Лекція 2

Тема лекції: Коментарі C++. Вступ до потоків в/в. Область бачення та оголошення змінних. Вбудовані функції. Керування пам'яттю за допомогою NEW та DELETE

Вступ до C++. Необ'єктні властивості C++

У назві мови C++ використовується оператор інкременту мови C. Дослівно C++ - наступний крок за C. Винахідник її - Б'єрн Страуструп назвав цю мову "поліпшена C". Мова C++ зробила значний внесок до поняття класу - особливої структури, членами якої можуть бути як дані, так і функції. Крім класів, у C++ додані декілька нових ключових слів і операторів, вбудовані та перевантажені функції, перевантаження операторів, нова техніка керування пам'яттю та інші засоби. Програми на C та C++ дуже схожі. В них приблизно однаковий синтаксис, ті самі види циклів, типи даних, покажчики та ін. Для того, щоб зазначити компілятору, яку з мов він повинен обробити, програмам, написаним на C++ привласнюється розширення .CPP. Заголовочним файлам можна давати розширення .HPP, хоча компілятор не робить розходжень між .H і .HPP.

Приклад програми на C++:

```
#include <iostream. h>
main()
{
  cout << "Welcome to Borland C++! \n";
  return 0;
}
```

У цій програмі, по-перше, включається заголовочний файл *iostream.h* замість *stdio.h*. По-друге, використовується стандартний потік виведення об'єктів *cout* (character out або "виведення символів") бібліотеки потоків введення/виведення для виведення тексту на дисплей. Для того, щоб направити інформацію на екран, бібліотека потоків введення/виведення використовує

оператор C++ << (направити в). Можна також використовувати складений оператор потоків введення/виведення. Якщо *FirstName* і *LastName* мають тип покажчиків на рядок *char**, то й вираз

```
cout << LastName << ",” << FirstName <<'\n';
```

виводить два імені, що розділені комою й закінчуються переходом на наступний рядок. Можливе написання кожної частини складеного оператора *iostream* з нового рядка. Наприклад:

```
cout << LastName
<< ",”
<< FirstName
<< '\n';
```

Бібліотека потоків введення/виведення спроможна виводити різні типи даних, включаючи вбудовані (*int, long, double*), а також типи, створені програмістом. Якщо *count* має тип *long*, тоді вираз

```
cout << "count==" << count << '\n';
```

виведе рядок, що завершується значенням count і переходом на наступний рядок.

У C++ можливо використовувати методи *stdio.h* і бібліотеки потоків введення/виведення разом, а також використовувати *printf()* і аналогічні функції. Але функція *printf()* може маніпулювати тільки вбудованими типами даних і своїми форматами виведення, які не можна запрограмувати.

Основні відмінності C та C++

- В C++ були введені додаткові ключові слова: *catch, class, delete, friend, inline, new, operator, private, protected, public, template, this, throw, try, virtual*.
- C++ потребує, щоб прототипи усіх функцій були визначені до їхнього виклику. У C++ здійснюється більш сувора перевірка типів у виразах.
- Значення у виразах повинні бути або однакового типу, або легко претворюватися на відповідний тип. Там, де C видає попередження про невідповідність типів, C++ видає помилку компіляції.
- У C++ дуже послаблена потреба в *typedef*-оголошеннях. Наприклад,

можна оголосити структуру типу

```
struc mystruct {...  };
```

і потім оголошувати змінні як

```
mystruct x;
```

• В С потрібно використовувати *typedef* для структури *mystruct* (або *struct mystruct x*).

• В С символьні константи мають тип *int*, і вираз *sizeof('x')* еквівалентний до *sizeof(int*). В С++ символьні константи мають тип *char*. І *sizeof('x')* еквівалентно *sizeof(char)*.

Цей список не є повним.

Коментар С++

У програмі С++ '//' (подвійна риса) означає коментар. Займає він один рядок. Можна використовувати і коментар С. Всередину оператора можна вставити тільки коментар С. Наприлад:

```
result = count /*Вкладений коментар*/ + 100;
```

В одній і тій самій програмі одночасно можуть вживатися коментарі С и С++.

