Семестр 2. Лекция 1. Шаблоны.

Евгений Линский

3 Марта 2017

Параметр шаблона: целое число

Битовое множество (доступ к отдельным битам):

```
template<size_t Size>
class Bitset{
private:
  char m[ (Size - 1) / 8 + 1 ];
public:
  bool get(size_t index) {
    // mask, &, «, etc
  }
};

Bitset<128> b1;
Bitset<7> b2;
```

Реализация стека

- Хранение элементов стека можно реализовать на списке или на векторе.

```
template <typename T, class Container>
class Stack {
private:
  Container c;
public:
  void push(const T& v);
};
```

- Проблема (потеря точности при хранении в s2):

```
Stack< int, List<int> > s1; //OK
Stack< double, Vector<int> > s2; //legal, but not OK!
```

Параметр шаблона: неистанциированный шаблон

- Неистанциированный шаблон — не заданы параметры шаблоны (Array вместо Array<int>)

```
template <typename T,
          template <typename> Container>
class Stack {
private:
  Container<T> c;
  ...
};

Stack< int, List > s1; // List<int>
Stack< double, Vector > s2; // Vector<double>
```

- Значение по умолчанию

```
template <typename T,
          template <typename> Container = Deque>
class Stack { ... };

Stack< int > s1; // Deque<int>
```

Специализация шаблона

- Идея: оптимизированная версия шаблонного класса под конкретный тип
- Общая версия:

```
template<typename T>
class Array{
private:
  T *a;
  ...
public:
  Array(size_t size) {
    a = new T [size]
    ...
  }
};
```

Специализация шаблона

- Реализация для bool аналогична реализации Bitset.

```
template <>
class Array<bool>{
private:
  T *a;
  ...
public:
  Array(size_t size) {
    a = new T [ (size-1)/8 + 1 ];
    ...
  }
};
```

- Частичная специализация

```
template<class T>
class Array< Array<T> > {
  T **a;
};
//Если T -- массив Array.
```

Виды ошибок

Виды ошибок:

- Ошибки “по вине программиста”. Примеры:

```
char *s = NULL;
size_t l = strlen(s);
Array a(-1);
```

  Обработка ошибок:

  - Лучше выявить на стадии тестирования (assert, unit test, etc).
  - При выполнении “идеальной” программы их не происходит.
  - Библиотека C подобные ошибки не обрабатывает.
  - Библиотека C++ — по-разному в разных местах: vector.at(i) и vector.operator[i].
  - Обрабатывть или нет — на усмотрение программиста.
- Ошибки “по вине окружения программы”. Примеры:
  - Файл не существует.
  - Сервер разорвал сетевое соединение.
  - Пользователь вместо числа ввел букву.

  Обработка ошибок:

  - Могут произойти и при выполнении “идеальной” программы.
  - **Обязательно надо обрабатывать!**

Обработка ошибок

- Проверка на наличие ошибки (if)
- Освободить ресурсы

```
delete [] array;
fclose(f);
```

- Сообщить пользователю и/или вызывающей функции

```
FILE *f = fopen("a.txt", "r");
if( f == NULL ) {
  printf("File a.txt not found\n");
}
//or
if( f == NULL ) {
  return -1;
}
```

- Предпринять действия по восстановлению от ошибки (например, не смогли соединиться — попробовать еще три раза)

Обработка ошибок в C-style

Информация об ошибки: через возвращаемое значение и через глобальную переменную

```
FILE* fopen(...) {
  if(file not found) {
    errno = 666;
    return NULL;
  }
  if(permission denied) {
    errno = 777;
    return NULL;
  }
  ...
}
```

Обработка ошибок в C-style

- Мало информации! Не знаем причину: нет файла, нет прав доступа, ...

```
FILE *f = fopen(...);
if( f == NULL ) {
  ...
}
```

- Глобальная переменная *errno* хранит код ошибки (*strerror(..)* — сообщение об ошибки)

```
#include <errno.h>
FILE *f = fopen(...);
if( f == NULL ) {
  switch(errno) {
    ...
  }
}
```

Почему не всем нравится C-style?

Attention! Holy war!

- Не всегда хватает диапазона возвращаемых значений функции

```
class Array {
  int *a;
public:
  //return -1 in case of index out-of-bound ???
  int get(size_t index);
};
int r1 = atoi("0");
int r2 = atoi("a");
```

- Код логики и обработка ошибок перемещены

```
r = fread(...);
if (r < ...) {
  //error
}
r = fseek(...);
if (r != 0 )  {
  //error
}
```

C++-style: исключения (exception)

```
class MyException {
private:
  char message[256];
  // possible fields: filename, line, function name
public:
  const char* get();

};

double divide(int a, int b) {
  if(b == 0) {
    throw MyException("Devision by zero");
  }
  return a/b;
}
```

C++-style: исключения (exception)

```
try {
  x = divide(c, d);
}
catch(MyException& e) {
  std::cout << e.get(); // 1. tell user
  // 2. delete [] ...; free resources
  // 3. throw e; inform caller function
}
```

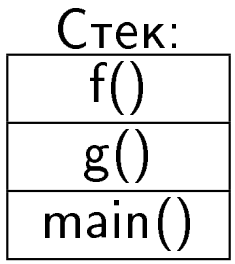
Stack unwinding - I

```
f() {
   if(...) throw MyException("Error: ....");
   printf(...);
}

g() { f(); }

main() {
   try {
      g();
   }
   catch(MyException& e) { ... }
}
```

Stack unwinding - II



Если “брошено” исключение:

- Нормальный процесс выполнения программы заканчивается, т.е. поток управления до *printf* в *f()* не дойдет.
- Начинается stack unwinding: последовательный просмотр стека до тех пор, пока не будет найден подходящий по типу исключения (в нашем примере тип *MyException*) блок *try/catch*.
- Если подходящий блок не был найден, и исключениие “вылетело” за *main()*, то программа аварийно завершается.

Несколько типов исключений - I

Если в программе несколько подсистем (GUI, Network, Model), то можно:

1. у каждой подсистемы сделать свой тип исключения (GuiException, NetworkException, ModelException)
2. обрабатывать их по-разному

```
main() {
  try {
    doGame();
  }
  catch(GuiException& e) {
    showMessageBox(...);
  }
  catch(NetworkException& e) {
    showMessageBox(...);
    logger.log(...)
  }
  catch(ModelException &e) {
    logger.log(...)
  }
}
```

Несколько типов исключений - II

- Можно организовать иерархию наследования исключений, чтобы не пропустить какое-нибудь в main.

```
class MyException {};
class GuiException : public MyException {};
class NetworkException : public MyException {};
class ModelException : public MyException {};
```

- Однако, надо помнить, что в блоке *try/catch* выбирается первый *catch*, подходящий по типу.
  - Всегда будет срабатывать первый *catch*

```
try { ... }
catch(MyException &e) { ... }
catch(GuiException &e) { ... }
```

  - Правильный порядок обработки

```
try { ... }
catch(GuiException &e) { ... }
catch(MyException &e) { ... }
```

- В STL все исключения — наследники *std::exception*

Как поймать исключение любого типа?

```
try {
  doMainWork();
}
catch(...) {   // ...  -- catch anything
  throw;       // without argument -- rethrow anything
}
```