Программирование на языке C++

Шаблов Анатолий
anatoliishablov@gmail.com

ИТМО, весенний семестр 2025

Базовые элементы программы

Составляющие функции

```
[[nodiscard]] constexpr int sum(int lhs, int rhs) noexcept {
    return lhs + rhs;
}
```

Составляющие функции

```
[[nodiscard]] constexpr int sum(int lhs, int rhs) noexcept {
    return lhs + rhs;
}
```

Составляющие функции

```
[[nodiscard]] constexpr int sum(int lhs, int rhs) noexcept {
    return lhs + rhs;
}
```

Составляющие функции

```
[[nodiscard]] constexpr int sum(int lhs, int rhs) noexcept {
    return lhs + rhs;
}
```

Составляющие функции

```
[[nodiscard]] constexpr int sum(int lhs, int rhs) noexcept {
    return lhs + rhs;
}
```

Составляющие функции

```
[[nodiscard]] constexpr int sum(int lhs, int rhs) noexcept {
    return lhs + rhs;
}
```

Составляющие функции

```
[[nodiscard]] constexpr int sum(int lhs, int rhs) noexcept {
    return lhs + rhs;
}
```

Вызов функции

```
void f() {}

void g(int, char, double = 0.1) {}

int h(int a = -1) {
  g(a, 'x');  // third parameter is 0.1
  return a + 2;
}
int main() {
  f();
  g(h(), 'a', 0.5);
}
```

Вызов функции

```
void f() {}

void g(int, char, double = 0.1) {}

int h(int a = -1) {
  g(a, 'x');  // third parameter is 0.1
  return a + 2;
}
int main() {
  f();
  g(h(), 'a', 0.5);
}
```

Вызов функции

```
void f() {}

void g(int, char, double = 0.1) {}

int h(int a = -1) {
  g(a, 'x');  // third parameter is 0.1
  return a + 2;
}
int main() {
  f();
  g(h(), 'a', 0.5);
}
```

Вызов функции

```
void f() {}

void g(int, char, double = 0.1) {}

int h(int a = -1) {
  g(a, 'x');  // third parameter is 0.1
  return a + 2;
}
int main() {
  f();
  g(h(), 'a', 0.5);
}
```

Области видимости

```
void foo() {
  int a{5};
  {
    int b = a;
    { int a, b = 101; }
  }
  {
    long a, b = 1L, c = -5;
    char d = 'X';
  }
}
```

Области видимости

```
void foo() {
  int a{5};
  {
    int b = a;
    { int a, b = 101; }
  }
  {
    long a, b = 1L, c = -5;
    char d = 'X';
  }
}
```

Области видимости

```
void foo() {
  int a{5};
  {
    int b = a;
    { int a, b = 101; }
  }
  {
    long a, b = 1L, c = -5;
    char d = 'X';
  }
}
```

Области видимости

```
void foo() {
  int a{5};
  {
    int b = a;
    { int a, b = 101; }
  }
  {
    long a, b = 1L, c = -5;
    char d = 'X';
  }
}
```

Области видимости

```
void foo() {
  int a{5};
  {
    int b = a;
    { int a, b = 101; }
  }
  {
    long a, b = 1L, c = -5;
    char d = 'X';
  }
}
```

Области видимости

```
void foo() {
  int a{5};
  {
    int b = a;
    { int a, b = 101; }
  }
  {
    long a, b = 1L, c = -5;
    char d = 'X';
  }
}
```

Области видимости

```
void foo() {
  int a{5};
  {
    int b = a;
    { int a, b = 101; }
  }
  {
    long a, b = 1L, c = -5;
    char d = 'X';
  }
}
```

Области видимости

```
void foo() {
  int a{5};
  {
    int b = a;
    { int a, b = 101; }
  }
  {
    long a, b = 1L, c = -5;
    char d = 'X';
  }
}
```