Вступ до потоків введення/виведення

Введення та виведення в С++ не є частиною мови, а забезпечується зовнішніми бібліотечними модулями. Тому програмісти можуть створювати свої функції і засоби введення/виведення. Потоки *cout* і *cin* мають багато можливостей. Щоб відобразити символ p, можна написати:

```
cout << p;//або
cout.put(p);
```

Функція *put()* - член потоку *cout*. Потік *cout* - приклад класу, що має стосовні до нього функції. Потрібно викликати функцію-член тільки з посиланням на об'єкт, подібний *cout*. Без цього вираз

```
put(p);
```

спробує викликати незалежну функцію *put()*. Оператор

```
cout.put(p);
```

викликає функцію *put()*, що належить *cout*.

Для того, щоб прочитати символ зі стандартного потоку введення, треба використовувати потік введення *cin*, що стоїть перед оператором >> (узяти з). Оператор

```
cin >> p;
```

читає один символ зі стандартного потоку введення й привласнює його змінній *p*. Можна також викликати функцію-член *get* із *cin*:

```
cin.get(p);
```

Бібліотека потоків введення/виведення може виводити символьні рядки. Наприклад, наступне оголошення привласнює покажчику на рядок адресу рядка символів:

```
char *s = "символи";
```

Оператор

```
cout << s;
```

відправить даний рядок у системний стандартний потік виведення, зазвичай на дисплей. Для читання рядка провадиться зворотній процес.

Область бачення та оголошення змінних

Неоднозначності області бачення змінних можуть виникати в програмах, написаних як на С так і на С++. Наприклад, якщо існує глобальна змінна *int COUNT*, функція може оголосити змінну з тим самим ім'ям, не викликавши при цьому повідомлення про помилку:

```
int COUNT;     //глобальна змінна
void AnyFunction()
{
 int COUNT; ... .
}
```

Область бачення локальної змінної *COUNT* діє в межах оголошеної функції. У цій функції неможливий доступ до глобальної змінної *COUNT*. Для вирішення цієї проблеми в С++ використовується оператор розширення області

бачення: “::”. Перепишемо функцію в такому вигляді:

```
int COUNT;
void AnyFunction()
{
   int COUNT;
   COUNT = 2;
   ::COUNT = 3;
}
```

Тут локальній змінній *COUNT* привласнюється значення 2, а глобальній - 3. Ілюструється ще одна властивість C++: оголошення в C++ можна вводити в будь-якій точці програми, а не тільки глобально або на початку функції, як у C. Змінні, оголошені всередині блоку-оператора, існують тільки в межах цього блоку.

Наприклад:

```
if (ex)
{
  int COUNT = 0;……..
}
cout << COUNT << '\n';
```

При компіляції останній рядок викликає повідомлення про помилку, оскільки змінна *COUNT* оголошена поза областю бачення оператора *if*, але вона не існує поза нею. Буває корисно одночасно оголошувати та ініціалізувати змінну, що керує циклом for. Наприклад:

```
void f( )
{
  for(int i=0;i<max;i++)
…….
  }
```

Таке оголошення діє не тільки в межах бачення циклу *for*. Ініціалізація керуючої змінної виконується до того, як починається цикл. У даному прикладі *i* потрапляє до області бачення *f()* і може використовуватися будь-якими операторами, що йдуть за оператором for.

Константи

Оголошення змінних із ключовим словом *const* захищає їх від зміни під час виконання програми. Наприклад:

```
const int count = 1234;
```

Компілятор видасть повідомлення про помилку в операторі

```
count++;
```

Рекомендується використовувати оголошення *const* замість символьних констант, створених за допомогою *#define*. Символьна константа

```
#define MAX 100
```

визначає макрос з ім'ям *MAX*, пов'язаний із текстом 100 (не цілим значенням 100). Якщо *MAX* використовується таким чином:

```
for(int i=0;i<MAX;i++)
...;
```

то компілятор замінить *MAX* на текстові цифри 100. Якщо оголосити *MAX* так:

```
const int MAX = 100;
```

то цикл *for* залишиться таким самим, а оголошення *MAX* дійсним значенням константи дасть такі переваги:

- Компілятор зможе виконати перевірку типів із *MAX* більш суворо. C++ розпізнає, що константа *MAX* має тип *int*, у той час як компілятор нічого не знає про тип текстового макроса *MAX*.
- *Turbo Debugger* розпізнає дійсне значення *константи* MAX, але не розпізнає символьні константи, створені за допомогою #define.

