КОНСТРУИРОВАНИЕ ПРОГРАММ И ЯЗЫКИ ПРОГРАМИРОВАНИЯ

Лекция № 21.1 — Наборы бит (bitset)

Преподаватель: Поденок Леонид Петрович, 505а-5

+375 17 293 8039 (505а-5)
+375 17 320 7402 (ОИПИ НАНБ)
prep@lsi.bas-net.by

ftp://student:2ok*uK2@Rwox@lsi.bas-net.by

Кафедра ЭВМ, 2021

Оглавление

Битовый набор std::bitset

```
#include <bitset>
template <size_t N> class bitset;
```

Битовый набор хранит биты (элементы только с двумя возможными значениями — 0 или 1, истина или ложь, ...).

Класс имитирует массив элементов типа **bool**, но оптимизирован для распределения пространства памяти — обычно каждый элемент занимает только один бит, который в большинстве систем в восемь раз меньше, чем самый маленький элементарный тип **char**.

К каждой битовой позиции можно получить доступ индивидуально — например, для данного битового набора с именем **foo** выражение **foo[3]** обращается к своему четвертому биту, так же, как обычный массив обращается к своим элементам. Но поскольку в большинстве реализаций C++ элементарный тип не является единичным битом, доступ к отдельным элементам осуществляется как к специальному ссылочному типу (см. **bitset::reference**).

Битовые наборы могут быть созданными из и преобразованными как в целочисленные значения, так и в двоичные строки.

Их также можно напрямую вставлять и извлекать из потоков в двоичном формате.

Размер битового набора фиксируется во время компиляции, поскольку определяется его параметром шаблона N (количество бит). Это значение возвращается функцией **bitset::size()**.

Функции-члены

`bitset::bitset` — конструктор битового набора
`operator` — операторы битового набора

Доступ к битам

`operator[]` — доступ к биту
`count()` — подсчет установленных бит
`size()` — вернуть размер
`test()` — вернуть значение бита
`any()` — проверить, есть ли установленные биты
`none()` — проверить, что нет установленных бит
`all()` — проверить, все ли биты установлены

Операции с битами

`set()` — установить биты
`reset()` — сбросить биты
`flip()` — инвертировать биты

Операции с набором

`to_string()` — преобразовать в строку
`to_ulong()` — преобразовать в **`unsigned long int`**
`to_ullong()` — преобразовать в **`unsigned long long`**

bitset::bitset — контрукторы

(1) По умолчанию

```
bitset();
```

Объект инициализируется нулями

(2) Из целого

```
bitset(unsigned long long val);
```

Инициализирует объект значениями бит из **val**:

val — целочисленное значение, биты которого копируются в позиции набора битов.

- Если представление значения **val** больше, чем размер битового набора, во внимание принимаются только младшие значащие биты **val**.

- Если представление значения **val** меньше размера битового набора, оставшиеся битовые позиции инициализируются нулем.

(3) Из строки <basic_string>

```
template<class charT, class traits, class Alloc>
explicit bitset(const basic_string <charT, traits, Alloc>& str,
                typename basic_string <charT,traits, Alloc>::size_type pos = 0,
                typename basic_string <charT,traits, Alloc>::size_type n =
                        basic_string <charT,traits, Alloc>::npos);
```

Использует последовательность нулей и/или единиц в **str** для инициализации первых **n** битовых позиций построенного объекта битового набора.

str — строка **basic_string** или С-строка, содержимое которой используется для инициализации битового набора:

Конструктор анализирует строку, считывая не более **n** символов, начиная с **pos**, интерпретируя значения символов, указанных в качестве аргументов, как ноль и единица, соответственно, как нули и единицы.

Младший бит будет представлен последним прочитанным символом, а не первым.

Если эта последовательность короче, чем размер битового набора, оставшиеся битовые позиции инициализируются нулем.

pos — первый символ в **basic_string**, который нужно прочитать и интерпретировать.

Если **pos** больше, чем длина **str**, генерируется исключение **out_of_range**.

