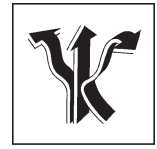
МІЖРЕГІОНАЛЬНА
АКАДЕМІЯ УПРАВЛІННЯ ПЕРСОНАЛОМ



О. В. ГАЛКІН, М. М. ВЕРЕС

МОВА ПРОГРАМУВАННЯ С++

Конспект лекцій

Київ
ДП «Видавничий дім «Персонал»
2015

Рецензенти: *М. Г. Глибовець,* д-р фіз.-мат. наук, проф.
В. Б. Зваридчук, канд. фіз.-мат. наук, доц.
В. П. Шевченко, канд. фіз.-мат. наук, доц.

Схвалено Вченою радою Міжрегіональної Академії управління персоналом (протокол № 1 від 26.01.11)

Галкін О. В., Верес М. М.

Мова програмування С++: конспект лекцій / О. В. Галкін, М. М. Верес. — К.: ДП "Вид. дім "Персонал", 2015. — 260 с. — Бібліогр.: с. 249.

ISBN 978-617-02-0194-2

У конспекті лекцій розглядаються всі основні можливості мови С++ і їх застосування при розробці об'єктно-орієнтованих програм. Дається короткий опис бібліотек мови С++, необхідних для створення типових програм. Систематизовано викладаються основні поняття й описуються можливості мови С++.

Для студентів вищих навчальних закладів, які вивчають програмування на мові С++.

ISBN 978-617-02-0194-2

ВСТУП

Мова C++ була створена на початку 80-х років XX ст. співробітником фірми Bell Laboratories Берном Страуструпом, як розширення мови С. До початку офіційної стандартизації мова розвивалася здебільшого силами Б. Страуструпа у відповідь на запити програмістського співтовариства. У 1998 р. був ратифікований міжнародний стандарт мови C++: ISO/IEC 14882:1998 "Standard for the C++ Programming Language"; після прийняття технічних виправлень до стандарту у 2003 р. — сучасна версія цього стандарту — ISO/IEC 14882:2003.

Ідея створення нової мови бере початок від мови моделювання Сімула (Simula). Вона мала низку можливостей, що були корисні для розробки великого програмного забезпечення, але працювала занадто повільно. У той самий час мова BCPL досить швидка, але занадто близька до мов низького рівня й не підходить для розробки великого програмного забезпечення. Б. Страуструп почав працювати у Bell Laboratories над задачами теорії черг (у додатку до моделювання телефонних викликів). Він вирішив доповнити мову С (спадкоємець BCPL) можливостями, наявними у мові Сімула. Мова С, яка є базовою мовою системи UNIX, швидка та багатофункціональна. Б. Страуструп додав їй можливість роботи із класами й об'єктами. У результаті практичні задачі моделювання виявилися доступними для розв'язання як з погляду часу розроблення (завдяки використанню Сімула-подібних класів), так і з погляду часу обчислень (завдяки швидкодії С).

Нововведеннями C++ порівняно із С є:

- підтримка об'єктно-орієнтованого програмування;
- підтримка узагальненого програмування через шаблони;
- додаткові типи даних;
- виключення;
- простір імен;
- вбудовані функції;
- перевантаження операторів;
- перевантаження імен функцій;

- посилання й оператори керування вільно розподіленою пам'яттю;
- доповнення до стандартної бібліотеки.

Отже, можна сказати, що мова C++ багато у чому є надмножиною мови *C*:

Варто також відзначити, що C++ має ряд недоліків, деякі з них успадковані від *C*:

- синтаксис, що провокує помилки, наприклад, операція присвоювання позначається як = , а операція порівняння як ==, їх легко переплутати; *if* (*x* = 0) {оператори}. Тут в умовному операторі помилково написане присвоювання замість порівняння. У результаті, замість того, щоб зрівняти поточне значення *x* з нулем, програма присвоїть *x* нульове значення;
- макроси (*#define*) є потужним, але небезпечним засобом. Вони збережені у C++, незважаючи на те, що в них немає необхідності завдяки шаблонам і вбудованим функціям. В успадкованих стандартних *C*-бібліотеках багато потенційно небезпечних макросів;
- деякі перетворення типів неінтуїтивні. Зокрема, операція над беззнаковим і знаковим числами видає беззнаковий результат;
- необхідність записувати *break* у кожному розгалужені оператора switch і можливість послідовного виконання кількох розгалужень при його відсутності провокує помилки через пропуск *break*. Ця сама особливість дає змогу робити сумнівні "трюки", що базуються на вибірковому незастосуванні *break*, і ускладнює розуміння коду.

Однак, незважаючи на свої недоліки, C++ є найпоширенішою мовою програмування для операційних систем Windows і Unix. Також, незважаючи на декларовану досконаліть мов C# і Java, витиснути C++ ним так і не вдалося.

