

# Структуры Дружественные и вложенные классы Исключения

# Структуры – открытые классы

```
struct s {  
    int i;  
    void print() {  
        cout<<i;  
    }  
};
```

```
s vs;  
vs.i=0;    // ok  
vs.print(); // ok
```

все public

```
class c {  
    int i;  
    void print() {  
        cout<<i;  
    }  
};
```

```
c vc;  
vc.i=0;    // error  
vs.print(); // error
```

все private

# Дружественные классы

```
class A {  
    private:  
    ...  
    public :  
    friend class B;  
    ...  
};  
class B {  
    //имеет доступ к private-элементам класса A  
};
```

# Методы класса могут быть дружественными

```
class A {  
    friend void B:: method();  
};
```

```
class B{  
    public:  
    void method();// имеет доступ к private класса A  
};
```

порядок описания?

нужно опережающее определение

```
class A;  
class B{  
    public:  
    void method();// только объявление прототипа  
};
```

```
class A {  
    friend void B:: method();  
};  
void B::method(){  
    // реализация  
}
```

# Вложенные классы

```
class List
{
  class Node
  {
    int item;
    Node * next;
    Node (int val) : item(val), next(0) { }
  };
  Node *f,*t;
public:
  void add ( int a);
  . . .
};
```

```
void List::add(int a)
{
    Node * p= new Node(a);
    ...
}
```

- ▶ Класс Node недоступен вне класса List
- ▶ чтобы он был доступен наследникам, его следует определять в разделе protected
- ▶ Если класс Node будет определен в разделе public он будет доступен вне List, но ссылкой на него через спецификатор класса

```
List::Node * p;
```



# Исключения

- ▶ Ошибки выполнения программы
  - ошибки в данных
  - ошибки при работе с памятью
  - ошибки при доступе к файлам
- ▶ Если ошибка в функции, она может:
  - аварийно завершить выполнение (`exit()`, `abort()`)
  - выдать диагностику и завершить выполнение
  - вернуть признак ошибки (например, EOF)
- ▶ Когда функция возвращает результат, то как поступать в случае ошибки?

# «Старый» подход

```
double calc(double a, double e)
```

заменяем на

```
bool calc ( double a, double e, double &res)
{
    if (...) res=...; return true;
    else return false;
}
```

**ВЫЗОВ**

```
if (calc(a,e,r)) cout<<r;
    else cout<<"ERROR";
```

# «Старый» подход

Почему это плохо?

Можно игнорировать результат функции

```
calc(a,e,r);  
cout<<r;
```

Результат может быть любым

- ▶ С++ вводит для этих целей механизм исключительных ситуаций
- ▶ Исключения позволяют передавать управление в другую часть программы при возникновении нестандартных ситуаций

- ▶ Исключения генерируются программным путем там, где нужно прервать выполнение из-за ошибки и передать его в другую часть программы
- ▶ Оператор генерации исключения

```
throw <исключение>;
```

```
double calc ( double a, double e)
{
    if (...) res=...; return res;
    else throw “Ошибка вычисления”;
}
```

# Перехват исключений

- ▶ вызов в блоке try

```
try {  
    cout << calc(a,e);  
}  
catch ( char * s )  
{  
    cout << s<<<endl;  
    // что делать в случае ошибки??,  
    // можно ввести новые а, е и вернуться к  
    // вычислению  
}
```

```
while (cin >> a>>e)
{
    try {
        cout << calc(a,e);
    }
    catch ( char * s)
    {
        cout << s<<endl;
        continue;
    }
}
```



```
try {  
    // Программный код, который может  
    генерировать исключения  
}  
catch (type1 id1){  
    // Обработка исключений типа type1  
}  
//Здесь продолжится нормальное  
выполнение программы...
```

```
try {  
    // Программный код, который может генерировать  
    исключения  
}  
catch (type1 id1){  
    // Обработка исключений типа type1  
}  
...  
catch (typeN idN){  
    // Обработка исключений типа typeN  
}  
//Здесь продолжится нормальное выполнение  
программы...
```