(3) Из С-строки

```
emplate <class charT>
explicit bitset(const charT* str,
                typename basic_string<charT>::size_type n =
                                                  basic_string<charT>::npos,
                charT zero = charT('0'),
                charT one = charT('1'));
```

Использует последовательность нулей и/или единиц в **str** для инициализации первых **n** битовых позиций построенного объекта битового набора.

Объекты битового набора имеют фиксированный размер, который определяется их аргументом шаблона класса, независимо от используемого конструктора — те битовые позиции, которые явно не установлены конструктором, инициализируются нулевым значением.

Пример создания

```
#include <iostream>       // std::cout
#include <string>         // std::string
#include <bitset>         // std::bitset

int main () {

  std::bitset<16> foo;
  std::bitset<16> bar (0xfa2);                    // 1111 1010 0010
  std::bitset<16> baz (std::string("0101111001"));
  std::bitset<16> boo (".@.@@.@@.@", 10, '.', '@');

  std::cout << "foo: " << foo << '\n';
  std::cout << "bar: " << bar << '\n';
  std::cout << "baz: " << baz << '\n';
  std::cout << "boo: " << boo << '\n';

  return 0;
}
```

Вывод

```
foo: 0000000000000000
bar: 0000111110100010
baz: 0000000101111001
boo: 0000000101101101
```

Операторы

Функции-члены

```
bitset& operator&= (const bitset& rhs) noexcept;    // AND
bitset& operator|= (const bitset& rhs) noexcept;    // OR
bitset& operator^= (const bitset& rhs) noexcept;    // XOR
bitset& operator<<= (size_t pos) noexcept;          // SHL
bitset& operator>>= (size_t pos) noexcept;          // SHR
bitset operator~() const noexcept;                  // NOT
bitset operator<<(size_t pos) const noexcept;       // SHL
bitset operator>>(size_t pos) const noexcept;       // SHR
bool operator== (const bitset& rhs) const noexcept; // Logical AND
bool operator!= (const bitset& rhs) const noexcept; // Logical NOT
```

Функции-не-члены

```
template<size_t N>
bitset<N> operator& (const bitset<N>& lhs, const bitset<N>& rhs) noexcept;
template<size_t N>
bitset<N> operator| (const bitset<N>& lhs, const bitset<N>& rhs) noexcept;
template<size_t N>
bitset<N> operator^ (const bitset<N>& lhs, const bitset<N>& rhs) noexcept;
```

lhs — левосторонний объект битового набора для функций, не являющихся членами/

rhs — правосторонний объект битового набора.

pos — количество позиций сдвига

Извлечение/вставка бит из потока/в поток

```
template<class charT, class traits, size_t N>
basic_istream<charT, traits>& operator>> (basic_istream<charT,
                                          traits>& is,
                                          bitset<N>& rhs);
template<class charT, class traits, size_t N>
basic_ostream<charT, traits>& operator<< (basic_ostream<charT,
                                          traits>& os,
                                          const bitset<N>& rhs);
```

is,os — объект **basic_istream** или **basic_ostream**, из которого соответственно извлекается или в который вставляется объект битового набора.

Формат, в котором битовые наборы вставляются/извлекаются, представляет собой последовательность расширенных символов «0» и «1».

Возвращаемое значение

Если ссылка, толевосторонний объект (*** this**, **is** или **os**).

Если не ссылка, результат операции — либо битовый объект, либо истина или ложь для операторов отношения.

Пример

```
#include <iostream>       // std::cout
#include <string>         // std::string
#include <bitset>         // std::bitset

int main () {

  std::bitset<4> foo (std::string("1001"));
  std::bitset<4> bar (std::string("0011"));

  std::cout << (foo ^= bar) << '\n';       // 1010 (XOR с присваиванием)
  std::cout << (foo &= bar) << '\n';       // 0010 (AND с присваиванием)
  std::cout << (foo |= bar) << '\n';       // 0011 (OR с присваиванием)

  std::cout << (foo <<= 2) << '\n';        // 1100 (SHL с присваиванием)
  std::cout << (foo >>= 1) << '\n';        // 0110 (SHR с присваиванием)

  std::cout << (~bar)     << '\n';         // 1100 (NOT)
  std::cout << (bar << 1) << '\n';         // 0110 (SHL)
  std::cout << (bar >> 1) << '\n';         // 0001 (SHR)

  std::cout << (foo == bar) << '\n';       // false (0110 == 0011) 0
  std::cout << (foo != bar) << '\n';       // true  (0110 != 0011) 1

  std::cout << (foo & bar) << '\n';        // 0010
  std::cout << (foo | bar) << '\n';        // 0111
  std::cout << (foo ^ bar) << '\n';        // 0101
}
```

