Программирование на языке C++

Вводный курс

Александр Морозов
gelu.speculum@gmail.com

ИТМО, весенний семестр 2021

Содержание

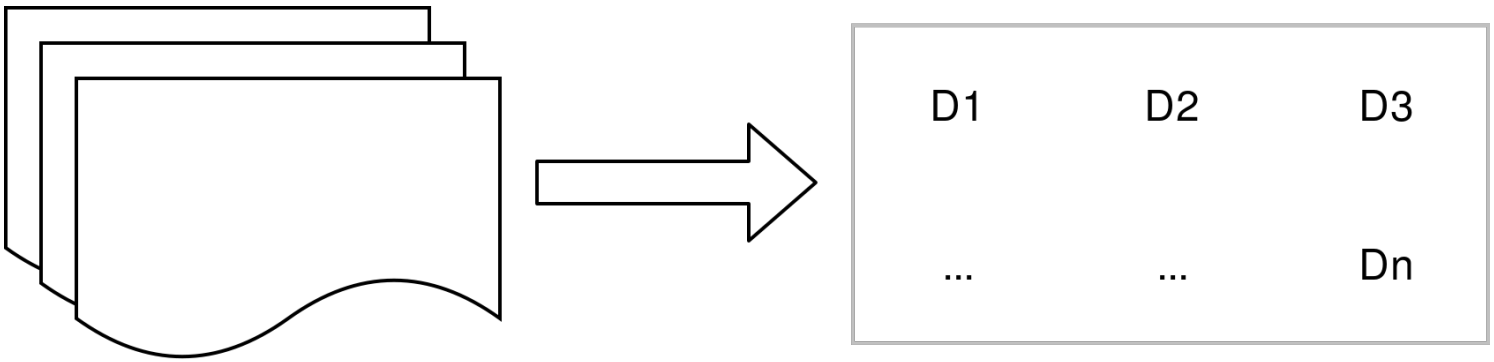
Текст программы и этапы его обработки

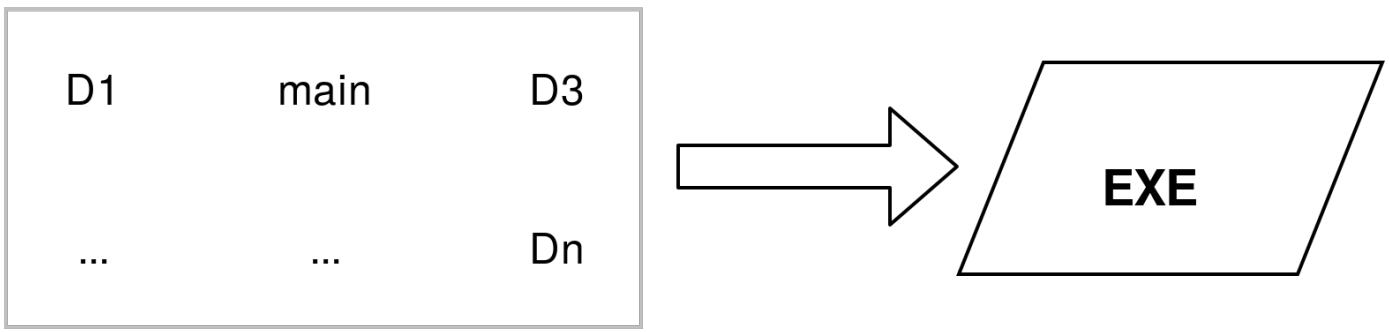
Функция main

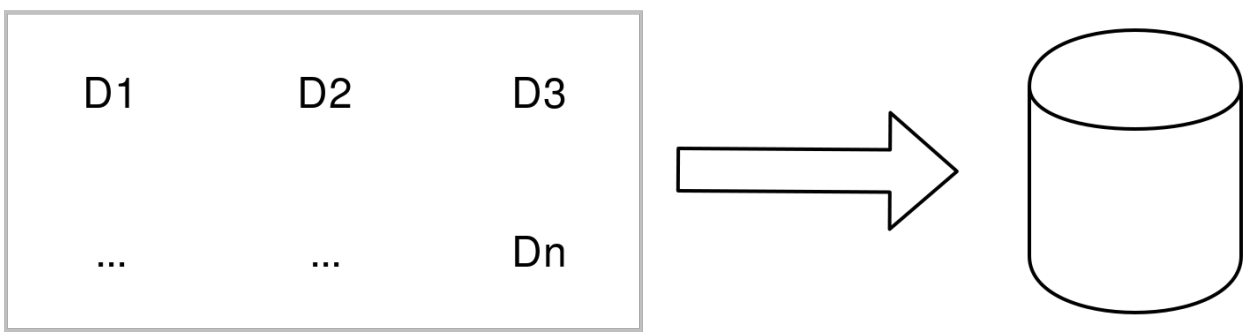
Базовые элементы программы

Базовые типы

Текст программы







Комментарии

```
1   /* Comment */
2
3   /*
4    * Multi-line comment
5    */
6
7   // Single-line comment
8
9   //
10  //
11  // Fancy comment formatting
12  //
```

Основные этапы трансляции

1. Склейка строк через \
2. Предварительная токенизация (комментарии, пробелы, базовые токены)
3. Препроцессор ($\forall I \rightleftharpoons 1 \ldots 3$)
4. Склейка смежных строковых литералов
5. Компиляция
6. Компоновка

Базовые токены

- идентификаторы
- числовые токены
- символьные, строковые литералы и аргументы `#include`
- операторы и символы пунктуации
- иное

```
1    a+++++b // a ++ ++ + b
2    a++ + ++b // a ++ + ++ b
3    1E+12 // OK
4    0x1E+12 // bad
5    0x1E +12 // OK
```

Содержание

Текст программы и этапы его обработки

Функция main

Базовые элементы программы

Базовые типы

Функция main

- нельзя явно вызвать
- нельзя взять адрес
- не может быть автоматически сгенерирована
- не может быть перегружена
- не может иметь спецификаторов `inline`, `static`, `constexpr`

```
1  int main() { /* body */ }
2  int main(int argc, char * argv[]) { /* body */ }
```

- вызывается после инициализации глобальных переменных
- выход из `main` $\equiv$ завершение программы
- `return 0;` автоматически генерируется в конце тела

Содержание

Текст программы и этапы его обработки

Функция main

Базовые элементы программы

Базовые типы

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

Неквалифицированные идентифицирующие выражения

- корректно определённые идентификаторы
- имя перегруженного оператора `operator ==`
- имя оператора пользовательского преобразования типа `operator bool`
- имя оператора пользовательского литерала `operator ""_km`
