Исключительные ситуации

Ниже представлен пример программы, генерирующей исключительную ситуацию "деление на 0".

Пример. Деление на ноль

```
#include <iostream>
using namespace std;
int main() {

	int a = 0, b = 10;
	cout << b / a << endl;
	return 0;
}
```

Операторы try catch и throw

Исключение — это событие при выполнении программы, которое приводит к её ненормальному или неправильному поведению.

Существует два вида исключений:

Аппаратные (структурные, SE-Structured Exception), которые генерируются процессором. К ним относятся, например,

- деление на 0;
- выход за границы массива;
- обращение к невыделенной памяти;
- переполнение разрядной сетки.

Программные, генерируемые операционной системой и прикладными программами – возникают тогда, когда программа их явно инициирует. Когда встречается аномальная ситуация, та часть программы, которая ее обнаружила, может сгенерировать, или возбудить, исключение.

Механизм структурной обработки исключений позволяет однотипно обрабатывать как программные, так и аппаратные исключения.

Для реализации обработки исключений в C++ используйте выражения **try**, **throw** и **catch**.

Общая схема try throw и catch

```
try {
	//блок_try
}
catch (тип1 arg) {
	//блок_catch_1
}
catch (тип2 arg) {
	//блок_catch_2
}
catch (тип3 arg) {
	//блок_catch_3
}
// . . .
catch (типN arg) {
	//блок_catch_N
}
catch (...) {
	//блок ...
}
```

Сразу за блоком **try** находится защищенный раздел кода. Выражение **throw** вызывает исключение, т.е. создает его.

Блок кода после **catch** является обработчиком исключения. Он перехватывает исключение, вызываемое, если типы в выражениях **throw** и **catch** совместимы. Если оператор **catch** задает многоточие (...) вместо типа, блок **catch** обрабатывает все типы исключений.

Поскольку блоки **catch** обрабатываются в порядке программы для поиска подходящего типа, обработчик с многоточием должен быть последним обработчиком для соответствующего блока **try**.

Пример. Обработка исключения с помощью try throw и catch

```
#include <iostream>
using namespace std;
int main() {
	int a = 0, b = 10;
	try {
		if (a == 0)
		{
			throw 0;
		}
		cout << b / a << endl;
	}
	catch (...) { cout << "error\n"; }
	return 0;
}
```

Задания

Задание 1. Определите ситуации, когда программа, представленная, ниже будет заканчиваться аварийной остановкой. Добавьте в программу обработку исключительной ситуации.

Пример к заданию 1

```
#include <iostream>
#include <ctime>
using namespace std;

void main()
{
	setlocale(0, "");
	int n1, n2;
	int k = 2;
	srand(time(NULL));
	do
	{	n1=round(rand()%10);
		n2 = round(rand() % 3);
		cout << n1 <<" / "<< n2<<" = " << n1 / n2 << endl;
	} while (n1);
	cout << "Программа завершила работу!" << endl << endl;
}
```

Задание 2. Воспользуйтесь примером к данному заданию и напишите следующие программы:

1. Генерируется случайным образом число в диапазоне от -100 до 100. Вывести ИСТИНА, если число положительное, и ЛОЖЬ в противном случае.
2. Генерируются случайным образом 2 числа в диапазоне от 3 до 7. Вывести ИСТИНА, если произведение чисел равно 21, и ЛОЖЬ в противном случае.

Пример к заданию 2

```
#include <iostream>
using namespace std;

void main()
{
	setlocale(0, "");
	int x;
	cout << "Какое число меньше 5?\n";
	cin >> x;
	try {
		if (x >= 5)  throw x;
		cout << "Правильно!\n";
	}
	catch (int) { cout << "Не верно!\n"; }
}
```

Задание 3. Воспользуйтесь примером из задания 2 и напишите следующую программу:

Генерируется случайным образом число - количество собранного урожая в тоннах. Пользователю необходимо перевести его в значение, выраженное в граммах. Вывести ИСТИНА, если пользователь ввел правильный ответ, и ЛОЖЬ в противном случае.

Задание 4. Напишите следующую программу:

Генерируются случайным образом 3 числа в диапазоне от 0 до 10. Нужно посчитать отношение этих чисел. Предусмотрите в программе все возможные исключения.

