються для утворення ключових слів і ідентифікаторів.

Знаки нумерації та спеціальні символи. Керуючі та розділові символи. Керуючі послідовності.

Етапи створення програм

1. ***Системний аналіз***. в рамках цього етапу здійснюється аналіз вимог до програмної системи. Він проводиться на основі первинного дослідження всіх потоків інформації при традиційному проведенні робіт і здійснюється в наступній послідовності:
а) уточнення видів і послідовності всіх робіт;
б) визначення цілей, які повинні бути досягнуті програмою, що розробляється;
в) виявлення аналогів, що забезпечують досягнення подібних цілей, їх переваг і недоліків.

2. ***Зовнішнє специфікування***. Полягає у визначенні зовнішніх специфікацій, тобто описів вхідної і вихідної інформації, форм їх подання та способів обробки інформації. Реалізується в наступній послідовності:
а) постановка завдання на розробку нової програми;
б) оцінка досяжності цілей розроблюваного програмного продукту.
Далі, при необхідності, етапи 1-2 можуть бути повторені до досягнення задовільного вигляду програмної системи з описом виконуваних нею функцій.

Етапи створення програм

3. ***Проектування програми.*** На цьому етапі проводиться комплекс робіт з формування опису програми. Вихідними даними для цієї фази є вимоги, викладені в специфікації, розробленої на попередньому етапі. Приймаються рішення, що стосуються способів задоволення вимог специфікації. Ця фаза розробки програми ділиться на два етапи:

а) архітектурне проектування. Являє собою розробку опису програми в найзагальнішому вигляді. Цей опис містить відомості про можливі варіанти структурної побудови програмного вироби (або у вигляді декількох програм, або у вигляді декількох частин однієї програми), а також про основні алгоритмах, і структурах даних. Результатом цієї роботи є остаточний варіант архітектури програмної системи, вимоги до структури окремих програмних компонент та організації файлів для міжпрограмного обміну даними;

б) робоче проектування. На цьому етапі архітектурний опис програми деталізується до такого рівня, який робить можливими її реалізацію (кодування та збірку). Для цього здійснюється складання і перевірка специфікацій модулів, складання описів логіки модулів, складання остаточного плану реалізації програми.

Етапи створення програм

4. ***Кодування і тестування.*** Ці види діяльності реалізуються для окремих модулів і сукупності готових модулів до отримання готової програми.
5. ***Комплексне тестування.***
6. ***Розробка експлуатаційної документації.***
7. ***Коригування програм***. Проводиться за результатами попередніх випробувань.
8. ***Тиражування.***
9. ***Супровід програми.*** У поняття «супровід» входять всі технічні операції, необхідні для використання даної програми в робочому режимі. Сюди входить не тільки виправлення помилок. На цьому етапі також здійснюється модифікація програми, внесення виправлень у робочу документацію, удосконалення програми та ін. Внаслідок широких масштабів подібних операцій супровід є ітеративним процесом, який бажано здійснювати не тільки після, але й до випуску програмного виробу. Роботи з супроводу часто поглинають більше половини витрат, що припадають на весь життєвий цикл програмної системи. Сучасні технології проектування програмного забезпечення спрямовані на часткову автоматизацію описаних вище етапів і на суміщення їх у часі з метою скорочення термінів виконання проектів.

Трансляція програм

Транслятор - програма або технічний засіб, що виконує трансляцію програми.

Трансляція програми - перетворення програми, представленої на одній з мов програмування, в програму на іншій мові і, в певному сенсі, рівносильну першої [Wikipedia].

Асемблер - це транслятор, у якого вихідним мовою є символічне уявлення машинного коду (асемблер), а об'єктним мовою є якийсь різновид машинного мови будь-якого реального комп'ютера.

Компілятор - транслятор, для якого вихідною є мова високого рівня, а його об'єктна мова близька до машинної мови реального комп'ютера. Це або мова асемблера, або який-небудь варіант машинної мови. Наприклад, програми на мові **C** компілюються, як правило, в програми на мові асемблер, які потім транслюються асемблером в машинну мову.

Зазвичай, ми абстрагуємося від процесів, які відбуваються під час збірки програми. Але при заглибленні знань та програмуванні на професійному рівні, процес створення додатків потребує більшого розуміння усіх процесів.