operator[] — доступ к биту

```
#include <bitset>

     bool operator[] (size_t pos) const;
reference operator[] (size_t pos);
```

Функция возвращает значение или ссылку на бит в позиции **pos**.

С этим оператором проверка диапазона не выполняется — для доступа к значению с установленными границами битового набора следует использовать **bitset::test()**.

pos — позиция бита, к значению которого осуществляется доступ.

Позиции отсчитываются от самого правого бита, то есть, от позиции 0.

size_t — целочисленный тип без знака.

Возвращаемое значение

Бит в позиции **pos**.

Если объект битового набора квалифицируется как **const**, функция возвращает значение типа **bool**.

Если объект битового набора не квалифицируется как **const**, функция возвращает значение специального типа ссылки, которое имитирует значение типа **bool**, но с семантикой ссылки на отдельный бит в битовом наборе (**bitset::reference**).

bitset::reference — специальный встроенный ссылочный тип

```
#include <bitset>

class reference;
```

Этот встроенный класс является типом, возвращаемым функцией **bitset::operator[]** при применении к неконстантным объектам **bitset**.

Он обращается к отдельным битам с помощью интерфейса, имитирующего ссылку на **bool**.

```
class bitset::reference {
  friend class bitset;
  reference();                                    // нет открытого конструктора
public:
  ~reference();
  operator bool() const;                          // преобразование в bool
  reference& operator= (bool x);                  // присваивание bool
  reference& operator= (const reference& x);      // присваивание бита
  reference& flip();                              // инверсия
  bool operator~() const;                         // инверсия
}
```

Пример

```
#include <iostream>       // std::cout
#include <bitset>         // std::bitset

int main () {

  std::bitset<4> foo;

  foo[1] = 1;             // 0010
  foo[2] = foo[1];        // 0110

  std::cout << "foo: " << foo << '\n';

  return 0;
}
```

Вывод

```
foo: 0110
```

count() — вернуть количество установленных бит в битовом наборе

```
#include <bitset>

size_t count() const noexcept;
```

Возвращает количество битов в наборе битов, которые установлены в 1.

size() — вернуть количество бит в битовом наборе.

```
#include <bitset>

constexpr size_t size() noexcept;
```

Равен параметру шаблона, с которым создается экземпляр класса битового набора (N).

test() — вернуть значение бита

```
#include <bitset>

bool test (size_t pos) const;
```

Возвращает, установлен ли бит в позиции **pos** (то есть равен ли он единице).

В отличие от оператора доступа (**operator []**), эта функция выполняет проверку диапазона для **pos** перед извлечением битового значения, выбрасывая **out_of_range**, если **pos** равен или больше размера битового набора.

pos — позиция бита, значение которого извлекается.

Позиция отсчитывается от самого правого бита, позиция которого есть 0.

size_t — это целочисленный тип без знака.

Возвращаемое значение

true, если бит в позиции **pos** установлен, и **false**, если он не установлен.

any(), none(), all() — проверить биты

```
#include <bitset>
bool any() const noexcept;  // проверить, что хотя бы один установлен
bool none() const noexcept; // проверить, что ни одного не установлено
bool all() const noexcept;  // проверить, что все установлены
```

Пример

```
#include <iostream>       // std::cin, std::cout, std::boolalpha
#include <bitset>         // std::bitset

int main () {

  std::bitset<8> foo;

  std::cout << "Введите 8-разрядное двоичное число: ";
  std::cin >> foo;

  std::cout << std::boolalpha;
  std::cout << "all: " << foo.all() << '\n';
  std::cout << "any: " << foo.any() << '\n';
  std::cout << "none: " << foo.none() << '\n';

  return 0;
}
```

