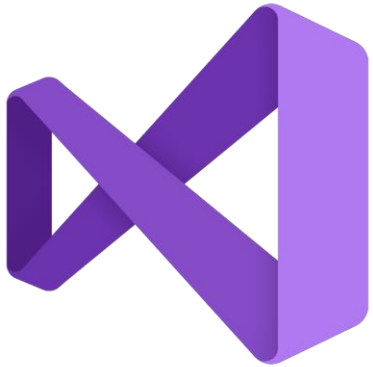
Информационные технологии и программирование

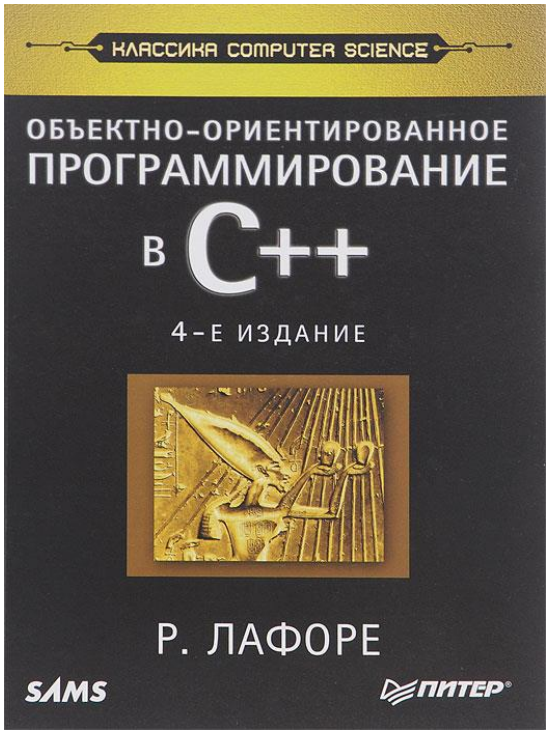


СПбГТИ(ТУ)
2022

Литература

Роберт Лафоре

«Объектно-ориентированное программирование в C++»



Литература

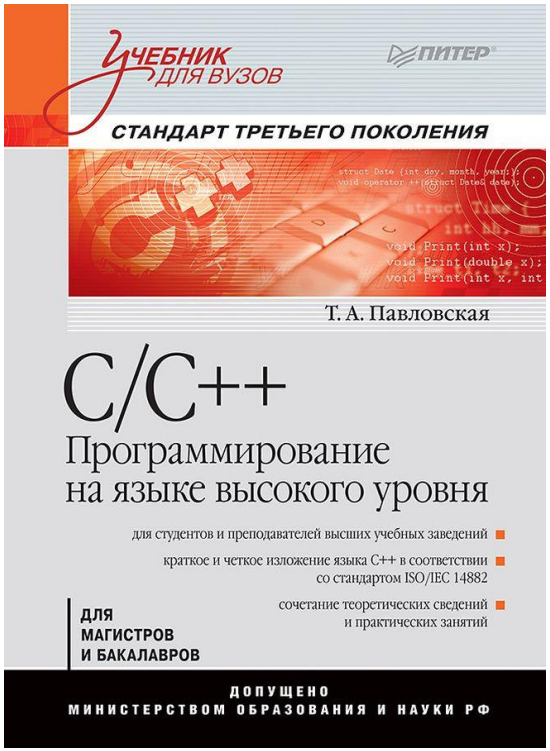
Стенли Б. Липпман

«Язык программирования C++. Базовый курс»

Литература

Татьяна Александровна Павловская

«C/C++. Программирование на языке высокого уровня»



Литература

Брюс Эккель

«Философия C++. Введение в стандартный C++»
«Философия C++. Практическое программирование»

Основные типы данных

Категория	Тип	Значение
Логический тип данных	`bool`	true или false
Символьный тип данных	`char, char16_t, char32_t`	Символ
Тип данных с плавающей запятой	`float, double, long double`	Десятичная дробь
Целочисленный тип данных	`short, int, long, long long`	Целое число

Виды инициализации

Копирующая инициализация:

```
int value = 5;
```

Прямая инициализация:

```
int value(5);
```

Универсальная инициализация:

```
int value{5};
```

Размер основных типов данных в C++

Категория	Тип	Минимальный размер
Логический тип данных	`bool`	1 байт
Символьный тип данных	`char`	1 байт
Тип данных с плавающей запятой	`float`	4 байта
	`double`, `long double`	8 байт
Целочисленный тип данных	`short, int`	2 байта
	`long`	4 байта
	`long long`	8 байт

Размер основных типов данных в C++

```
1 == sizeof(char) <= sizeof(short)
  <= sizeof(int) <= sizeof(long)
  <= sizeof(long long)


sizeof(float) <= sizeof(double)
  <= sizeof(long double)
```