Типи даних

Специфікатор типу	байт	Діапазон значень
цілочисельний (логічний) тип даних		
bool	1	0 / 255
цілочисельний (символьний) тип даних		
char	1	0 / 255
цілочисельні типи даних		
short int	2	-32 768 / 32 767
unsigned short int	2	0 / 65 535
int	4	-2 147 483 648 / 2 147 483 647
unsigned int	4	0 / 4 294 967 295
long int	4	-2 147 483 648 / 2 147 483 647
unsigned long int	4	0 / 4 294 967 295
типи даних з плаваючою точкою		
float	4	-2 147 483 648.0 / 2 147 483 647.0
long float	8	-9 223 372 036 854 775 808 .0 / 9 223 372 036 854 775 807.0
double	8	-9 223 372 036 854 775 808 .0 / 9 223 372 036 854 775 807.0

Тип bool

bool – цілочисельний тип даних, діапазон допустимих значень – цілі числа від 0 до 255. Так як **bool** використовується виключно для зберігання результатів логічних виразів. У логічного виразу може бути один з двох результатів true або false. true – якщо логічний вираз є позитивним, false – якщо логічний вираз є негативним.

Приклади ініціалізації.

```
bool a; //a – не визначена

bool a = true;
bool a = 1;
bool a = 255;



bool a = false;
bool a = 0;
```

Тип char

Тип даних char – це цілочисельний тип даних, який використовується для представлення символів. Тобто, кожному символу відповідає певне число з діапазону [0;255]. Тип даних char також ще називають символьним типом даних, так як графічне представлення символів в С можливо завдяки char. Для представлення символів в С типу даних char відводиться один байт, в одному байті – 8 біт, тоді зведемо двійку в ступінь 8 і отримаємо значення 256 – кількість символів, яке можна закодувати. Таким чином, використовуючи тип даних char можна відобразити будь-який з 256 символів. Всі закодовані символи представлені в таблиці ASCII.

Приклади об'яви та ініціалізації.

```
char symbol;  //об`ява змінної symbol
symbol = ’a’;  // ініціалізація змінні

char symbol = ’a’;
```

Цілочисельні типи

В таблиці її наведено шість штук: short int, unsigned short int, int, unsigned int, long int, unsigned long int**.** Всі вони мають свій власний розмір займаної пам'яті і діапазон значень. Залежно від компілятора, розмір займаної пам'яті і діапазон значень можуть змінюватися.

Приклади об'яви та ініціалізації.

```
int k;  //об`ява змінної k
k = -10;  // ініціалізація змінної

int k = -10;
```

Типи даних з плаваючою точкою

float, double.

Синтаксис мови встановлює, що десятковий роздільник – крапка.

Приклади об'яви та ініціалізації.

```
float k;  //об`ява змінної k
k = -10.1 ;  // ініціалізація змінні

float k = -10.1;
```

Також можливий запис:

```
float x = 1.1565E12;
```

Константи

У мові **C/C++** поділяють чотири типи констант: цілі константи, константи з плаваючою комою, символьні константи і рядкові.

Змінна будь-якого типу може бути оголошена як така, що не модифікується. Це досягається додаванням ключового слова **const** до специфікатора типу. Об'єкти з типом **const** являють собою дані, які використовуються тільки для читання, тобто цієї змінної не може бути присвоєно нове значення. Відзначимо, що якщо після слова **const** відсутній специфікатор-типу, то мається на увазі специфікатор типу **int**. Якщо ключове слово **const** стоїть перед оголошенням складових типів (масив, структура, суміш, перерахування), то це призводить до того, що кожен елемент також повинен бути немодифікованим, тобто значення йому може бути присвоєно лише один раз.

const double pi = 3.1415926358;

Операції

знак операції	операція	Група операцій
*	множення	мультиплікативні
/	ділення	
%	залишок від ділення	
+	додавання	адитивні
-	віднімання	
<<	зсув вліво	операції зсуву
>>	зсув вправо	
<	менше	операції відносини
<=	менше або дорівнює	
>=	більше або дорівнює	
==	дорівнює	
!=	не дорівнює	

