Тема 2-3.

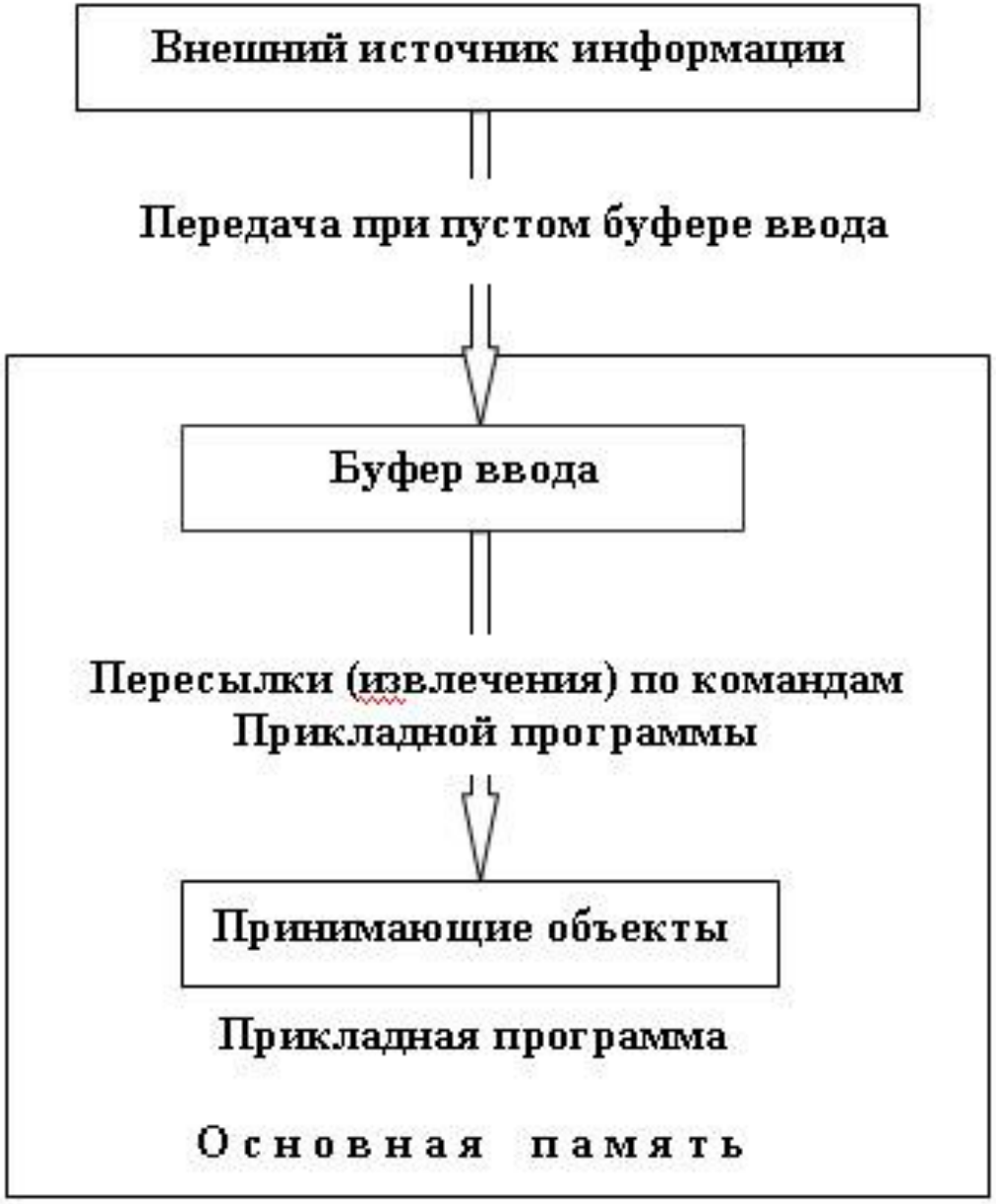
Потоки ввода вывода
Исключения

