C++ Exception Handling

Dr. Md. Humayun Kabir
CSE Department, BUET

Exception Handling

- An exception is an unusual behavior of a program during its execution
- Exception can occur due to
  - Wrong user input
  - Wong run-time environment
- Exceptions do not occur always. However, abnormally terminates the program execution when occur
- Compiler cannot report whether a part of a code will cause exception or not
- C++ allows programmers to anticipate and handle exceptions

Exception Due to Wrong User Input

```
#include <iostream>
using namespace std;
int main(){
    int a, b;
    cout<<"Enter a whole number a: ";
    cin>>a;
    cout<<"Enter another whole number b: ";
    cin>>b;
    cout<<"The result of a/b is: "<<a/b<<endl; //exception if b=0
    return 0;
}
```

Exception Due to Wrong User Input

```
#include <iostream>
using namespace std;
int main(){
	int a, b;
	cout<<"Enter a whole number a: ";
	cin>>a;
	cout<<"Enter another whole number b: ";
	cin>>b;
	if(b!=0){
		cout<<"The result of a/b is: "<<a/b<<endl;
	}
	else {
		cout<<“Can’t divide by zero”<<endl;
	}
	return 0;
}
```

Exception Due to Wrong User Input

```
#include <iostream>
using namespace std;
const int divideByZero=0;
int main(){
	int a, b;
	cout<<"Enter a whole number a: ";
	cin>>a;
	cout<<"Enter another whole number b: ";
	cin>>b;
	try{
		if(b!=0)
			cout<<"The result of a/b is: "<<a/b<<endl;
		else
			throw divideByZero;
	}
	catch (int e){
		if(e==divideByZero) cout<<“Can’t divide by zero”<<endl;
	}
	return 0;
}
```

Exception Due to Wrong User Input

```
#include <iostream>
using namespace std;
int divide(int a, int b){
	if(b!=0)
		return a/b;
	else
		cout<<“Can’t divide by zero”<<endl;
		//will give warning, function ends without returning an integer
}
int main(){
	int a, b;
	cout<<"Enter a whole number a: ";
	cin>>a;
	cout<<"Enter another whole number b: ";
	cin>>b;
	cout<<"The result of divide (a, b) is: "<<divide (a,b)<<endl;
	return 0;
}
```

Exception Due to Wrong User Input

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

const int divideByZero=0;

int divide(int a, int b) throw (int){
    if(b!=0)
        return a/b;
    else
        throw divideByZero;
}
```

Exception Due to Wrong User Input

```
int main(){
    int a, b;
    cout<<"Enter a whole number a: ";
    cin>>a;
    cout<<"Enter another whole number b: ";
    cin>>b;
    try{
        cout<<"The result of divide (a, b) is: "<<divide (a,b)<<endl;
    }
    catch (int e) {
        if (e==divideByZero) cout<<“Can’t divide by zero”<<endl;
    }
    return 0;
}
```

Exception Due to Wrong Run-time Environment

```
#include <iostream>
#include <cstring>
using namespace std;
int main(){
    char str[100]="Long Live Bangladesh";
    char *p;
    p=new char[100000]; //exception if new cannot allocate memory
    strcpy(p, str);
    cout<<str<<endl;
    cout<<p<<endl;
    return 0;
}
```

Exception Due to Wrong Run-time Environment

```
#include <iostream>
#include <cstring>
#include <cstdlib>
using namespace std;
int main(){
	char str[100]="Long Live Bangladesh";
	char *p;
	p=new char[100000];
	if(p==NULL){
		cout<<“Memory Allocation Problem”<<endl;
		exit(1);
	}
	strcpy(p, str);
	cout<<str<<endl;
	cout<<p<<endl;
	return 0;
}
```

Exception Due to Wrong Run-time Environment

```
#include <iostream>
#include <cstring>
#include <cstdlib>
using namespace std;
int main(){
	char str[100]="Long Live Bangladesh";
	char *p;
	try {
		p=new char[100000];
		if(p==NULL) throw “Memory Allocation Problem”;
	}
	catch ( char *str){
		cout<<“Exception Occurs: “<<str<<endl;
	}
	strcpy(p, str);
	cout<<str<<endl;
	cout<<p<<endl;
	return 0;
}
```

Exception Due to Wrong Run-time Environment