Представлений нижче конспект лекцій є узагальненням курсу з об'єктно-орієнтованого програмування на базі мови C++, що протягом кількох років читався авторами студентам факультету кібернетики Київського національного університету.

ОСНОВНІ ТИПИ ДАНИХ МОВИ С++

Лекція 1. ТИПИ ДАНИХ У С++

У С++ є набір вбудованих типів даних для подання цілих і дійсних чисел, символів, а також тип даних "символьний масив", що служить для зберігання символьних рядків.

Тип *char* використовується для зберігання окремих символів та невеликих цілих чисел і займає 1 байт.

Типи *short*, *int* і *long* призначені для представлення цілих чисел. Вони розрізняються тільки діапазоном значень, які можуть приймати числа, а конкретні розміри перелічених типів залежать від реалізації.

Тип *short* займає половину машинного слова, *int* — одне слово, *long* — одне або два слова. У 32-бітних системах *int* і *long*, як правило, одного розміру.

Типи *float*, *double* і *long double* призначені для чисел із плаваючою крапкою й розрізняються точністю представлення (кількістю значущих розрядів) і діапазоном. Тип *float* (одинарна точність) займає одне машинне слово, *double* (подвійна точність) — два, а l*ong double* (розширена точність) — три слова.

Тип *bool* має два значення — *true* і *false* — і займає 1 біт.

Char, *short*, *int* і *long* разом складають цілі типи, які, в свою чергу, можуть бути знаковими (*signed*) і беззнаковими (*unsigned*). У знакових типах самий лівий біт використовується для зберігання знака (0 — плюс, 1 — мінус), а біти, що залишилися, містять значення. У беззнакових типах всі біти використовуються для значення. 8-бітовий тип *signed char* може представляти значення від -128 до 127, а *unsigned char* — від 0 до 255.

Коли у програмі зустрічається деяке число, наприклад 1, то це число називається літералом, або літеральною константою. Літерали цілих типів можна записати у десятковому, восьмеричному й шістнадцятиричному вигляді. Наприклад, число 20, представлене десятковим, восьмеричним і шістнадцятиричним літералами, має вигляд:

- *20 // десятковий;*
- *024 // восьмеричний;*
- *0x14 // шістнадцятиричний.*

Якщо літерал починається з 0, він трактується як восьмеричний, якщо з 0*x* або 0*X*, то як шістнадцятиричний.

За замовчуванням всі цілі літерали мають тип *signed int*. Можна явно визначити цілий літерал таким, що має тип *long*, приписавши наприкінці числа літеру *L* (використовується як прописна *L*, так і рядкова *l*). Буква *U* (або *u*) наприкінці визначає літерал як *unsigned int*, а дві літери — *UL* або *LU* — як тип *unsigned long*.

Наприклад:

128u 1024UL 1L 8Lu.

Літерали, що представляють дійсні числа, можуть бути записані як з десятковою крапкою, так і в науковій (експонентній) нотації. За замовчуванням вони мають тип *double*. Для явної вказівки типу *float* потрібно використовувати суфікс *F* або *f*, а для *long double* — *L* або *l*, але тільки у випадку запису з десятковою крапкою.

Наприклад:

3.14159F 0/1f 12.345L 0.0 3el 1.0 E-3E 2. 1.0L.

Спеціальні символи (табуляція, повернення каретки) записуються як *escape*-послідовності. Визначено наступні послідовності (вони починаються із символу зворотної косої риски):

- новий рядок *n*;
- горизонтальна табуляція *t*;
- забій *b*;
- вертикальна табуляція *v*;
- повернення каретки *r*;
- прогін аркуша *f*;
- дзвінок *a*;
- зворотна коса риска \\;
- питання \?;
- одиночні лапки \';
- подвійні лапки \";

- *escape* — послідовність загального вигляду має форму \ooo, де ooo — від одного до трьох восьмеричних цифр. Це число є кодом символу.

Наприклад:

- *\7 (дзвінок);*
- *\14 (новий рядок);*
- *\0 (null);*
- *\062 ('2').*

Змінні

Змінна, або об'єкт — це іменована область пам'яті, до якої ми маємо доступ із програми; туди можна записувати значення а потім їх використовувати. Кожна змінна C++ має певний тип, що характеризує розмір і розташування в області пам'яті, діапазон значень, який вона може зберігати, і набір операцій, які застосовуються до цієї змінної.

Наприклад:

int student_count;
double salary;
bool on_loan;
string street_address;
char delimiter.

Початкове значення може бути задане прямо в операторі визначення змінної. У C++ припустимі дві форми ініціалізації змінної — явна, з використанням оператора присвоювання:

int ival = 1024; string project = "Fantasia 2000";

і неявна, із завданням початкового значення у дужках:

int ival (1024); string project ("Fantasia 2000").

Обидва визначення еквівалентні.