Области видимости

```
int a = 11;

void foo() {
  a++;
  {
    int a = a;
    a *= 3;
  }
}
```

```
int b;

int main() {
  foo();
  return a + b;
}
```

Зависимости внутри одного объявления

```
int main() { int a = 11, b = a + 2; }
```

Переменные, значения, объекты

Время жизни объектов

```
int a = 11;

int main() {
  int b;
  { int c = 11; }
  return b;
}
```

Типы размещений

- Автоматический
- Статический
- Thread-local
- Динамический

Идентификаторы

- [A-Za-z_][A-Za-z0-9_]*
- Совпадающие с ключевыми словами – зарезервированы
- Содержащие __ – зарезервированы
- Начинающиеся с _[A-Z] – зарезервированы
- Начинающиеся с _ – зарезервированы в глобальном пространстве имён

Составляющие текста программы

- Идентификаторы
- Числовые литералы
- Символьные и строковые литералы
- Операторы и прочие символы пунктуации

Составляющие текста программы

Имя - идентифицирующее выражение, связанное с некой программной сущностью через определение.

```
int a = 1;  // declaration

int f() {
  return a;  // usage
}
```

Использование → поиск имён → сущность

Литералы

- Булевские true, false
- Целочисленные
- Дробные
- Символьные ’a’
- Строковые "Hello\n"
- nullptr

Выражения и операторы

```
int main(int argc, char** argv) {
  argc++ + ++argc;  // UB
  argc = ++argc * 3;
  return (1 + 2 * 3) * *(argv[argc - 1]);
}
```

Выражения и операторы

```
int main(int argc, char** argv) {
  argc++ + ++argc;  // UB
  argc = ++argc * 3;
  return (1 + 2 * 3) * *(argv[argc - 1]);
}
```

Выражения и операторы

```
int main(int argc, char** argv) {
  argc++ + ++argc;  // UB
  argc = ++argc * 3;
  return (1 + 2 * 3) * *(argv[argc - 1]);
}
```

Выражения и операторы

```
int main(int argc, char** argv) {
  argc++ + ++argc;  // UB
  argc = ++argc * 3;
  return (1 + 2 * 3) * *(argv[argc - 1]);
}
```

Список операторов и их свойства

https://en.cppreference.com/w/cpp/language/operator_precedence

Результат и побочные эффекты

```
int f(int a, int b) { return a + b; }

int main() {
  int a = 0, b = -13;
  int c = a++ + ++b;
  b *= 2;
  return f(a, b);
}
```

Невычисляемый контекст

```
double f() { return 0.5; }

int main() {
  decltype(f()) a = f();
  auto b = f();
  return sizeof(b);
}
```

Полные выражения

```
int main(int argc, char** argv) {
  int a = argc + argc / 2, b = a;
  decltype(a + 2) c = 3;
  c += a / b;
  return a + b;
}
```

Константные выражения

```
int main() {
  const std::size_t len = 10;
  // constexpr std::size_t len = 10;
  std::array<int, len> xxs;
}
```

Временные объекты

```
int& f(int& a) {
  return a;
}

int main(int argc, char** argv) {
  return 11 + f(argc * 2);
}
```

Порядок исполнения

```
int main(int argc, char** argv) {
  int a, b = argc * 3;
  a = argc++ + b;
  f(a, b);
  f(a++, a);
  bool x = a > 5 || b < 3;
  a = a++ + 2, b = a;
}
```

Контекст игнорирования результата

```
int main() {
  int a = 1, b = 2;
  a + b;
  return a, b;
}
```