Операции с битами

set(), reset(), flip() — установить, сбросить, инвертировать биты

```
#include <bitset>

bitset& set( ) noexcept;                       // (1) установить все биты
bitset& set (size_t pos, bool val = true); // (2) установить один бит

bitset& reset( ) noexcept;                     // (1) сбросить все биты
bitset& reset (size_t pos);                    // (2) сбросить один бит

bitset& flip( ) noexcept;                      // (1) инвертировать все биты
bitset& flip (size_t pos);                     // (2) инвертировать один бит
```

(1) никогда не генерирует исключений (отсутствия исключения гарантировано).

(2) в случае исключения объект находится в допустимом состоянии.

Если **`pos`** больше или равно размеру bitset, функция выдает исключение **`out_of_range`**.

Операции с набором

to_string() — преобразовать в строку

```
#include <bitset>

template <class charT = char,
          class traits = char_traits<charT>,
          class Alloc = allocator<charT>>
basic_string<charT, traits, Alloc> to_string(charT zero = charT('0'),
                                             charT one  = charT('1')) const;
```

Создает объект **basic_string**, который представляет биты в битовом наборе как последовательность нулей и/или единиц.

Строка, возвращаемая этой функцией, имеет то же представление, что и результат, полученный путем вставки битового набора непосредственно в выходной поток с помощью оператора **<<**.

Шаблон функции использует параметры шаблона для определения типа возвращаемого значения.

Типы по умолчанию для этих параметров шаблона в качестве возвращаемого типа выбирают **string** .

zero, **one** — символьные значения для представления нуля и единицы.

Пример bitset::to_string

```
#include <iostream>       // std::cout
#include <string>         // std::string
#include <bitset>         // std::bitset

int main () {

  std::bitset<4> mybits;     // mybits: 0000
  mybits.set();              // mybits: 1111

  std::string cool_string =
  mybits.to_string<char,
                   cstd::string::traits_type,
                   std::string::allocator_type>();

  std::cout << "cool_string: " << cool_string << '\n';

  return 0;
}
```

Вывод

```
mystring: 1111
```

to_ulong(), to_ullong() — преобразовать в целое число без знака

```
#include <bitset>

unsigned long to_ulong( ) const;
unsigned long long to_ullong( ) const;
```

Возвращает беззнаковое длинное число с целочисленным значением, для которого установлены те же биты, что и для набора бит.

Если размер битового набора слишком велик для представления в значении типа **unsigned long** или **unsigned long long**, функция генерирует исключение типа **overflow_error**.

Пример

```
#include <iostream>       // std::cout
#include <bitset>         // std::bitset

int main () {

  std::bitset<4> foo;     // foo: 0000
  foo.set();              // foo: 1111

  std::cout << foo << "  как целое: " << foo.to_ulong() << '\n';
}
```

Вывод

```
1111 как целое: 15
```

Специализации классов

hash<bitset> — хеш для битового набора

```
#include <bitset>

template <class T> struct hash;                 // неспециализированный шаблон
template <size_t N> struct hash<bitset<N>>;  // специализизация для bitset
```

Класс унарного функционального объекта, который определяет специализацию хеширования для битового набора.

Функциональный вызов возвращает хеш-значение на основе битового набора, переданного в качестве аргумента. **Хеш-значение** — это значение, которое зависит исключительно от своего аргумента, возвращая всегда одно и то же значение для одного и того же аргумента (для данного прогона программы).

Ожидается, что возвращаемое значение будет иметь небольшую вероятность совпадения со значением, которое будет возвращено для другого аргумента (с вероятностью столкновения, приближающейся к **`1/numeric_limits <size_t>::max`**).

Это позволяет использовать объекты битового набора в качестве ключей для неупорядоченных контейнеров (таких как **`unordered_set`** или **`unordered_map`**).

Функция-член

`operator()` — возвращает хеш-значение для своего аргумента как значение типа **`size_t`**.