Целочисленные типы данных

Тип	Размер в байтах	Формат	Промежуток значений
`char`	1	знаковый	от **-127** до **127**
		беззнаковый	от **0** до **255**
`short, int`	2	знаковый	от **-32 767** до **32 767**
		беззнаковый	от **0** до **65 535**
`long`	4	знаковый	**$\pm 2{,}14 \cdot 10^9$**
		беззнаковый	от **0** до **$4{,}29 \cdot 10^9$**
`long long`	8	знаковый	**$\pm 9{,}22 \cdot 10^{18}$**
		беззнаковый	от **0** до **$1{,}84 \cdot 10^{19}$**

Целочисленные типы фиксированного размера

```
int8_t      int_fast8_t     int_least8_t
int16_t     int_fast16_t    int_least16_t
int32_t     int_fast32_t    int_least32_t
int64_t     int_fast64_t    int_least64_t

uint8_t     uint_fast8_t    uint_least8_t
uint16_t    uint_fast16_t   uint_least16_t
uint32_t    uint_fast32_t   uint_least32_t
uint64_t    uint_fast64_t   uint_least64_t
```

Рекомендации

Если размер неважен — **`int`**.

Если важен размер и необходима максимальная производительность — **`int_fast#_t`**.

Если важен размер и экономия памяти важнее производительности — **`int_least#_t`**.

Типы с плавающей точкой

Тип	Размер в байтах	Промежуток значений	Точность
`float`	4	от $\pm 1{,}18 \cdot 10^{-38}$ до $\pm 3{,}4 \cdot 10^{38}$	~ 7
`double`, `long double`	8	от $\pm 2{,}23 \cdot 10^{-308}$ до $\pm 1{,}80 \cdot 10^{308}$	~ 15

Логический тип данных

```
bool isEqual(int x, int y) {
  return (x == y);
}

...

if (5) {
  cout << "hi";
}
else {
  cout << "bye";
}
```

Символьный тип данных

Название	Символ
Горизонтальная табуляция	`\t`
Двойная кавычка	`\"`
Обратная косая черта	`\\`

Литералы

```
auto unsignedIntVal = 32U;
auto longVal = 32L;
auto unsignedLongVal = 32UL;
auto signedLongLongIntVal = 32LL;
auto unsignedLongLongIntVal = 32ULL;

...

auto floatVal = 5.0f;
auto doubleVal = 5.0L;

...

auto oct = 0377;       // 255
auto hex = 0x3FFF;     // 16 383
auto bin = 0b101010;   // 42
```

Приведение типов

`const_cast<тип>(выражение)` — удаление спецификаторов **`const`** и **`volatile`**.

```
int i = 5;
const int *pi = &i;
int *j  = const_cast<int*>(pi);
```

Приведение типов

`static_cast<тип>(выражение)` — преобразование между встроенными (целыми, вещественными, перечисляемыми) типами, созданными типами, указателями (один из которых должен быть на **`void`**) и объектами классов, в случае когда один класс является наследником другого.

```
double doubleVal = 100;
int intVal = static_cast<int>(doubleVal);
```

Приведение типов

`dynamic_cast<тип*>(выражение)` — преобразование по иерархии наследования.

`reinterpret_cast<тип>(выражение)` — преобразование указателя в указатель любого другого типа, целочисленного типа в любой тип указателя и наоборот.

Выражение, операторы, инструкции

Основные термины

Выражение состоит из одного или большего числа операндов и символов операций. Выражение, заканчивающееся точкой с запятой, является оператором (инструкцией).

Оператор – это символ, который указывает компилятору на выполнение конкретных математических действий или логических операций.

```
first_name + " " + last_name
```

Приоритет и ассоциативность

Приоритет определяет, в каком порядке будут выполняться операторы умножения, разности и др.

Ассоциативность определяет в каком порядке будут выполняться операторы с одинаковым приоритетом (слева направо или справа налево). Унарные операторы и операторы присваивания – правоассоциативны, остальные левоассоциативны.

Приоритет и ассоциативность конкретных операторов рассматривается самостоятельнно

Упрощённая запись

Существует ряд операторов, упрощающих запись стандартных выражений .

Например:

Запись a=a+b можно заменить на a+=b.

Запись a=a+1 можно заменить на a++.

Операторы отношения

По умолчанию операторы отношения и логические операторы имеют тип bool, т.е. при выполнении операций отношения и логических операций получаются значения true и false.

Пример некоторых операторов проверки:

a<b, a==b, a!=b, a>=b и др.

Пример логического оператора:

(x>0 && x<10).

Арифметические преобразования в выражениях

Если в выражении смешаны различные типы литералов и переменных, компилятор преобразует их к одному типу.