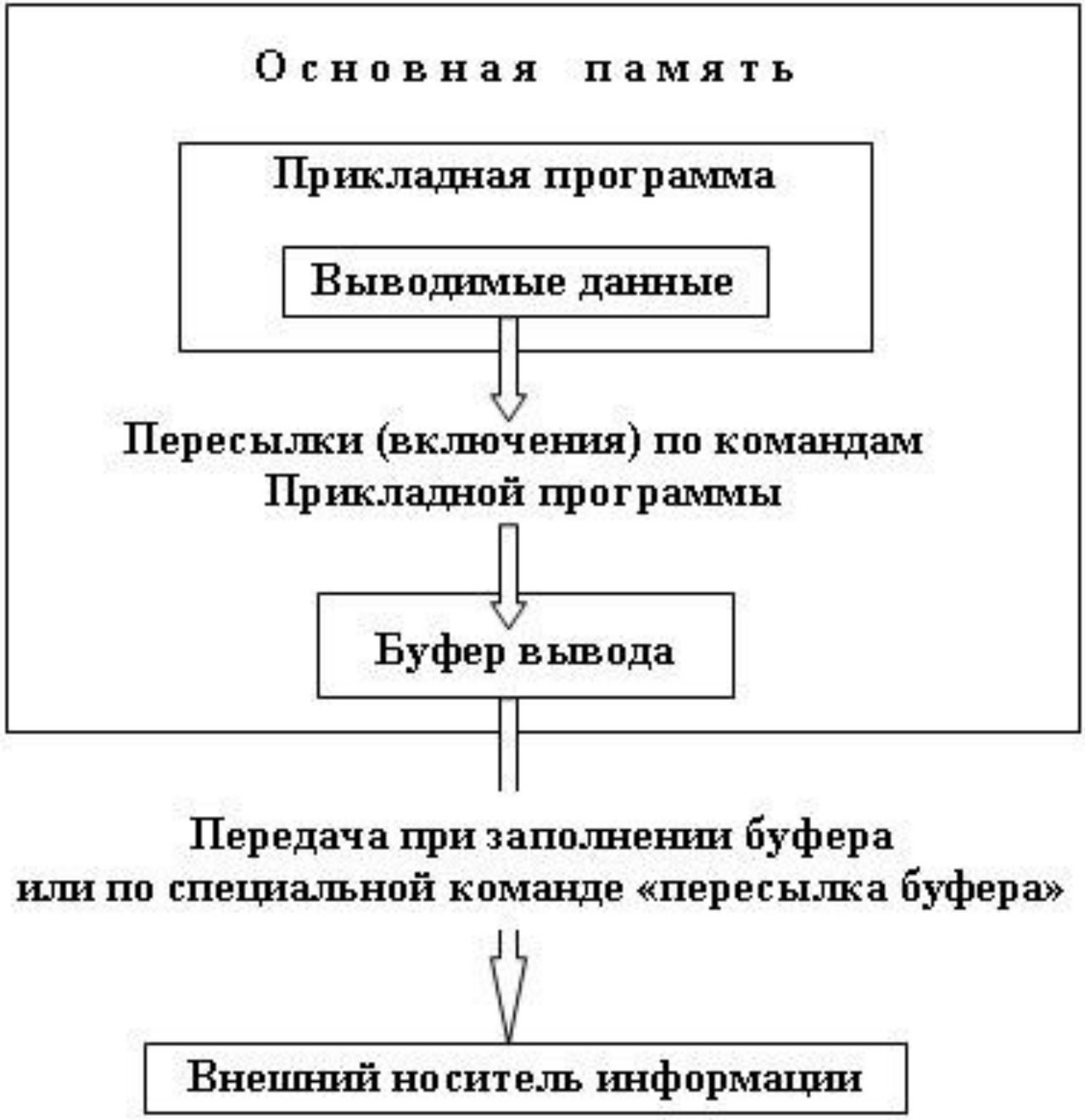
для АСУб и ЭВМб

Потоки ввода-вывода

Принцип "потокового ввода-вывода" следующий:

- В оперативной памяти средствами *операционной системы* создаётся некоторый "промежуточный буфер" для хранения данных, читаемых из *файла, устройства* или записывания на него этих данных;
- Средствами программы, созданной прикладным программистом, происходит чтение или запись информации в этот буфер;
- Средствами операционной системы осуществляется "синхронизация" этого буфера ("потока данных") с файлом или устройством;
- При создании или открытии файла для него выделяется буфер в оперативной памяти компьютера, а после закрытия файла этот буфер очищается.


Внешний источник информации
Передача при пустом буфере ввода
Буфер ввода
Пересылки (извлечения) по командам
Прикладной программы
Принимающие объекты
Прикладная программа
Основная память


Основная память
Прикладная программа
Выводимые данные
Пересылки (включения) по командам
Прикладной программы
Буфер вывода
Передача при заполнении буфера
или по специальной команде «пересылка буфера»
Внешний носитель информации

- При работе с потоками и файлами различают буферизированный (с использование буфера) и небуферизированный (без использования буферов) ввод-вывод.
- Буфер (buffer) представляет собой область оперативной памяти для промежуточного хранения данных, передаваемых между программой и внешним устройством.
- Вывод данных в поток с буфером приводит к выводу этих данных в соответствующий файл только после заполнения буфера.
- Вывод данных в **небуферизованный поток** приводит к немедленному выводу в файл.

Стандартные потоки ввода-вывода

«Стандартные потоки» присутствуют в операционной системе всегда и никогда не удаляются из оперативной памяти:

- *Стандартный поток ввода* (обозначение: stdin, cin и др.) - используется для ввода символьных данных в программу. По-умолчанию этот поток закреплён за клавиатурой компьютера.
- *Стандартный поток вывода* (обозначается как: stdout, cout и др.) - используется для вывода символьной информации, полученной в результате работы программы в "штатном режиме". По-умолчанию этот поток закреплён за экраном дисплея.
- Вывод данных на экран и чтение их с клавиатуры происходит потому, что по умолчанию стандартные потоки ассоциированы с терминалом пользователя. Это не является обязательным — потоки можно подключать к чему угодно — к файлам, программам и даже устройствам. В командном интерпретаторе ОС такая операция называется *перенаправлением*.
- Стандартные потоки можно перенаправлять не только в файлы, но и на вход других программ. Если поток вывода одной программы соединить с потоком ввода другой программы, получится конструкция, называемая каналом, конвейером или пайпом.

Стандартные потоки ввода-вывода

- **Стандартный поток ошибок** (обозначение: stderr, cerr и др.) - используется для вывода символьных *диагностических сообщений*, ошибок и предупреждений, возникших в результате работы программы. По-умолчанию этот поток закреплён за экраном дисплея;
- Примечание: стандартный поток и поток ошибок разделены в связи с тем, что при *перенаправлении вывода* часто совсем не нужно записывать в результаты работы программы диагностические сообщения.
- **Стандартный поток печати** (обозначение: stdprn и др.) - используется для вывода результатов работы программы на печать. По-умолчанию этот поток закреплён за текущим принтером в системе, подключённым к порту *LPT1*. В настоящее время этот поток почти не используется, поскольку чаще проще и безопаснее перенаправить *стандартный поток вывода* на принтер, чем разделять потоки отдельно для экрана и отдельно для принтера.
- Все остальные потоки создаются или уничтожаются с помощью функций открытия и закрытия файлов, на период чтения/записи/добавления информации в эти файлы.

Потоки ввода-вывода

- **Поток данных** – абстракция, используемая для чтения или записи данных в единой манере. Поддержка потоков включена в большинство языков программирования (C++, C#, Java).
- Существует два типа потоков: текстовые и двоичные.