Операції

&	порозрядне І	порозрядні операції
|	порозрядне АБО	
^	порозрядне виключне АБО	
&&	логічне І	логічні операції
||	логічне АБО	
,	послідовне обчислення	послідовного обчислення

Операції

=	присвоєння	операції присвоювання
*=	Множення з привласненням	
/=	Розподіл з привласненням	
%=	Залишок від ділення з привласненням	
-=	Віднімання з привласненням	
+=	Додавання з привласненням	
<<=	Зрушення вліво з привласненням	
>>=	Зрушення вправо привласненням	
&=	Порозрядне І з привласненням	
|=	Порозрядне АБО з привласненням	
^=	Порозрядне виключає АБО з привласненням	

```
int m = 1;
m = m*3;  еквівалентний запис m *= 3;
m = m+3;  еквівалентний запис m += 3;
m = m/3;  еквівалентний запис m /= 3;
```

Мультиплікативні операції

До цього класу операцій відносяться операції множення (*), ділення (/) та отримання залишку від ділення (%). Операндами операції (%) повинні бути цілі числа.
Відзначимо, що типи операндів операцій множення і ділення можуть відрізнятися, і для них справедливі правила перетворення типів. Типом результату є тип операндів після перетворення.
Операція множення (*) виконує множення операндів.

```
int  i=5;
float f=0.2;
double g, z;
g=f*i;
```

Типи змінних і та f перетворяться до типу double, потім результат присвоюється змінній g.
Операція ділення (/) виконує ділення першого операнда на другий. Якщо дві цілі величини не діляться без остачі, то результат округлюється в бік нуля.

Адитивні операції

До адитивних операцій відносяться додавання (+) і віднімання (-). Операнди можуть бути цілого або плаваючого типів. У деяких випадках над операндами адитивних операцій виконуються спільні арифметичні перетворення.
Перетворення, що виконуються при адитивних операціях, не забезпечують обробку ситуацій переповнення і втрати значущості. Інформація втрачається, якщо результат адитивної операції не може бути представлений типом операндів після перетворення. При цьому повідомлення про помилку не видається.

Приклад:

```
unsigned char i = 255, j = 1, k;
k = i + j;
```

В результаті складання k одержить значення рівне 0.

Операції зсуву

Операції зсуву здійснюють зсув операнда вліво (<<) або вправо (>>) на число бітів, що задається другим операндом. Обидва операнди повинні бути цілими величинами. Виконуються звичайні арифметичні перетворення. При зсуві вліво праві звільняються біти встановлюються в нуль. При зсуві вправо метод заповнення вивільнюваних лівих бітів залежить від типу першого операнда.

Перетворення, виконані операціями зсуву, не забезпечують обробку ситуацій переповнення і втрати значущості.

Відзначимо, що зрушення вліво відповідає множенню першого операнда на ступінь числа 2, рівну другого операнд, а зрушення вправо відповідає поділу першого операнда на 2 в ступені, що дорівнює другого операнд.

Приклад:

```
int  i=0x1234, j,  k ;
k = i<<4 ;        /*    k = 0x0234   */
j = i<<8 ;        /*    j = 0x3400   */
i = j>>8 ;        /*    i = 0x0034   */
```

Порозрядні операції

Операнди порозрядних операцій можуть бути будь-якого **цілого** типу. Операція поразрядного логічного І (&) порівнює кожен біт першого операнда з відповідним бітом другого операнда. Якщо обидва порівнюваних біта одиниці, то відповідний біт результату встановлюється в 1, в іншому випадку в 0.

Операція поразрядного логічного АБО (|) порівнює кожен біт першого операнда з відповідним бітом другого операнда. Якщо будь-який (або обидва) з порівнюваних бітів дорівнює 1, то відповідний біт результату встановлюється в 1, в іншому випадку результуючий біт дорівнює 0.

Операція поразрядного виключає АБО (^) порівнює кожен біт першого операнда з відповідними бітами другого операнда. Якщо один з порівнюваних бітів дорівнює 0, а другий біт дорівнює 1, то відповідний біт результату встановлюється в 1, в іншому випадку, тобто коли обидва біти рівні 1 або 0, біт результату встановлюється в 0.