Инструкции

```
int main(int argc, char** argv) {
  int a = argc + 2;
  a *= 3;
  if (a < 3) {
    return 0;
  }
  if (a > 5) {
    a += 4;
  } else
    a -= 3;
  for (int i = 1; i < argc; ++i) {
    a += argv[i][0];
    continue;
    a -= 1;
  }
}
```

for

```
int main(int argc, char** argv) {
  for (int i = 0, j = 1; i + j < argc; ++i, ++j) {
    i += 2;
    j -= 2;
  }
}
```

for

```
int main(int argc, char** argv) {
  for (int i = 0, j = 1; i + j < argc; ++i, ++j) {
    i += 2;
    j -= 2;
  }
}
```

for

```
int main(int argc, char** argv) {
  for (int i = 0, j = 1; i + j < argc; ++i, ++j) {
    i += 2;
    j -= 2;
  }
}
```

Минимальный for

```
int main() {
  for (;;)
    ;
}
```

if

```
int main(int argc, char** argv) {
  if (argc > 3) {
  }
  if (int i, j, k; argc < 3) {
    i = 1;
    j = 2;
    k = 3;
  } else {
    k = 10;
  }
  return i + j;  // error
}
```

switch

```
int main(int argc, char** argv) {
  int a = 0, b = 1;
  switch (argc) {
    case 1:
      a += 2;
      b -= 3;
    case 2:
      a *= b;
      break;
    case 3:
      return b;
    default:
      a += b * 3;
  }
}
```

Объявление и тело switch

```
int main(int argc, char** argv) {
  switch (argc) {
    case 1:
      int a = 1;
      break;
    default:  // error!
      return argc;
  }
}
```

Объявление и тело switch: правильно

```
int main(int argc, char** argv) {
  switch (argc) {
    case 1: {
      int a = 1;
      break;
    }
    default:
      return argc;
  }
}
```

Как можно использовать switch

```
void g(const std::size_t count, char* to, const char* from) {
  std::size_t n = (count + 7) / 8;
  switch (count % 8) {
    case 0: do { *to = *from++; [[fallthrough]];
        case 7:  *to = *from++; [[fallthrough]];
        case 6:  *to = *from++; [[fallthrough]];
        case 5:  *to = *from++; [[fallthrough]];
        case 4:  *to = *from++; [[fallthrough]];
        case 3:  *to = *from++; [[fallthrough]];
        case 2:  *to = *from++; [[fallthrough]];
        case 1:  *to = *from++;
      } while (--n);
  }
}
```

Базовые типы

Базовые типы

- `void`
- `std::nullptr_t`
- Арифметические
  - Дробные
  - Целые
    - Логический `bool`
    - Символьные `char`…
    - Знаковые целые `int`…
    - Беззнаковые целые `unsigned`...

Требования к фундаментальным типам

https://en.cppreference.com/w/cpp/language/types

https://en.cppreference.com/w/cpp/types/climits#Limits_of_floating_point_types

Приведение базовых типов

Числовые расширения

- signed char →int
- unsigned char →int или unsigned int
- short →int
- unsigned short →int или unsigned int
- char – либо как signed char, либо как unsigned char
- float →double

https://en.cppreference.com/w/cpp/language/implicit_conversion

Числовые преобразования

```
int main() {
  int a = true;           // 1
  double b = true;        // 1.0
  float c = b;            // 1.0
  unsigned char d = c;    // 1
  unsigned int e = -1;    // 0xFFFFFFFF
  int f = 1.33;           // 1
}
```

https://en.cppreference.com/w/cpp/language/implicit_conversion

Стандартные преобразования

```
int main() {
  int a = 1;
  unsigned b = 2;
  long long c = -3;
  float d = 0.5;
  double e = -1.33;

  auto x = a + b;  // unsigned
  auto y = b + c;  // long long
  auto z = d + e;  // double
  auto u = a + e;  // double;
}
```

https://en.cppreference.com/w/cpp/language/implicit_conversion

Явные преобразования

```
int main() {
  long long x = -1;
  auto y = static_cast<unsigned>(x) + 2;  // unsigned
}
```

C-style cast

- `(T)x`
- `T(x)`

- `const_cast`
- `static_cast`
- `static_cast + const_cast`
- `reinterpret_cast`
- `reinterpret_cast + const_cast`