Текстовые потоки

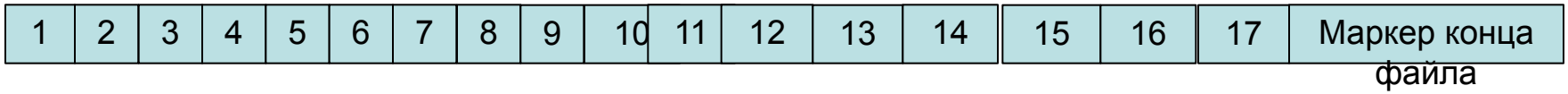
- *Текстовые потоки* - это последовательность символов. В текстовых потоках некоторые символы могут преобразовываться согласно требованиям среды.
- Поэтому может не быть однозначного соответствия между записываемыми или считываемыми символами и символами во внешнем устройстве.
- Например, символ новой строки может преобразовываться в пару «возврат каретки - перевод строки».

Двоичный поток

- *Двоичный поток* – это последовательность байт, имеющих однозначное соответствие с байтами во внешнем устройстве (преобразование символов не возникает).
- Число байт, записанных или прочитанных из внешнего устройства, совпадает с числом во внешнем устройстве. Может добавляться некоторое количество нулевых байт к двоичному потоку.

Файлы и потоки

В C++ каждый файл рассматривается как последовательный поток байтов.



Маркер конца файла – EOF (End Of File marker)

Потоки ввода-вывода

Отличительные особенности применения механизма потоков:

- буферизация при обменах с внешними устройствами;
- независимость программы от файловой системы конкретной операционной системы;
- контроль типов передаваемых данных;
- возможность удобного обмена для типов, определенных пользователем.

Потоки ввода-вывода C++

- Для использования стандартных потоков ввода-вывода достаточно поместить в текст программы препроцессорную процедуру:

  #include <iostream>

- Каждый раз при включении в программу библиотеки iostream происходит формирование объектов **cin**, **cout**, **cerr**, т.е. создаются соответствующие стандартные потоки.

Извлечение из потока

- Выполнение операции >> (**извлечение из потока**) заключается в преобразовании последовательности **символов** потока в значение типизированного объекта, частным случаем которого является переменная базового типа int, long, double и т.д.

cin >> идентификатор;

- Игнорирует пробелы, табы, переход на новые строки
- Возвращает 0 если достигнут конец файла, т.е. **EOF**
- Популярынй способ извлечение из потока:

while (cin >> grade)

- Вместо операций можно использовать функцию:

c=cin.get();

!!! **Считывает любой символ**

Извлечение из потока

- Функция getline извлекает данные из входного потока до строкового разделителя, который не записывается в получившийся массив данных.

cin.getline(stream, string, separator);

- где stream - это поток данных,
- string – переменная, в которую запишется строка
- separator – строковый разделитель, показывающий на конец строки.

- Последний параметр функции можно опустить, тогда будет задан сепаратор по умолчанию - '\n'.

Извлечение из потока

- Вместо операций можно использовать функцию:

cin.getline(stroka, size);
cin.getline(stroka, size, '\0');

- Определение конца файла потока

cin.eof()

Пример:

```
#include <iostream>

void main(){

int size=50;

char stroka[size];

cin.getline(stroka,size);

}
```

Включение в поток

- При выполнении операции << (**включение в поток**) осуществляется обратное преобразование – типизированное значение выражения (int, float, char и т.д.) трансформируется в последовательность символов потока.

cout << выражение;

cout.put(‘A’);

Сцепление операций

- **cout << 'значение1' << 'значение2' << ... << 'значение n';**

cin >> переменная1 >> переменная2 >>...>> переменная n;

```
setlocale(LC_ALL, "Russian");

int curNumber = 0;
int curNumber2 = 0;
std::string line;

cin >> curNumber >> curNumber2;
cout << curNumber<<" and " << curNumber2;
```