# Классы исключений

- ▶ Позволяют передать больше информации о характере исключения
- ▶ Позволяют различать и обрабатывать отдельно разные исключения в одном блоке `try`

```
class Problem1  
{  
};
```

```
class Problem2  
{  
    public:  
    int code;  
    Problem (int c) : code(c) {}  
};
```

## генерация исключений

```
throw Problem1 ();  
throw Problem2(1);  
throw Problem2(2);
```

обработка

```
catch( Problem1 )
```

```
{
```

```
...
```

```
}
```

```
catch (Problem2 &p)
```

```
{
```

```
    cout << p.code;
```

```
}
```

Обработчик для любых типов исключений

```
catch (...){
```

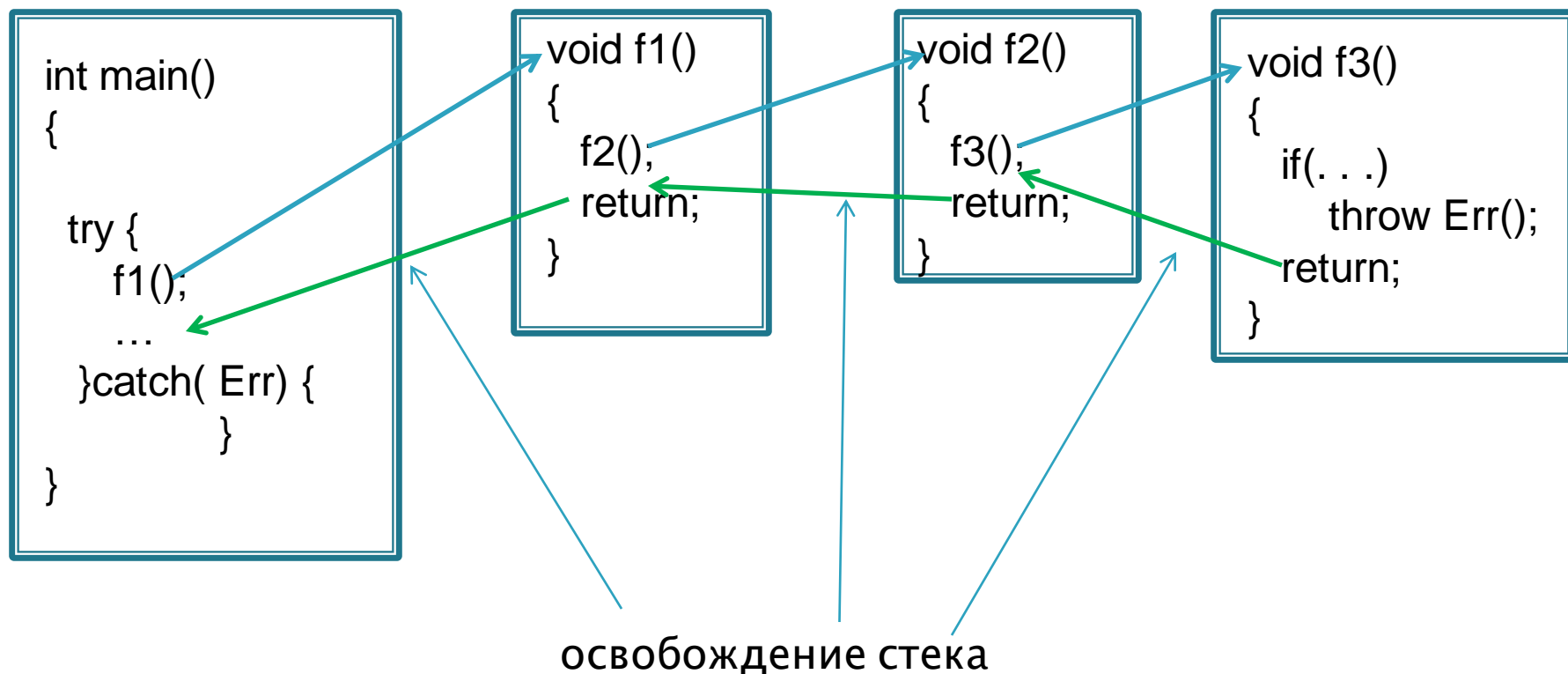
```
}
```

- ▶ Для уточнения видов исключений генерируемых функцией рекомендуется включать их в прототип функции

