C++ slides - 8

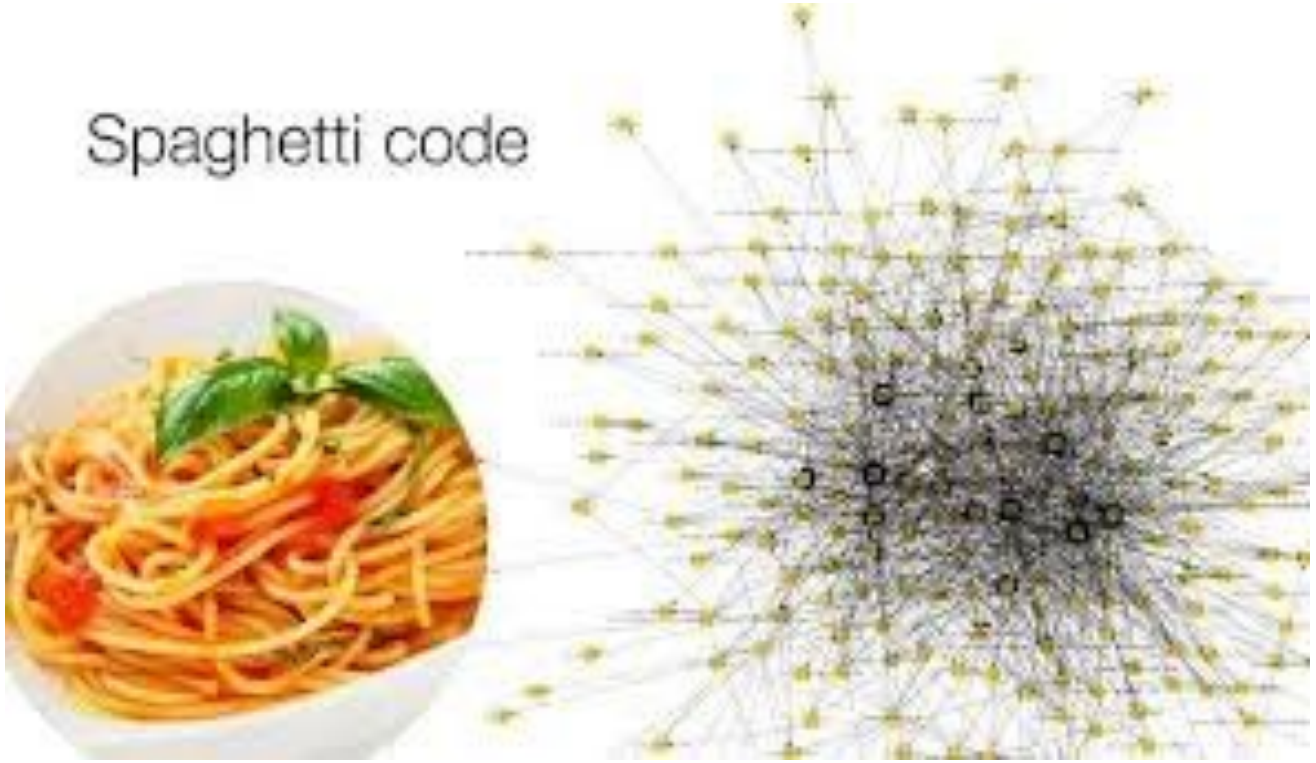
Exception Handling: Exception handling mechanism, Multiple Catch Blocks, Catch All exceptions, Throw an exception, Exception Specification.

Exception handling definition & motivation

- Exception is a run-time error, such as divide by zero, array out of bounds, file not found etc.
- It helps to continue the normal flow of the application even in the case of runtime errors
- It makes the code simpler, cleaner, and less likely to miss errors

if-else vs. exception handling

Only using *errno* & *if*-statements can make the error handling intertwine with normal code to result a spaghetti



Try-catch organizes the code

```
try{
fun1();
fun2();
fun3();
} catch(error){
// … do something if error happens in any of the functions
}
```

// if any of the fun() is a constructor then you have to use *if-else* in the constructor to handle the error since it cannot return anything, which is not neat practice.