Строковые потоки <sstream>

- Для вывода в память строки существует специализированный тип **stringstream**
- Как правило, строковый поток ввода инициируется объектом класса string, а затем считывает из него символы, используя операции ввода.
- И наоборот, строковый поток вывода инициируется пустым объектом класса string, а затем заполняется с помощью операций вывода.
- Строковый поток ввода-вывода обычно используется для отделения собственно ввода-вывода от обработки данных

Строковые потоки

- stringstream strm; — создает несвязанный объект класса stringstream;
- stringstream strm(s); — создает объект класса stringstream, содержащий копию строки s;

- strm.str() — возвращает копию строки, которую хранит объект strm;
- strm.str(s) — копирует строку s в объект strm,

- eof() – проверяет, достигнут ли конец потока,
- fail() – проверяет, произошла ли исправимая ошибка,
- bad() – проверяет, произошла ли исправимая ошибка

```
#include <iostream>
#include <sstream>
using namespace std;
int main()
{
	setlocale(LC_ALL, "Russian");
	int curNumber = 0;
	std::string line;
	while (std::getline(std::cin, line))
	{
		std::stringstream ss(line);
		if (ss >> curNumber)
		{
			if (ss.eof())
			{

				break;
			}
		}
		std::cout << "Ошибка!" << std::endl;
	}
	cout << "Ответ: "<< curNumber;
}
```

Манипуляторы

- **Манипуляторами** называют специальные функции, позволяющие программисту изменять состояния и флаги потока.
- Особенность манипуляторов состоит в том, что их имена (без параметров) и вызовы (с параметрами) можно использовать в качестве правого операнда для операции обмена (<< или >>).
- В качестве левого операнда в этом выражении, как обычно, используется поток (ссылка на поток), и именно на этот поток оказывает влияние манипулятор.

Манипуляторы ввода/вывода

- Манипуляторы – это вспомогательные функци, которые позволяют управлять потоками ввода / вывода (например, std::cin, std::cout).
- std::endl - новая строка и передача данных из буфера на устройство (flush)
- Например,

```
std::cout << std::endl;
std::cout << std::endl <<
std::endl;
```

std::hex | std::oct | std::dec

- Выводят число в заданной системе счисления.
- Находятся в заголовке <ios>

```
std::cout << std::hex << 10 <<
std::endl;   // a
std::cout << std::oct << 0xFF <<
std::endl; // 377
std::cout << std::dec << 0771 <<
std::endl; // 505
```

std::showbase | std::noshowbase

- Отображают выводимую систему счисления
  Находятся в заголовке <ios>
-
```
std::cout << std::hex << 10 <<
std::endl; // a
std::cout << std::showbase << 10 <<
std::endl; // 0xa
```

std::boolalpha | std::noboolalpha

Эти флаги управляют выводом выражений типа bool
Находятся в заголовке <ios>

```
#include <iostream>

int main()
{
    bool flag = true;

    std::cout << flag << std::endl; // 1
    std::cout << std::boolalpha << flag <<
std::endl; // true
    std::cout << std::noboolalpha << flag <<
std::endl; // 1
}
```

std::showpoint | std::noshowpoint

- Управляет отображением плавающей точки (показывать / не показывать)
- Находятся в заголовке <ios>

```
std::cout << std::showpoint << 1.0 <<
std::endl; // 1.00000
std::cout << std::noshowpoint << 1.0 <<
std::endl; // 1
```

std::showpos | std::noshowpos

- Отображает знак ‘+’ перед положительными числами
- Находятся в заголовке <ios>

```
int value = 4;

std::cout << value; // 4
std::cout << std::endl <<
std::showpos;
std::cout << value << std::endl; //
+4
```

std::skipws | std::noskipws

- Поток ввода пропускает / не пропускает пробелы при форматировании
  Находятся в заголовке <ios>

```
char c1, c2, c3;

std::cin >> c1 >> c2 >> c3; // "a b c"

// c1 - a
// c2 - b
// c3 - c

std::cin >> std::noskipws >> c1 >> c2 >>
c3; // "a b c"

// c1 - a
// c2 - пробел
// c3 - b
```

std::uppercase | std::nouppercase

Данные флаги управляют регистром шестнадцатеричных чисел и экспоненциальных. Находятся в заголовке <ios>

```
std::cout << std::hex << std::showbase;
std::cout << std::uppercase << 0x1f <<
std::endl; // 0x1F
std::cout << std::nouppercase << 0x1f <<
std::endl; // 0x1f
```

std::setw(int n)

