Программирование на языке C++

Вводный курс

Александр Морозов
gelu.speculum@gmail.com

ИТМО, весенний семестр 2021

Содержание

Числовые расширения

$$T1 \triangleright T2 \implies \textbf{value}(1) \equiv \textbf{value}(2)$$

- `signed char`, `short` →`int`
- `unsigned char`, `unsigned short` →`int` или `unsigned int`
- `char` →`int` или `unsigned int`
- `float` →`double`

Интегральные преобразования

$$\forall T1, T2 : T1 \in \mathbb{I}, T2 \in \mathbb{I}, T1 \mapsto T2$$

- $T2 \in \mathbb{U}, \textbf{sizeof}(T1) > \textbf{sizeof}(T2) \implies$ результат – младшие биты исходного представления
- $T2 \in \mathbb{U}, T1 \in \mathbb{S}, \textbf{sizeof}(T1) \leq \textbf{sizeof}(T2) \implies$ результат – знаковое дополнение
- $T2 \in \mathbb{U}, T1 \in \mathbb{U}, \textbf{sizeof}(T1) \leq \textbf{sizeof}(T2) \implies$ результат – дополнение нулями
- $T2 \in \mathbb{S}$ и исходное значение помещается в $T2 \implies$ значение не меняется
- $T2 \in \mathbb{S}$ и исходное значение **не** помещается в $T2 \implies$ implementation defined

Дробные преобразования

$$\forall T1, T2 : T1 \in \mathbb{F}, T2 \in \mathbb{F}, T1 \mapsto T2$$

- если исходное значение *точно* представимо в целевом типе, оно не меняется
- если исходное значение попадает между двумя корректными значениями целевого типа, то результат – одно из этих значений (implementation defined)
- иначе – **UB**

Преобразования между дробными и интегральными типами

- дробное $\mapsto$ целое путём отбрасывания дробной части, если целая часть помещается в целевой тип, иначе – **UB**
- целое $\mapsto$ дробное, возможно, с округлением (implementation defined); если исходное значение слишком велико – **UB**

Преобразования с `bool`

$$\forall T : T \in \mathbb{I} \vee \mathbb{F} \vee \mathbb{E} \vee \mathbb{P}, T \mapsto \texttt{bool}$$

$$\forall T : T \in \mathbb{I} \vee \mathbb{F}, \texttt{bool} \mapsto T$$

Преобразования к `bool`

- $value \neq 0 \implies$ `true`
- $value = 0 \implies$ `false`

Преобразования из `bool`

- `false` $\rightarrow 0$
- `true` $\rightarrow 1$

Стандартные арифметические преобразования

1. расширение операндов интегрального типа
2. приведение к общему типу
   - если один операнд `long double`, второй приводится к `long double`
   - если один операнд `double`, второй приводится к `double`
   - если один операнд `float`, второй приводится к `float`
   - если знаковость одинакова, то подтягивается ранг
   - если ранг беззнакового $\geq$ ранга знакового, то знаковый $\rightarrow$тип беззнакового
   - если тип знакового может представить все значения типа беззнакового, то беззнаковый $\rightarrow$тип знакового
   - иначе оба $\rightarrow$беззнаковый вариант типа знакового

Ранг: `bool` $<$ `signed char` $<$ `short` $<$ `int` $<$ `long` $<$ `long long`

Унарные плюс и минус

- `+x` – числовое расширение
- `-x`
  - числовое расширение
  - для беззнаковых целых – $-x = 2^n - x$
  - для остальных – инвертирование знака

Целочисленное переполнение

- беззнаковая целочисленная арифметика – по модулю 2^n
- знаковое переполнение – **UB**

Контекст преобразования к `bool`

- условие `if`, `while`, `for`
- операнды встроенных `!`, `&&`, `||`
- первый операнд `?:`
- предикат `static_assert`
- выражение в `noexcept`

Old-style cast

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

- `const_cast<T>(x)`
- `static_cast<T>(x)*`
- `static_cast*` + `const_cast`
- `reinterpret_cast<T>(x)`
- `reinterpret_cast` + `const_cast`

Содержание

Объекты

Характеризуется:

- размер (`sizeof`)
- выравнивание (`alignof`)
- тип размещения
- размещение
- время жизни
- тип
- значение (возможно, неопределенное)
- имя (не обязательно)

Не является объектом: значение, ссылка, функция, не статический член класса, `this`.