Во-первых все char и short int-значения автоматически преобразуются к типу int.

Во-вторых, все операнды преобразуются к типу самого большого операнда.

После преобразования оба операнда будут иметь один и тот же тип и таким же будет тип результата.

Константы

В C++ введена концепция определенных пользователем констант для указания, что значение нельзя изменять непосредственно.

Например:

```
const double pi = 3.1415926;
```

Для того, чтобы избежать в коде «магических чисел» следует систематически пользоваться константами

Объявления и определения

Определение и объявление:

```
char ch;
```

Определение,объявление,инициализация

```
int count = 1;
```

Объявление без определения

```
void foo();
```

В программе на C++ для каждого имени должно быть ровно одно определение, а объявлений может быть несколько. Все объявления некой сущности должны согласовываться по типу этой сущности.

Область видимости. Время жизни

Для имени, объявленного в теле функции (локального), область видимости начинается с места объявления имени и заканчивается в конце блока, в котором это имя объявлено.

```
void foo() {
  int z;
  …
}
```

Область видимости. Время жизни

Имя называется *глобальным*, если оно объявлено вне функции, класса (структуры) или пространства имен.

Область видимости глобальных имен простирается от места их объявления до конца файла, содержащего объявление.

Объявление имени в блоке может скрыть это объявление в охватывающем блоке или глобальное имя. После выхода из блока имя восстанавливает свой прежний смысл.

Область видимости. Время жизни

```
int v; // глобальная переменная
void foo() {
  int v; // внешняя локальная переменная
    if (…) {
       // внутренняя локальная переменная
      int v;
       …
    }
//здесь видна внешняя локальная переменная
  …
}
```

Инструкции

Оператор if

Полная форма записи оператора if:

```
if (выражение)
   on1
else
   on2
```

Сокращённая форма оператора if:

```
if (выражение)
   on1
```

Оператор ветвления switch

Формат оператора:

```
switch (выражение) {

 case константа: операторы
 case константа: операторы
 case константа: операторы
 …
 default: операторы
}
```

Оператор ветвления switch

```
int a=1;
switch(a)
{
    case 1:
        a++;
       break;
    case 2:
        a++;
    case 3:
        a++;
}
cout << “a=“ << a;
```

Оператор ветвления switch

```
enum RainbowColor {
    RC_RED, RC_ORANGE, RC_YELLOW,
    RC_GREEN, RC_BLUE, RC_INDIGO
};
RainbowColor chosen_color = RC_RED;
switch (chosen_color){
    case RC_RED:    /* red */
    case RC_ORANGE: /* orange */
    case RC_YELLOW: /* yellow */
    case RC_GREEN:  /* green */
    case RC_BLUE:   /* blue */
    case RC_INDIGO: /* indigo */
    default:        /* все остальное */
}
```

Оператор цикла while

Формат оператора:

```
while(выражение)
  оператор
```

Оператор цикла do-while:

```
do
  оператор
while(выражение);
```

Оператор цикла for

Формат оператора:

```
for(выражение1;выражение2;выражение3)
  оператор
```

Выражение1 описывает инициализацию цикла.

Выражение2 – проверка условия завершения цикла. Если оно **истинно**, то выполняется оператор тела цикла for. Затем выполняется *выражение3*. После этого управление передаётся снова на проверку *выражения2* и так продолжается до тех пор, пока оно не станет **ложным.**

Выражение3 вычисляется после каждой итерации.

Оператор цикла for

```
int i = 0;

for (i = 0; i < 10; ++i){ }

std::cout << i;
```

Оператор цикла for

```
for(for-range-declaration : expression)
   statement

int x[] = { 1,2,3,4,5,6,7,8,9,10 };

for( int y : x ) {
    cout << y << " ";
}
```

Оператор завершения break

Прекращает выполнение ближайшего вложенного внешнего оператора switch, while, do-while или for. Управление передаётся оператору, следующему за прерываемым. Типичные случаи использования этого оператора – закончить выполнение цикла при присваивании некоторой переменной определённого значения или завершения выполнения последовательности операторов в операторе switch.

Оператор завершения break

```
int main()
{
  for (int i = 1; i < 10; i++)
  {
    cout << i << endl;
    if (i == 4)
      break;
  }
}
```

Оператор продолжения continue

Передаёт управление в начало ближайшего вложенного внешнего оператора цикла while, do-while или for, вызывая тем самым начало новой итерации.

Оператор продолжения continue

```
int main()
{
  int i = 0;
  do
  {
    i++;
    cout << "before the continue‟ << endl;
    continue;
    cout << "should never print";
  }
  while (i < 3);
}
```

Оператор возврата return

Прекращает выполнение текущей функции и возвращает управление вызвавшей программе с передачей значения выражения в качестве возвращаемого значения функции.