Старается выравнивать выводимые числа по заданной длине. По умолчанию – по правому краю. Сбрасывается после использования.
Находятся в заголовке <iomanip>

```
std::cout << std::setw(10) << 1002 << ' ' << 12 << std::endl;
//  "      1002 12"
```

std::setfill(CharT c)

- Позволяет задать символ для заполнения
  Находятся в заголовке <iomanip>

```
std::cout << std::setw(10);
std::cout << std::setfill('#');
std::cout << std::hex << std::showbase << 0xA <<
std::endl; //########0xa
```

std::left | std::right | std::internal

Управляют выравниванием при заданной ширине
Находятся в заголовке <ios>

```
std::cout << std::setw(10);
std::cout << std::setfill('#');
std::cout << std::hex << std::showbase << std::internal << 0xA <<
std::endl; //0x#######a
std::cout << std::setw(10) << std::left << 10 << std::endl; //
0xa#######
```

std::fixed | std::scientific | std::hexfloat std::defaultfloat

- Управляют выводом чисел с плавающей запятой.
- Находятся в заголовке <ios>

Вывод:

```
0.010000
1.000000e-02
0x1.47ae14p-7
0.01
```

```
std::cout
  << "The number 0.01 in fixed:      " << std::fixed        <<
0.01 << '\n'
  << "The number 0.01 in scientific: " << std::scientific   <<
0.01 << '\n'
  << "The number 0.01 in hexfloat:   " << std::hexfloat     <<
0.01 << '\n'
  << "The number 0.01 in default:    " << std::defaultfloat <<
0.01 << '\n';
```

std::setprecision(int n)

- Устанавливает количество выводимых разрядов в числе с плавающей точкой
- Находятся в заголовке <iomanip>

```
const long double pi = std::acos(-1.L);

std::cout << "default precision (6): " << pi << '\n'
          << "std::setprecision(10): " <<
std::setprecision(10) << pi << '\n';
```

Вывод:

```
3.14159
3.141592654
```

std::flush

- «Сбрасывает» данные из буфера выходного потока. Результат может быть не виден на конкретной машине, так как стандарт C++ не регламентирует, когда поток снова «очистит» буфер, но при использовании std::flush сброс обязан произойти.
- Находятся в заголовке <ostream>
-

std::quoted(const CharT *s) [c++14]

Данный манипулятор выводит текст в кавычках.
Находятся в заголовке <iomanip>

```
char string[] = "test";

std::cout << std::quoted(string);
// “test”
```

Вывод целого числа в двоичной форме

```
#include <iostream>
#include <bitset>

int main() {
    int a = -58, b = a>>3, c = -315;

    std::cout << "a = " << std::bitset<8>(a)  << std::endl;
    std::cout << "b = " << std::bitset<8>(b)  << std::endl;
    std::cout << "c = " << std::bitset<16>(c) << std::endl;
}
```

```
a = 11000110
b = 11111000
c = 1111111011000101
```

Обработка исключений

- Иногда при выполнении программы возникают ошибки, которые трудно предусмотреть или предвидеть.
- Такие ситуации называются **исключениями**. Язык C++ предоставляет разработчикам возможности для обработки таких ситуаций.
- Для этого в C++ предназначена конструкция try...catch...finally.