Вбудовані типи даних мають спеціальний синтаксис для завдання нульового значення:

// ival одержує значення 0, a dval — 0.0
int ival = int();
double dval = double().

Зі змінною асоціюються дві величини:

1) власне значення, або *r*-значення (від *read value* — значення для читання), що зберігається у цій області пам'яті й притаманне як змінній, так і літералу;

2) значення адреси області пам'яті, що асоційована зі змінної, або *l*-значення (від *location value* — значення місця розташування) — місце, де зберігається *r*-значення, притаманне тільки об'єкту.

Наприклад, у виразі

ch = ch –'0';

змінна *ch* перебуває і ліворуч, і праворуч від символу операції присвоювання.

Праворуч розташоване значення для читання (*ch* і символьний літерал '0'): асоційовані зі змінної дані зчитуються з відповідної області пам'яті.

Ліворуч — значення місця розташування: в область пам'яті, асоційовану зі змінної *ch*, міститься результат віднімання.

Ім'я змінної, або ідентифікатор, може складатися з латинських літер, цифр і символу підкреслення. Прописні й малі літери в іменах розрізняються. Мова C++ не обмежує довжину ідентифікатора, однак користуватися занадто довгими іменами типу незручно. Ключові слова не можна використовувати як ідентифікатори.

Покажчики

Покажчик — це об'єкт, що містить адресу іншого об'єкта і дає змогу побічно маніпулювати цим об'єктом. Найчастіше покажчики використовуються для роботи з динамічно створеними об'єктами для побудови зв'язаних структур даних, таких як зв'язані списки й ієрархічні дерева, і для передачі у функції великих об'єктів — масивів і об'єктів класів.

Кожний покажчик асоціюється з деяким типом даних, причому їх внутрішнє подання не залежить від внутрішнього типу: і розмір пам'яті, що займає об'єкт типу покажчик, і діапазон значень у них однаковий. Різниця полягає в тому, як компілятор сприймає об'єкт, що адресується. Покажчики на різні типи можуть мати те саме значення, але область пам'яті, де розміщуються відповідні типи, може бути різною: покажчик на *int*, що містить значення

адреси 1000, вказує на область пам'яті 1000–1003 (у 32-бітній системі); покажчик на *double*, що містить значення адреси 1000, вказує на область пам'яті 1000–1007 (у 32-бітній системі).

Наприклад:

```
int *ip1, *ip2;
complex<double> *cp;
string *pstring;
vector<int> *pvec;
double *dp.
```

Розглянемо приклади використання покажчиків:

```
int ival = 1024;
//pi ініціалізовано нульовою адресою
int *pi = 0;
// pi2 ініціалізовано адресою ival
int *pi2 = &ival;
// правильно: pi і pi2 містять адресу ival
pi = pi2;
// pi2 містить нульову адресу
pi2 = 0;
// помилка: pi не може приймати значення int pi = ival;
double dval;
double *ps = &dval; // помилки компіляції, неприпустиме присвоювання типів даних: int* //<== double* pi = pd pi = &dval;
//void* може містити адреси будь-якого типу
void *pv = pi;
pv = pd;
// непряме присвоювання змінній ival значення ival2 *pi = ival2;
// непряме використання змінної ival як rvalue і lvalue
*pi = abs(*pi); // ival = abs(ival);
*pi = *pi + 1; // ival = ival + 1;
// C++ допускає використання покажчика на покажчик
int **ppi = &pi;
int *pi2 = *ppi;
cout << "Значення ival\n" << "явне значення: " << ival << "\n"
<< "непряма адресація: " << *pi << "\n"
<< "двічі непряма адресація: " << **ppi << endl.
```

Покажчики можуть бути використані в арифметичних виразах:

```
int i, j, k;
int *pi = &i; // i = i + 2;
*pi = *pi + 2; // збільшення адреси, що міститься в pi, на
2pi = pi + 2.
```

Специфікатор const

Специфікатор const використовується для оголошення константи:

Наприклад:

```
const int bufSize = 512;
// помилка: неініціалізована константа
const double pi.
```

Можна також використовувати покажчики на константи

```
const double *pc = 0;
const double minWage = 9.60;        // правильно: не можемо змінювати minWage за допомогою pc;
pc = &minWage;
double dval = 3.14;                 // правильно: не можемо змінювати minWage за допомогою pc, хоча dval і не константа;
pc = &dval;                         // правильно;
dval = 3.14159;                     //правильно;
*pc = 3.14159.                      // помилка.
```

Існують і константні покажчики

```
int errNumb = 0;
int *const currErr = &errNumb.
```

У цьому випадку *curErr* — константний покажчик на неконстантний об'єкт. Це значить, що ми не можемо присвоїти йому адресу іншого об'єкта, хоча сам об'єкт допускає модифікацію. Спроба присвоїти значення константному покажчику викличе помилку компіляції:

```
curErr = &myErNumb;                 // помилка.
```