- имя шаблона со списком аргументов `T<a, b, c>`
- символ `~` с последующим именем класса `~Foo`
- `~decltype(x)`

Квалифицированные идентифицирующие выражения

```
1   int a = 13;
2
3   struct C
4   {
5     int get() const
6     { return a; }
7
8     int get2() const
9     { return ::a; }
10
11    int get3() const
12    { return ::C::a; }
13
14    static int a = 111;
15  };
16
17  int b = C::a;
```

Имена

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

```
1       int a = 1; // declaration
2
3       int f()
4       {
5           return a; // usage
6       }
```

Использование $\rightarrow$поиск имён $\rightarrow$сущность

Литералы

- булевские `true`, `false`
- целочисленные
- дробные
- символьные `'a'`
- строковые `"Hello\n"`
- `nullptr`

Целочисленные литералы

```
1       -1 // -(1)
2
3       31 // decimal
4       037 // octal
5       0x1F // hexadecimal
6       0X1f // hexadecimal
7       0b11111 // binary
```

Тип десятичного литерала – `int`, `long`, `long long`.

Типы других баз могут быть беззнаковые.

- суффикс `u`, `U` $\Longrightarrow$ `unsigned`
- суффикс `l`, `L` $\Longrightarrow$ `long`
- суффикс `ll`, `LL` $\Longrightarrow$ `long long`

Дробные литералы

```
1       1e-5 // 10^-5
2       1. // 1.0
3       1.23 // 1.23
4       0x1.2p-1 // 0.5625
```

Тип дробного литерала – `double`, либо задаётся суффиксом:

- `f F` $\Longrightarrow$ `float`
- `l L` $\Longrightarrow$ `long double`

Операторы

Оператор – элемент языка, задающий некоторое вычисление над операндами. Может иметь побочные эффекты и результат.

```
a + b
```

Свойства:

- арность
- приоритет
- ассоциативность

```
1    a = b = c // a = (b = c)
2    a , b, c // (a , b) , c
```

Список операторов и их свойства

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

Выражения

Операторы + операнды $\Longrightarrow$ выражения

Первичные выражения

- литералы
- идентификаторы
- (выражение)
- лямбда-выражения
- выражения свёртки

Операнд – первичное выражение или составное выражение.