Exception thrown and caught

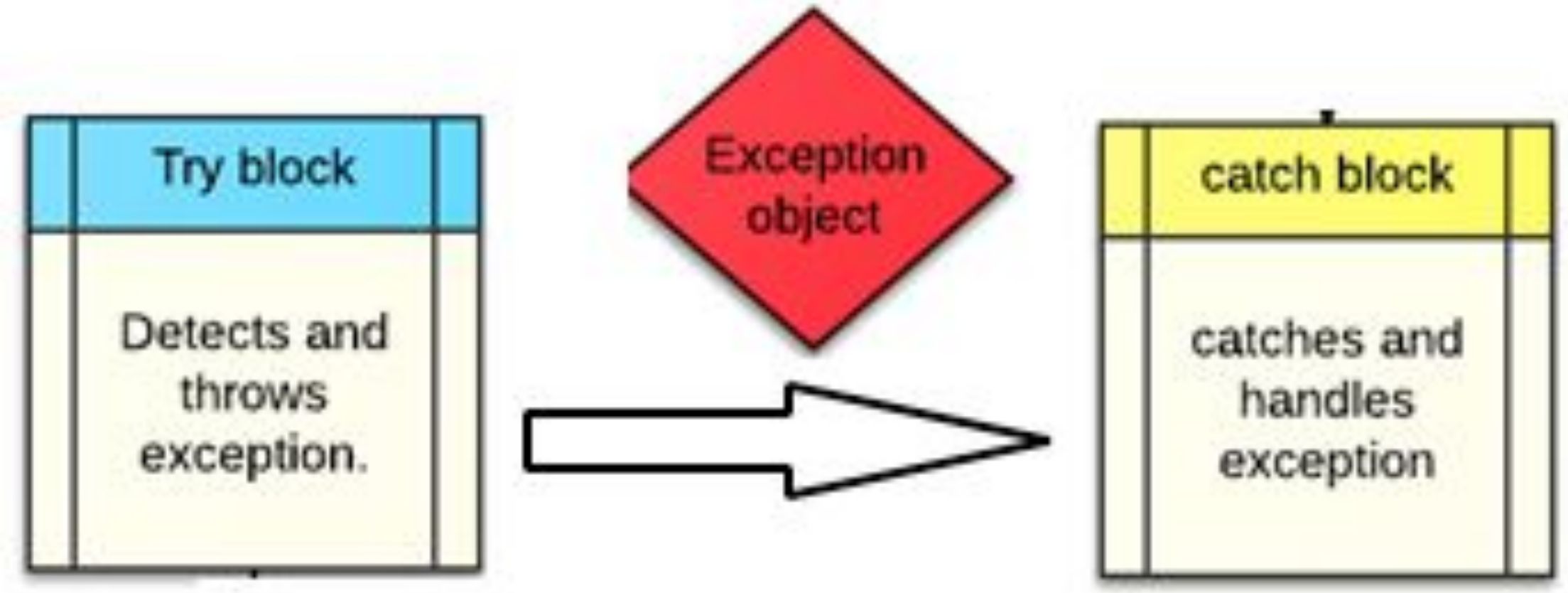
- Exception is thrown at runtime.
- If an exception is not handled (caught), it prints exception message and terminates the program.
- Programmer's job is to handle possible exceptions using *try*, *throw* and *catch* to avoid abnormal program termination


Running
Exception
Exception Handled

Syntax of exception handling

```
try {
//some code
throw exception;
}
catch(type arg) {
 //some code
}
```

Exception handling mechanism



```
// Without exception handling

float division(int x, int y){
return (x/y);
}
int main(){
int i=50, j=0;
float k=0;
k=division(i,j);
cout<<k<<endl;
}
```

Output

- Floating point exception (core dumped)

// With try, throw & catch

```
float division(int x, int y) {
   if( y == 0 ) throw “Custom msg - divide by zero!";
   return (x/y);
}
int main () {
   int i = 25,j = 0;
   float k = 0;
   try {
      k = division(i, j);
      cout << k << endl;
   } catch (const char* e) { cout<< e << endl;}
}
```

Output

Custom msg - divide by zero!

// What will be the output?

```
float division(int x, int y) {
   if( y == 0 ) throw 5;
   return (x/y);
}
int main () {
   float k = 0;
   try {
      k = division(25, k);
      cout << k << endl;
   }
   catch (const char* e) { cout<< e << endl;}
   catch(int i) {cout<<"caught i = "<<i<<endl;}
}
```

Output

caught i = 5

Multiple catch is possible

```
try {
//some code
throw exception;
}
catch(specific type exception) {//some code }
…
catch(generic type exception) { //some code }
```

Catch blocks must be ***ordered*** so that most specific exceptions get caught before generic exceptions.

// Catch all exceptions – catch(…)

```
int main() {
    try  {
       throw 10;
    }
    catch (char *e)  { cout << "Caught " << e; }
    catch (...)  { cout << "Default Exception\n"; }
    return 0;
}
```

Output

Default Exception

// What will be the output?

```
int main() {
    try  {
       throw "10";
    }
    catch (const char *e)  { cout << "Caught " << e; }
    catch (...)  { cout << "Default Exception\n"; }

 }
```

Output

Caught 10

Exception Specification

- C++ provides a mechanism to ensure that a given function is limited to throw only a specified list of exceptions.

```
void translate() throw(unknown_word,bad_grammar) {
/* ... */
}
```

- It will not throw any exception other than *unknown _word* or *bad_grammar*

```
class unknown_word{};
class bad_grammar{};
void translate(int a) throw(unknown_word,bad_grammar) {
if (a==0)   throw unknown_word();
else if(a==1) throw bad_grammar();
else throw 10;
}
int main(){
try{ translate(22);}
 catch(unknown_word u){cout<<" unknown word";}
 catch(bad_grammar g){cout<<" bad grammar";}
 catch(...) {cout<<"Default catch";}
}
```

Output

terminate called after throwing an instance of 'int'

Aborted (core dumped)