Пример к заданию 4

```
#include <iostream>
using namespace std;

void main()
{
	setlocale(0, "");
	int x;
	cout << "Введите целое число\n";
	cin >> x;
	try {
		if (x == 0) { throw 0; }
		else if (x<0) { throw -1.0; };
		cout << "Правильно!\n";
	}
	catch (int x1) { cout << "Пожалуйста, введите число не равное нулю\n"; }
	catch (double x2) { cout << "Пожалуйста, введите число больше нуля\n"; }
}
```

Задание 5. Напишите следующую программу:

Доступ к паролю имеет пользователь под номером 7. Всего пользователей 9. Пароль генерируются каждый день – это трехзначное число. Вывести пароль на экран лишь в том случае, если запрос пароля произошел от пользователя 7. Все остальные случаи – это исключительная ситуация.

Макрос assert

assert — это макрос препроцессора, который обрабатывает условное выражение во время выполнения. Если условное выражение истинно, то оператор assert ничего не делает. Если же оно ложное, то выводится сообщение об ошибке, и программа завершается.

Сам assert реализован в **заголовочном файле** <cassert> и часто используется как для проверки корректности переданных параметров функции, так и для проверки **возвращаемого значения функции**.

Макрос assert() добавляет к программе процедуру диагностики. После выполнения, если выражение ложно (то есть, результат сравнения 0), assert() пишет информацию о вызове в поток stderr и вызывает функцию abort(). Информация, которая пишется в stderr включает в себя:

- имя файла с исходным кодом (предопределённый макрос __FILE__)
- строка у файла с исходным кодом (предопределённый макрос __LINE__)
- функция в исходном коде (предопределённый макрос __func__) (добавлено в стандарте C99)
- текст выражения, значение которого равно нулю 0

Пример 1. assert

```
#include <iostream>
#include <cassert>
using namespace std;

int main()
{
	setlocale(0, "");
	int a = 11, b = 10, c;

	cout<<"Если "<<a<<" меньше "<<b<<" то все хорошо, иначе ОШИБКА! "<<endl;

	//Проверка условия
	if (a>b) c=0;
	else c = 1;
	//Проверка верности функционирования программы
	assert(c);

	cout << "Программа отработала корректно." << endl;
	return 0;

}
```

Для того, чтобы отключить проверку, не обязательно исключать её из кода или комментировать объявление макроса, достаточно лишь объявить ещё один макрос — `NDEBUG` в программе перед `#include <assert>`:

```
#define NDEBUG
```

тогда объявление макроса `assert()` будет иметь следующий вид:

```
#define assert(ignore)((void) 0)
```

и поэтому никак не будет влиять на работу программы.

Пример 2. Отключили assert

```
#include <iostream>
#define NDEBUG
#include <cassert>
using namespace std;

int main()
{	setlocale(0, "");
	int a = 11, b = 10, c;

	cout<<"Если "<<a<<" меньше "<<b<<" то все хорошо, иначе ОШИБКА! "<<endl;

	//Проверка условия
	if (a>b) c=0;
	else c = 1;
	//Проверка верности функционирования программы
	assert(c);

	cout << "Программа отработала корректно." << endl;
	return 0;
}
```

В примере, представленном ниже если assert сработает, то строка будет включена в сообщение assert, так как строка всегда имеет значение true.

Пример 3. Проверяем параметры функции

```
#include <iostream>
//#define NDEBUG
#include <cassert>
using namespace std;

double multAB(double &a, double &b){
	assert((a != 0) && (b != 0)&&" a && b must be more than 0 ");
	return a*b;
}

int main()
{
	double a = 0, b= 0;
	cout<<a<<" * "<<b<<" = "<<multAB(a,b)<<endl;
	return 0;
}
```


Задание 5. Придумайте 3 функции, в которых используйте проверку входных данных с помощью assert.

Контрольные вопросы

1. Что такое исключение? Приведите несколько примеров исключительных ситуаций.
2. Что располагают в блоке try?
3. Где располагают блок throw?
4. Какую функцию выполняет блок catch?
5. Что происходит, если в блоке catch вместо типа задать многоточие (...)?
6. Для чего нужен asset?
7. Если assert выполняется, что происходит с программой?
8. Что возвращает assert?
9. Что является параметром assert?
10. Каким образом выводить на экран дополнительное сообщение во время выполнения assert?