Представление объекта

- объектное представление $B_1 \dots B_n, n = \texttt{sizeof(T)}$
- представление значения $b_1 \dots b_m, \frac{m}{8} \leq n$

Подобъекты – члены классов, представления базовых классов в наследнике, элементы массивов.

Размер любого полного объекта ≥ 1.

$\forall a, b : L_1 \cap L_2 \implies \mathbf{addr}(a) \neq \mathbf{add}(b)$

Типы размещения

- автоматический
- статический
- тред-локальный
- динамический

Размещение и инициализация

```
1       int a = 9, aa;
2       thread_local double b = 0.5;
3       C c(1, 2);
4
5       void f()
6       {
7         static C cc;
8         thread_local double d = -1.5;
9         {
10          ...
11          int e = 101;
12          ...
13        }
14      }
```

Временные объекты

- автоматическое размещение в рамках исполнения выражения
- удаляются в конце исполнения полного выражения
- порядок удаления = обратный порядок создания
- время жизни может быть продлено ссылкой

```
1       x = process(std::string("a␣=␣b␣+␣c").substr(4, 5));
```

Содержание

Невычисляемый контекст

- `typeid`
- `sizeof`
- `noexcept`
- `decltype`

```
1       decltype(a + b) c = a + b;
```

Контекст игнорирования результата

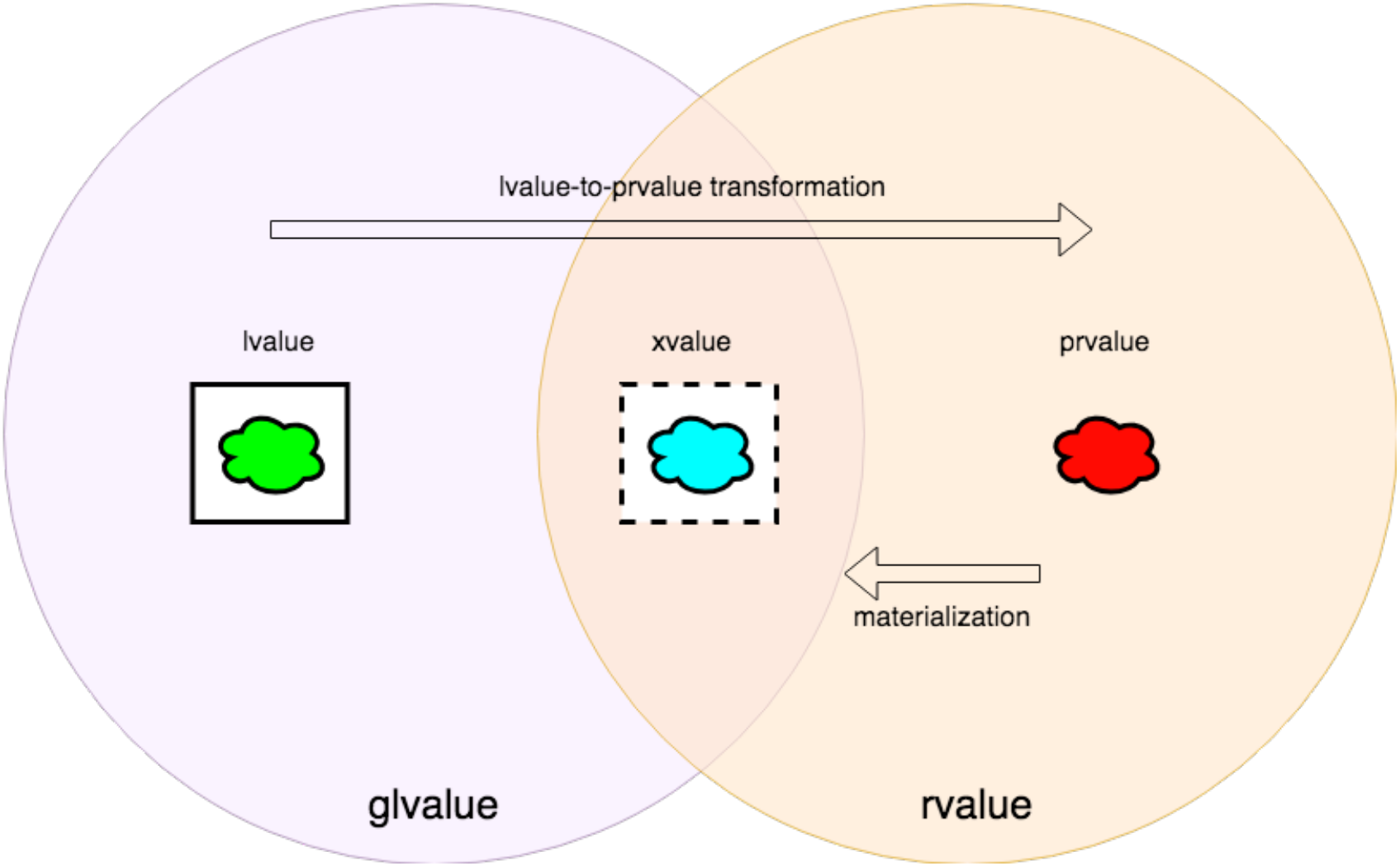
- инструкция выражения
- левый операнд ,
- приведение к типу `void`

```
1       a + b;
2       f();
3       x = f(), g();
4       auto y = f(), g(); // error
```