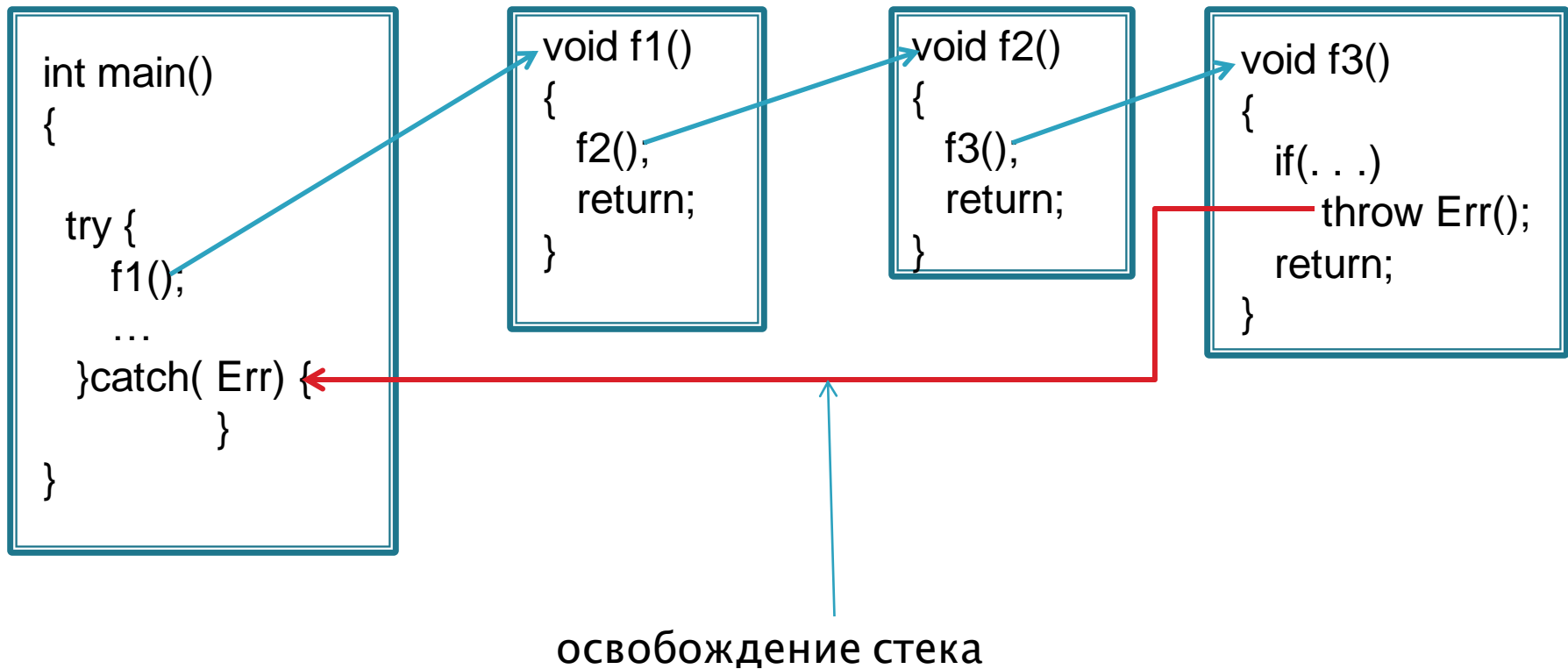
```
double calc (double a, double e) throw (char *);  
double charAt( char *s, int i) throw (Problem1);  
void copyFile ( char *nm, char* cp)  
                throw (FileNotFoundException, Error);
```