Приклад.

```
int  i=0x45FF;    // i= 0100 0101 1111 1111
int  j=0x00FF;    // j= 0000 0000 1111 1111
```

```
int r;
 r = i^j;     //r=0x4500  = 0100 0101 0000 0000
 r = i|j;     //r=0x45FF = 0100 0101 0000 0000
 r = i&j;     //r=0x00FF = 0000 0000 1111 1111
```

Логічні операції

До логічних операцій відносяться операція логічного І (&&) і операція логічного АБО (||). Операнди логічних операцій можуть бути цілого типу, плаваючого типу або типу покажчика, при цьому в кожній операції можуть брати участь операнди різних типів.

Операнди логічних виразів обчислюються зліва направо. Якщо значення першого операнда достатньо, щоб визначити результат операції, то другий операнд **не** обчислюється.

Логічні операції не викликають стандартних арифметичних перетворень. Вони оцінюють кожен операнд з точки зору його еквівалентності нулю. Результатом логічної операції є 0 або 1, тип результату int.

Операція логічного І (&&) виробляє значення 1, якщо обидва операнда мають нульові значення. Якщо один з операндів дорівнює 0, то результат також дорівнює 0. Якщо значення першого операнда дорівнює 0, то другий операнд НЕ обчислюється.

Операція логічного АБО (||) виконує над операндами операцію включає АБО. Вона виробляє значення 0, якщо обидва операнда мають значення 0, якщо який-небудь з операндів має нульове значення, то результат операції дорівнює 1. Якщо перший операнд має нульове значення, то другий операнд НЕ обчислюється.

Умовні операції

Умовні операції використовуються сумісно з операторами порівняння та циклу.

Приклади:

```
int a = 1;
int b = 0;
```

```
if(a >= b){
  //так
}else{
 //ні
}
```

```
if(a == b){
  //так
}else{
 //ні
}
```

```
if(a != b){
  //так
}else{
 //ні
}
```

Операції збільшення та зменшення

Операції збільшення (++) і зменшення (--) є унарними операціями присвоєння. Вони відповідно збільшують або зменшують значення операнда на одиницю. Операнд може бути цілого або плаваючого типу або типу покажчик і повинен бути модифікується. Операнд цілого або плаваючого типу збільшуються (зменшуються) на одиницю. Тип результату відповідає типу операнда. Операнд адресного типу збільшується або зменшується на розмір об'єкта, який він адресує. У мові допускається префіксна або постфіксний форми операцій збільшення (зменшення), тому значення виразу, що використовує операції збільшення (зменшення) залежить від того, яка з форм зазначених операцій використовується.

Якщо знак операції стоїть перед операндом (префиксная форма запису), то зміна операнда відбувається до його використання у виразі і результатом операції є збільшене або зменшене значення операнда.

У тому випадку якщо знак операції стоїть після операнда (Постфіксний форма запису), то операнд спочатку використовується для обчислення виразу, а потім відбувається зміна операнда.

Операції збільшення та зменшення

Приклади:

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

Спочатку відбувається множення t * 5, а потім збільшення t. В результаті вийде z = 5, t = 2.

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

Спочатку значення s збільшується, а потім використовується в операції ділення. В результаті отримаємо s = 3, f = 1.

У випадку, якщо операції збільшення та зменшення використовуються як самостійні оператори, префиксная і Постфіксний форми запису стають еквівалентними.

```
z ++; / * Еквівалентно * / ++ z;
```

Оператори вибору

У мові програмування С ++ існує два оператора вибору:

1) Оператор вибору **if**

2) Оператор вибору **switch**

Оператори вибору дозволяють прийняти програмі рішення, грунтуючись на істинності або хибності умови. Якщо умова є істиною (true), оператор в тілі if виконується, після чого виконується наступний по порядку оператор. Якщо умова хибна (false), оператор в тілі if не виконується (ігнорується або пропускається) і відразу ж виконується наступний оператор.

```
a=5; b = 7;
if (a > b){
    // код невиконується
}
```

Оператори вибору

```
// синтаксис оператора выбору switch
switch (a){
  case 0:
  //код
  break;

  case 1:
  //код
  break;

  . . .

  default:
  //код
}
```

a – змінна
break; - оператор виходу
case 0: - випадок, якщо a = 0.
case 1: - випадок, якщо a = 1.