Константные выражения

```
1       char str[50];
2
3       const std::size_t size = 50;
4       std::array<int, size * 10> arr;
```

Категории значений



Пример материализации

```
1       const int a = 2 + 3;
2
3       const int & b = 2 + 3;
```

Содержание

Ветвления

`if` [`constexpr`] ([*иниц.*] *условие*) *инструкция*

`if` [`constexpr`] ([*иниц.*] *условие*) *инструкция* `else` *инструкция*

- *иниц.*
  - инструкция выражения
  - инструкция объявления
- *условие*
  - выражение, приводимое к `bool`
  - объявление единственной переменной

Примеры ветвлений

```
1       int a = f();
2       if (a) return x;
3
4       if (int a = f(); a)
5         return x;
6
7       int a;
8       if (a = f(); a) {
9         return x;
10      }
11
12      if (int a = f()) {
13        return x;
14      }
15
16      if (T t; !t.empty()) return y;
17
18      if (auto a = g(); std::size_t b = a.size()) {
19        std::cout << a << ":" << b << std::endl;
20      }
```

Примеры constexpr ветвлений

```
1   auto f()
2   {
3     if constexpr (true) {
4       return "Hi!";
5     } else {
6       return 101;
7     }
8   }
9
10  template <class T>
11  std::string f(const T & t)
12  {
13    if constexpr (std::is_same_v<T, std::string>) {
14      return t.substr(3, 4);
15    } else {
16      return std::to_string(t);
17    }
18  }
```

switch

`switch` ([*иниц.*] *условие*) *инструкция*

- *иниц.*
  - инструкция выражения
  - инструкция объявления
- *условие*
  - выражение, неявно приводимое к интегральному типу или `enum`
  - объявление единственной переменной

В инструкции разрешены метки `case` и `default`, а `break` имеет специальное значение.

