Object-Oriented Programming (OOP)
Lecture No. 45

Resource Management

- Function acquiring a resource must properly release it
- Throwing an exception can cause resource wastage

Example

```
int function1(){
        FILE *fileptr =
        fopen("filename.txt","w");
        ...
        throw exception();
        ...
        fclose(fileptr);
        return 0;
}
```

Resource Management

- In case of exception the call to close will be ignored

First Attempt

```
int function1(){
        try{
                FILE *fileptr = fopen("filename.txt","w");
                fwrite("Hello  World",1,11,fileptr);
                ...
                throw exception();
                fclose(fileptr);
        } catch(...) {
                fclose(fileptr);
                throw;
        }
        return 0;
}
```

Resource Management

- There is code duplication

Second Attempt

```
class FilePtr{
        FILE * f;
public:
        FilePtr(const char *name,
                    const char * mode)
                { f = fopen(name, mode);}
        ~FilePtr()          {  fclose(f);}
        operator FILE * ()      {  return f; }
};
```

Example

```
int function1(){
        FilePtr file("filename.txt","w");
        fwrite("Hello  World",1,11,file);
        throw exception();
        ...
        return 0;
}
```

Resource Management

- The destructor of the FilePtr class will close the file
- Programmer does not have to close the file explicitly in case of error as well as in normal case

Exception in Constructors

- Exception thrown in constructor cause the destructor to be called for any object built as part of object being constructed before exception is thrown
- Destructor for partially constructed object is not called

Example

```
class Student{
        String FirstName;
        String SecondName;
        String EmailAddress;
        …
}
```

- If the constructor of the SecondName throws an exception then the destructor for the First Name will be called

Exception in Initialization List

- Exception due to constructor of any contained object or the constructor of a parent class can be caught in the member initialization list

Example

```
Student::Student (String aName) :
   name(aName)
/*The constructor of String can throw a
   exception*/
{
      ...
}
```

Exception in Initialization List

- The programmer may want to catch the exception and perform some action to rectify the problem

Example

```
Student::Student (String aName)
  try
      : name(aName) {
      ...
  }
  catch(…) {
  }
```

Exceptions in Destructors

- Exception should not leave the destructor
- If a destructor is called due to stack unwinding, and an exception leaves the destructor then the function std::terminate() is called, which by default calls the std::abort()

Example

```
class Exception;
class Complex{
        …
public:
        ~Complex(){
                throw Exception();
        }
};
```

Example

```
int main(){
        try{
                Complex obj;
                throw Exception();
                …
        }
        catch(…){
        }
        return 0;
}
// The program will terminate abnormally
```

Example

```
Complex::~Complex()
{
        try{
                throw Exception();
        }
        catch(…){
        }
}
```

Exception Specification

- Program can specify the list of exceptions a function is allowed to throw
- This list is also called throw list
- If we write empty list then the function wont be able to throw any exception

Syntax

```
void Function1() {…}
void Function2() throw () {…}
void Function3( )
    throw (Exception1, …){}
```

- Function1 can throw any exception
- Function2 cannot throw any Exception
- Function3 can throw any exception of type Exception1 or any class derived from it

Exception Specification

- If a function throws exception other then specified in the throw list then the function unexpected is called
- The function unexpected calls the function terminate

Exception Specification

- If programmer wants to handle such cases then he must provide a handler function
- To tell the compiler to call a function use set_unexpected

Course Review

Object Orientation

- What is an object
- Object-Oriented Model
  - Information Hiding
  - Encapsulation
  - Abstraction
- Classes

Object Orientation

- Inheritance
  - Generalization
  - Sub-Typing
  - Specialization
- "IS-A" relationship
- Abstract classes
- Concrete classes

Object Orientation

- Multiple inheritance
- Types of association
  - Simple association
  - Composition
  - Aggregation
- Polymorphism

Classes – C++ Constructs

- Classes
  - Data members
  - Member functions
- Access specifier
- Constructors
- Copy Constructors
- Destructors

Classes – C++ Constructs

- this pointer
- Constant objects
- Static data member
- Static member function
- Dynamic allocation

Classes – C++ Constructs

- Friend classes
- Friend functions
- Operator overloading
  - Binary operator
  - Unary operator
  - operator []
  - Type conversion

Inheritance – C++ Constructs

- Public inheritance
- Private inheritance
- Protected inheritance

- Overriding
- Class hierarchy

Polymorphism – C++ Constructs

- Static type vs. dynamic type
- Virtual function
- Virtual destructor
- V-tables

- Multiple inheritance
- Virtual inheritance

Templates – C++ Constructs

- Generic programming
- Classes template
- Function templates
- Generic algorithm
- Templates specialization
  - Partial Specialization
  - Complete specialization

Templates – C++ Constructs

- Inheritance and templates
- Friends and templates

- STL
  - Containers
  - Iterators
  - Algorithms

Writing Reliable Programs

- Error handling techniques
  - Abnormal termination
  - Graceful termination
  - Return the illegal value
  - Return error code from a function
  - Exception handling