Вбудовані функції

На виклик функцій витрачається час. Численні виклики погіршують загальну швидкість програми. Вирішення цієї проблеми - вбудовані функції. Щоб оголосити функцію вбудованою, потрібно використовувати ключове слово *inline*. Наприклад:

```
inline int func(a,b);
```

Зазвичай вбудовані функції оголошуються в заголовочних файлах і

включаються туди, де необхідно їхнє використання. Вбудовану функцію необхідно цілком визначити до того, як можна буде її використовувати і включення її тексту до заголовочного файлу - найпростіший засіб задоволення даної вимоги. У скомпільованому коді програми Borland C++ не викликає вбудовану функцію в рядку, де вона записана. Замість цього компілятор вставляє тіло функції безпосередньо до тіла програми. Але, якщо код вбудованої функції занадто великий, то компілятор може відмовитися вбудовувати функцію в потік коду і замість цього згенерує звичайний виклик. Також при компіляції програм Turbo Debugger всі вбудовані функції будуть перетворені на звичайні функції, що викликаються.

Керування пам'яттю за допомогою операторів new і delete

Можна використовувати однакову техніку керування пам'яттю в програмах на C та на C++. Але C++ пропонує альтернативні оператори керування пам'яттю *new* і *delete*. Використання цих операторів замість стандартних функцій керування пам'яттю дає можливість при необхідності самостійно обробляти ситуацію керування пам'яттю. Щоб виділити пам'ять для змінної, що має тип double і привласнити її адресу покажчику, можна написати:

```
double *dp = new double;
```

Покажчик *dp* використовується так само, як і будь-який інший покажчик на пам'ять, виділену функцією *malloc()* або подібною функцією. Закінчивши використання динамічної змінної, потрібно видалити її таким чином:

```
delete dp;
```

Видалення покажчика означає звільнення виділеної раніше пам'яті, адреса якої зберігається в *dp*, для наступного використання. Для звільнення пам'яті, виділеної за допомогою *new*, необхідно використовувати тільки *delete*.

Виявлення помилок пов'язаних із нестачею пам'яті

При створенні динамічних змінних необхідно з'ясувати, чи достатньо

вільної пам'яті. Якщо операція *new* завершилася успішно, результатом її є покажчик на зарезервовану пам'ять необхідного розміру. У випадку невдалого завершення, можливі такі ситуації:

- *new* збуджує виключну ситуацію.
- *new* повертає нуль.

Щоб не збуджувати виключну ситуацію, потрібно включити заголовочний файл *new.h* до програми і додати в *main()* або в будь-яке інше місце до використання *new,* оператор:

```
set_new_handler(0);
```

При завершенні програми уся виділена пам'ять автоматично звільняється. Проте, потрібно видаляти динамічні змінні після закінчення їхнього використання. Також бажано видаляти покажчик, навіть якщо він дорівнює нулю. Видалення того ж самого покажчика повторно або повторне використання адресованої ним пам'яті може викликати серйозні помилки. Видаливши покажчик, ні в якому разі не можна використовувати його ні для яких цілей, крім адресації пам'яті, наново виділеної new. Оператор *new* можна використовувати для динамічного виділення пам'яті для рядка будь-якого розміру. Наприклад:

```
#include <iostream. h>
#define SIZE 80
main()
{
  char *sp;
  sp = new char[SIZE];        //Виділяємо символьний масив
  cout << "String? ";         //розміром SIZE
  cin.getline(sp,SIZE);       //Привласнюємо адресу 1-го байту
sp
  cout << "You entered:" << sp << '\n';
  delete[] sp;
  return 0;
}
```

Можна оператор *delete* написати без дужок. Дужки необхідні лише в тому випадку, коли видаляється масив класових об'єктів. У C++ *new* і *delete* також

можуть використовуватися для створення і звільнення динамічних масивів будь-якого типу.