```
#include <iostream>
#include <cstring>
#include <new>
using namespace std;
int main(){
    char str[100]="Long Live Bangladesh";
    char *p;
    try {
        p=new char[100000];
    }
    catch ( bad_alloc & ba){
        cout<<“Exception Occurs: “<<ba.what()<<endl;
    }
    strcpy(p, str);
    cout<<str<<endl;
    cout<<p<<endl;
    return 0;
}
```

Throwing Primitive Data as Exceptions

```
#include <iostream>
using namespace std;
int main(){
	int n;
	cout<<“Enter a number: “;
	cin>>n;
	try {
		if (n==0) throw ‘z’;
		else if(n%2==0) throw 2;
		else throw 3.14
	}
	catch (char c){
		cout<<“Zero Exception: ”<<c<<endl;
	}
	catch (int x){
		cout<<“Integer Exception: ”<<x<<endl;
	}
	catch (double d){
		cout<<“Double Exception: ”<<d<<endl;
	}
	return 0;
}
```

Catch All Exceptions by Ellipses

```
#include <iostream>
using namespace std;
int main(){
	int n;
	cout<<“Enter a number: “;
	cin>>n;
	try {
		if (n==0) throw ‘z’;
		else if(n%2==0) throw 2;
		else throw 3.14
	}
	catch (…){
		cout<<“Exception: ”<<endl;
	}
	return 0;
}
```

Throw User Defined Object as Exception

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

class myclass{
public:
    char* what() throw(){
        return "Divide By Zero Exception";
    }
};
```

Throw User Defined Object as Exception

```
int main () {
      int a, b;
      myclass ex;
      cout<<"Enter a and b: ";
      cin>>a>>b;
  try
  {
    if(b==0) throw ex;
      else cout<<"a/b:= "<<a/b<<endl;
  }
  catch (myclass& e)
  {
    cout << "exception caught: " << e.what() << endl;
  }
  return 0;
}
```

Standard Exception Class

```
#include <exception>

class exception {
    public:
    exception () throw();
    exception (const exception&) throw();
    exception& operator= (const exception&) throw();
    virtual ~exception() throw();
    virtual const char* what() const throw();
} ;
```

Standard Exception Class

- All objects thrown by components of the standard library are derived from this exception class
- Therefore, all standard exceptions can be caught by catching this type
- Some classes derived from exception class are:
  - bad_alloc
  - bad_cast
  - bad_exception
  - bad_typeid
  - logic_error
  - runtime_error

Standard Exception Class

```
#include <iostream>
#include <typeinfo>
using namespace std;
class Polymorphic {virtual void Member(){}};
int main () {
  try
  {
    Polymorphic * pb = 0;
    typeid(*pb);  // throws a bad_typeid exception
  }
  catch (exception& e)
  {
    cerr << "exception caught: " << e.what() << endl;
  }
  return 0;
}
```

Standard Exception Class

```
#include <iostream>
#include <typeinfo>
using namespace std;
class Base {virtual void Member(){}};
class Derived : Base {};
int main () {
    try { Base b;
          Derived& rd = dynamic_cast<Derived&>(b);
    }
    catch (bad_cast& bc) {
          cerr << "bad_cast caught: " << bc.what() << endl; }
    return 0;
}
```

Standard Exception Class

```
#include <iostream>
#include <typeinfo>
using namespace std;
class Base {virtual void Member(){}};
class Derived : Base {};
int main () {
    try {
        Base b;
        Derived& rd = dynamic_cast<Derived&>(b);
    }
    catch (bad_cast& bc) {
        cerr << "bad_cast caught: " << bc.what() << endl; }
    return 0;
}
```

User Defined Exception Class Derived from Standard Exception Class

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

class myexception : public exception{
public:
    const char* what() const throw(){
        return "Divide By Zero Exception";
    }
};
```

User Defined Exception Class Derived from Standard Exception Class

```
int main () {
      int a, b;
      myexception ex;
      cout<<"Enter a nd b: ";
      cin>>a>>b;
  try
  {
    if(b==0) throw ex;
      else cout<<"a/b:= "<<a/b<<endl;
  }
  catch (exception& e) {cout << "exception caught: " << e.what() << endl; }
  return 0;
}
```

cout<<“Thank You”<<endl;

cout<<“**Have a Good Day**”<<endl;