Полное выражение

- операнд в невычисляемом контексте
- константное выражение
- выражение инициализации [1]
- вызов деструктора, сгенерированный в конце времени жизни не-временного объекта
- выражение, не являющееся частью другого выражения

```
1       size_t n = sizeof(x) * 2 + y;
2
3       int a = 1, b = a;
4
5       {
6           S s;
7       }
```

[1] https://stackoverflow.com/questions/24224115/interdependent-initialization-with-commas

Порядок исполнения

Порядок исполнения выражения – порядок вычисления подвыражений составного выражения $\Longrightarrow$ не определён.

$\succ$ **“упорядочено до”** – это несимметричное, транзитивное отношение между парой вычислений в рамках одного треда.

$A \succ B \Longrightarrow$ A полностью завершено до B

$\neg\, A \succ B \wedge \neg\, B \succ A \Longrightarrow$

- A и B разнесены во времени
- A и B пересекаются во времени

Некоторые правила упорядочивания

$F_i \succ F_{i+1}$

$a\ \mathbf{op}\ b \implies \mathbf{result}(a) \succ \mathbf{result}(a\mathbf{op}b) \wedge \mathbf{result}(b) \succ \mathbf{result}(a\mathbf{op}b)$

$\mathbf{fun}(args) \implies args \succ fun - eval$

Больше правил:
`https://en.cppreference.com/w/cpp/language/eval_order`

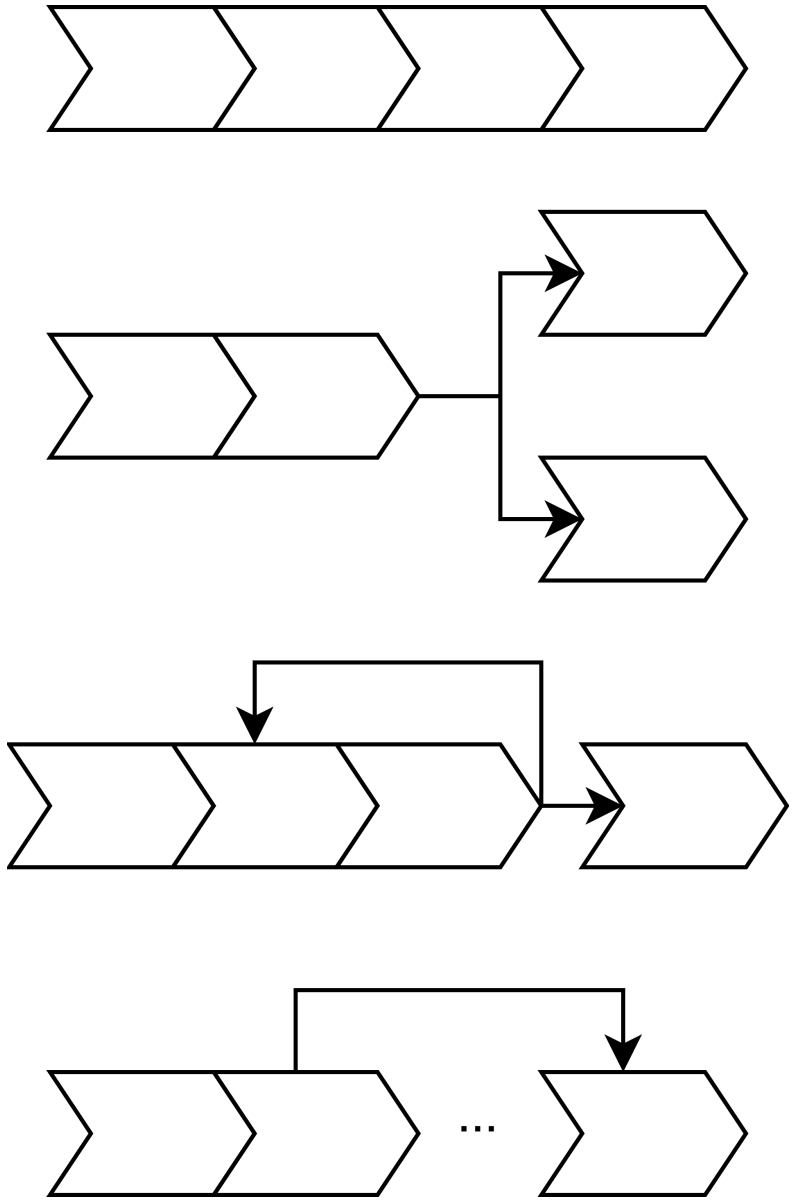
“Short-circuit evaluation” для *встроенных* операторов `&&` и `||`.