Примеры `switch`

```
1       int a = f();
2       switch (a) {
3         case 11:
4         case 3+5:
5           ++x;
6           y = a;
7           break;
8       }
9
10      switch (int a = f()) {
11        case 11: ++a; [[fallthrough]];
12        case 8: { ++x; y = a; } break;
13      }
14
15      switch(int a = f(); a) {
16        case 8: ++a;
17        case 11: ++x; y = a; break;
18      }
19
20      switch (std::string s = get_line(); s.size());
```

Объявления и тело `switch`

```
1       switch (x) {
2         case 31: {
3           int a = x * 20;
4           std::cout << a;
5           break;
6         }
7         case 99: {
8           const auto l = get_line;
9           std::cout << l;
10        }
11        break;
12      }
13
14      switch (x) {
15        case 31:
16          int a = x * 20;
17          std::cout << a;
18          break;
19        default: // error
20          std::cout << x;
21      }
```

Ещё более странные вещи с `switch`

```
1   void g(const std::size_t count, char * to, const char * from)
2   {
3     std::size_t n = (count + 7) / 8;
4     switch (count % 8) {
5     case 0: do { *to = *from++; [[fallthrough]];
6     case 7: *to = *from++; [[fallthrough]];
7     case 6: *to = *from++; [[fallthrough]];
8     case 5: *to = *from++; [[fallthrough]];
9     case 4: *to = *from++; [[fallthrough]];
10    case 3: *to = *from++; [[fallthrough]];
11    case 2: *to = *from++; [[fallthrough]];
12    case 1: *to = *from++;
13            } while (--n);
14    }
15  }
```

https://en.wikipedia.org/wiki/Duff's_device

Цикл `while`

`while` (*условие*) *инструкция*

- *условие*
  - выражение, приводимое к `bool`
  - объявление единственной переменной

Примеры while

```
1       while (!q.empty()) {
2         std::cout << q.front() << std::endl;
3         q.pop_front();
4       }
5
6       while (int a = f()) {
7         std::cout << a << std::endl;
8       }
9
10      while (x)
11        int y = 111;
12      std::cout << y << std::endl; // error
```

Цикл `do while`

`do` *инструкция* `while` (*условие*);

```
1       do x++; while (x);
2
3       do {
4         int a = f();
5         x = a % 2;
6       } while (x > 5);
```

Цикл `for`

`for` (*иниц.* [*условие*] ; *итер.*) *инструкция*

- *иниц.*
  - инструкция выражения
  - инструкция объявления
- *условие*
  - выражение, приводимое к `bool`
  - объявление единственной переменной
- *итер.*
  - выражение

Примеры с `for`

```
1       for (;;) {
2         std::cout << "Forever␣failure" << std::endl;
3       }
4
5       for (; !q.empty(); ) {
6         std::cout << q.front() << std::endl;
7         q.pop_front();
8       }
9
10      for (int i = 0; i < f(); ++i) {
11        if (i % 2) {
12          std::cout << "Odd" << std::endl;
13          continue;
14        }
15        std::cout << "Even" << std::endl;
16      }
17
18      for (int i = 0, end = f(); i < end; ++i) {
19        // ...
20      }
```

Содержание

Функции

- имя
- тип возвращаемого значения
- список аргументов
- тип `ret(params)`
- тело

- объявление – задание типа и связывание с именем
- определение – задание тела и связывание с именем

Форма объявления функции

T имя (*параметры*);

`auto` *имя* (*параметры*) -> *T* ;

```
1       int a = 5, f();
```

Объявление аргументов

параметры : $p_1, \ldots, p_n$

- `type name`
- `type name = init`
- `type`
- `type = init`

Variadic functions

```
1    int printf(const char * fmt, ...);
2    int print(...);
```

Примеры объявлений функций

```
1       // declare the same function of type void(int, char *)
2       void bar(const int x, char[]);
3       void bar(const volatile int x, char[3]);
4       void bar(int x, char * p);
5
6       void g(int, int);
7       void g(int, int = 5);
8       void g(int = 1, int);
9       void g(int, int = 5); // error
10
11      int a = 1;
12      int f(int = a);
13      void b()
14      {
15        g(); // calls g(1, 5)
16        a = 2;
17        {
18          int a = 3;
19          f(); // calls f(2)
20        }
21      }
```

Определение функции

объявление тело

- блок
- try блок
- = `delete`
- = `default`

Инструкция `return`

- `return` *выражение* `;`
- `return { `*список инициализации*` };`