Исключения

- Это ошибки, возникающие во время работы программы
- Они могут быть вызваны различными обстоятельствами:
  - выход за пределы памяти
  - ошибка открытия файла
  - попытка инициализировать объект недопустимым значением
  - использование индекса, выходящего за пределы вектора

Синтаксис исключений

Механизм обработки исключительных ситуаций основан на трех ключевых словах: **try, catch, throw**

```
try {
// код, подлежащий контролю
// функции могут генерировать исключительные ситуации
// может содержать несколько оператор или целую программу
}
catch (тип1 аргумент) {
// перехват исключительных ситуаций
}
catch (тип2 аргумент) {
//
}
…
catch (типN аргумент) {
//
}
```

throw исключительная ситуация;

Оператор throw генерирует указанную исключительную ситуацию. Если в программе есть ее перехват, оператор throw должен выполняться либо внутри блока try, либо внутри функции, явно или неявно вызываемой внутри блока try

Если генерируется исключительная ситуация, для которой **НЕ ПРЕДУСМОТРЕНА ОБРАБОТКА**, программа может прекратить свое выполнение. В этом случае вызывается стандартная функция **terminate**(), которая по умолчанию вызывает функцию **abort**()

Исключения: пример

```
setlocale(LC_ALL, "Russian");

cout << "начало \n";
try {
	cout << "внутри блока try \n";
	throw 100;		// генерируем ошибку
	cout << "этот оператор не выполняется";
}
catch (int i) {	// перехват ошибки

	cout << "перехват исключительной ситуации – значение равно : ";
	cout << i << "\n";
}
cout << "конец";
```

Исключения

- Код, который потенциально может сгенерировать исключение помещается в блок try.
- Если ошибка исправлена, то выполнение программы возобновляется с оператора, следующего за блоком catch.
- Если исправить ошибку невозможно, то в блок catch можно вызвать функции **exit**() или **abort**(), прекращает выполнение программы
- Код блока catch выполняется только при перехвате исключительной ситуации
- Исключительная ситуация может иметь любой тип, в том числе быть объектом класса, определенного пользователем
- Если необходимо обрабатывать не отдельные типы исключительных ситуаций, а перехватывать все подряд, то применяется следующий вид оператора catch:

catch (…)

Исключения

- Все исключения в языке C++ описываются типом **exception**, который определен в заголовочном файле <exception>
- Если мы хотим отловить исключения типа **exception**, то нам надо в выражении catch определить переменную этого типа

catch (std::exception err)

- Все типы исключений имеют метод **what()**, который возвращает информацию об ошибке

Исключения

- **runtime_error**: общий тип исключений, которые возникают во время выполнения
- **range_error**: исключение, которое возникает, когда полученный результат превосходит допустимый диапазон
- **overflow_error**: исключение, которое возникает, если полученный результат превышает допустимый диапазон
- **underflow_error**: исключение, которое возникает, если полученный в вычислениях результат имеет недопустимые отрицательное значение (выход за нижнюю допустимую границу значений)
- **logic_error**: исключение, которое возникает при наличии логических ошибок к коде программы
- **domain_error**: исключение, которое возникает, если для некоторого значения, передаваемого в функцию, не определено результата
- **invalid_argument**: исключение, которое возникает при передаче в функцию некорректного аргумента
- **length_error**: исключение, которое возникает при попытке создать объект большего размера, чем допустим для данного типа
- **out_of_range**: исключение, которое возникает при попытке доступа к элементам вне допустимого диапазона

Исключения

- Конструкция **try...catch** может использовать несколько блоков catch для обработки различных типов исключений. При возникновении исключения для его обработки будет выбран тот, который использует тип возникшего исключения.

- При использовании нескольких блоков catch вначале помещаются блоки catch, которые обрабатывают более частные исключения, а только потом блоки catch с более общими типами исключений

Исключения

```
try {
   // code to try
}
catch (exceptionType1 &name1) {
   // handle exceptions of exceptionType1
}
catch (exceptionType 2&name2) {
   // handle exceptions of exceptionType2
}
catch (exceptionType3 &name3) {
   // handle exceptions of exceptionType3
}
...
/* далее код*/
```

Сработает только первый по порядку!
(не switch)