Инструкции

$$P = S_1 \to S_2 \to \ldots S_n$$

- выражений <*выражение*> ;
- блоки { <*инструкции*> }
- ветвлений
- циклов
- переходов
- объявлений
- `try` блоки

Инструкции, наглядно



Ветвления

```
1       if (err)
2           std::cout << err << std::endl;
3
4       if (x > y) {
5           a = x;
6       } else {
7           a = y;
8       }
9
10      switch (z) {
11          case 'a': return 101;
12          case 'c': return 33;
13          default: return -1;
14      }
```

Циклы

```
1       for (int i = -10; i < 10; ++i) {
2           a *= i * i;
3       }
4
5       for (const auto & id : x.get_ids()) {
6           std::cout << id << std::endl;
7       }
8
9       while (!q.empty()) {
10          send(q.front());
11          q.pop_front();
12      }
13
14      do {
15          std::cout << "Hello there" << std::endl;
16      } while (false);
```

Переходы

- `goto` *метка* ;
- `break`;
- `continue`;
- `return` *выражение* ;
- `return` *braced init list* ;

Переменные

<список спецификаторов> <список инициализации> ;

список спецификаторов ≡ [[*спецификатор*]...] *тип*

список инициализаци ≡ *id* [= *expr*] [, *id* ...]

```
1       int a;
2       char b = '\t';
3       const float x = 0.001, y = 1e+10;
```

Область видимости

```
1       // no 'x' visible yet
2
3       int x = 11;
4       int main()
5       {
6         {
7           int y = x + 2; // 13
8           int x = x; // uninitialized
9           x = y * 2; // 26
10          {
11            int z = x; // 26
12          }
13        }
14        int z = x; // 11
15        int u = y; // compilation error
16      }
```

Некоторые операторы

```
1       int x, y = 5; // not an assignment
2       x = 1 + 2;
3       y *= x / 2;
4       x = ++y;
5       x = y++;
6       x--;
7       y = -x;
8       x = 10 % 4;
9       x = 0b101 & 0xF;
10      x = 0xC | 0xA;
11      x ^= 011;
12      x = 0b101 << 4;
13      auto b = !x || y;
14      auto bb = !x && y;
15      auto bbb = x != y;
16      auto bbbb = x == y;
```

Содержание

Текст программы и этапы его обработки

Функция main

Базовые элементы программы

Базовые типы

Типы

Объекты, выражения, функции, ... : тип

- базовые (фундаментальные)
- сложные (составные)

```
1       using T = int;
2
3       static const int * x;
```

Базовые типы

- `void`
- `std::nullptr_t`
- арифметические
  - дробные
  - интегральные
    - логический `bool`
    - символьные `char`...
    - знаковые целые `int`...
    - беззнаковые целые `unsigned`...

Явные и неявные приведения типов

```
1       float a = 10; // int -> float
2       int i = -101;
3       unsigned long x = 1; // int -> unsigned long
4       auto y = x + i; // int -> unsigned long
5       int b = 1.5; // double -> int
6
7       int c{static_cast<int>(a)}; // explicit cast
```

Требования к целым числовым типам

Тип	Минимальное значение	Максимальное значение	ILP32	LP64	LLP64
char			8	8	8
signed char	-127	127	8	8	8
unsigned char	0	255	8	8	8
short short int	-32767	32767	16	16	16
unsigned short unsigned short int	0	65535	16	16	16
int	-32767	32767	32	32	32
unsigned unsigned int	0	65535	32	32	32
long long int	-2147483647	2147483647	32	64	32
unsigned long unsigned long int	0	4294967295	32	64	32
long long long long int	$-2^{63}-1$	$2^{63}-1$	64	64	64
unsigned long long unsigned long long int	0	$2^{64}-1$	64	64	64

Требования к дробным числовым типам

Тип	**Минимальное число точно представимых десятичных цифр**	**Максимально представимое число**
float	6	1E+37
double	10	1E+37
long double	10	1E+37