# Разворачивание стека



# Разворачивание стека



- ▶ Если исключения относятся к проблемам класса, можно определить исключение как вложенный класс.
- ▶ Это позволяет предотвратить конфликт имен

```
class ArrayDin {  
    private :  
        unsigned int size;  
        double * arr;  
    public :  
        class BadIndex {  
            public :  
                int idx;  
                BadIndex (int i) : idx(i) { }  
        };  
        ArrayDin ();  
        double & operator [ ] (int i) throw (BadIndex &);  
        . . .  
};
```

```
double & ArrayDin:: operator[ ](int i) throw (BadIndex &)  
{  
    if ( i<0 || i >= size) throw BadIndex(i);  
    return arr [ i ];  
}
```

```
ArrayDin mas;
```

```
....
```

```
try {
```

```
    cin>>i;
```

```
    cout << mas[ i ];
```

```
}
```

```
catch (ArrayDin::BadIndex &er)
```

```
{
```

```
    cout<< "ошибка – " << er.idx;
```

```
}
```

# Класс exception

- ▶ заголовочный файл

<exception>

Можно генерировать исключения класса или создавать свои классы – наследники

Класс содержит одну виртуальную функцию, возвращающую строку

```
virtual const char* what();
```

- ▶ наследник может определить свою реализацию

```
class MyException : public exception
{
    public :
        MyException () : exception() { }
        const char* what( )
        { return "function was interrupted\n";}
};
```



- ▶ этот метод используется при обработке исключения

```
catch ( MyExcep & p)
{
    cout << p.what( );
}
```

# bad\_alloc

- ▶ Класс – исключение, наследник exception
- ▶ Описан в заголовочном файле <new>
- ▶ Исключение генерируется при ошибке распределения памяти оператором new

```
Big * p;  
try {  
    p = new Big [1000];  
}  
catch ( bad_alloc &b)  
{  
    cout<< b.what ( );  
}
```

- ▶ Если исключение генерируется между явными операциями выделения и освобождения памяти, то возможна утечка памяти

```
void test (int n) {  
    double * ar= new double [n];
```

```
if (???) throw exception();
```

```
    delete [ ] ar;  
    return;  
}
```

при разворачивании стека переменная ar  
удаляется, а память не освобождается!

- ▶ рекомендуется перехватывать и обрабатывать исключение в самой функции

```
void test (int n) {  
    double * ar= new double [n];  
  
    try {  
        if (???) throw exception();  
    } catch (exception & e) {  
        delete [ ] ar;  
        throw 1;  
    }  
    delete [ ] ar;  
    return;  
}
```

- ▶ Нужно следить, чтобы при обработке исключений не оставались открытыми файлы

# Проблемы

- ▶ Функция выбрасывает исключение, отличающееся от тех, которые указаны в его описании (это не относится к функциям, у которых в прототипе не указаны вообще ни какие исключения)
- ▶ Такое исключение называется непредвиденным
- ▶ При этом вызывается функция `terminate()`, которая прервет выполнение программы, вызвав системную функцию `abort()`

- ▶ Можно вместо функции abort() подставить в terminate() любую свою функцию типа void без параметров

```
void myQuit () {  
    cout <<"Что-то пошло не так!!!"<<endl;  
    exit(-5);  
}
```

```
set_terminate(myQuit); // заменили
```



# Проблемы

- ▶ Если выброшено исключение, но оно не перехватывается (нет блока `try` или нет подходящего блока `catch`), оно определяется как неперехваченное и приводит к прерыванию программы.
- ▶ При этом вызывается функция `unexpected()`, которая вызывает функцию `terminate()` → `abort()`
- ▶ Можно заменить, вызвав функцию `set_terminate( ...);`