03. A Tour of C++: Flow

Data Structure and Algorithms

Hanjun Kim

Compiler **O**ptimization **Re**search **Lab**

Yonsei University

Last time: Types

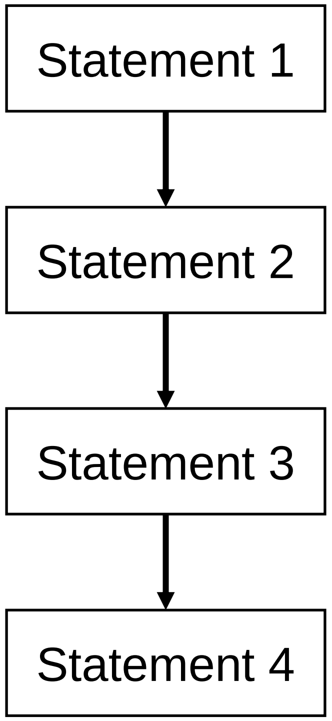
- Hello World
  - `main` function
  - `std::cout` and `std::cin`
  - namespace
- Types
  - Built-in Types
    - Boolean, Character, Integer, Floating-point Number
  - Programmer-defined Types
  - Standard library-provided Types
    - String, vector, complex
  - auto: the type of the initializer (C++14)
- Constants

Today: Flow

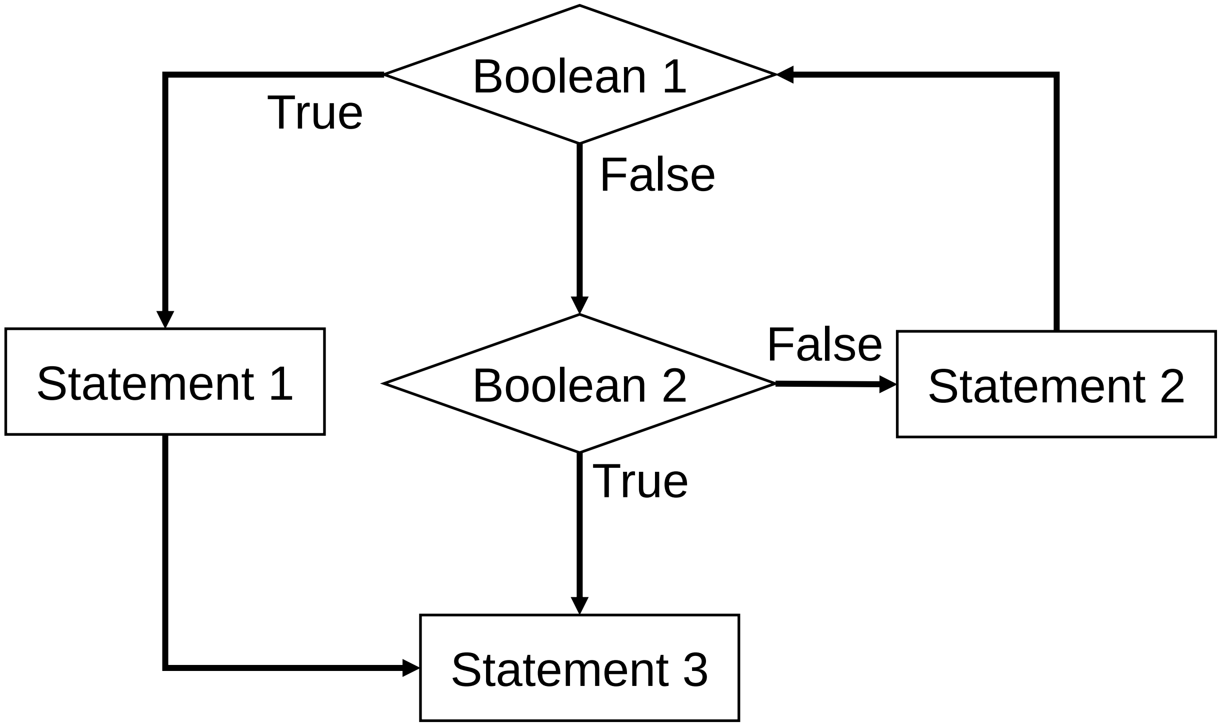
- Conditionals
  - IF statement
  - Switch-case statement
- Loops
  - While loop
  - Do While loop
  - For loop
- Nesting conditionals and loops
- Error Handling
  - Error
  - Assertion
  - Exception

Control Flow

- Sequence of statements that are actually executed in a program



straight-line control flow



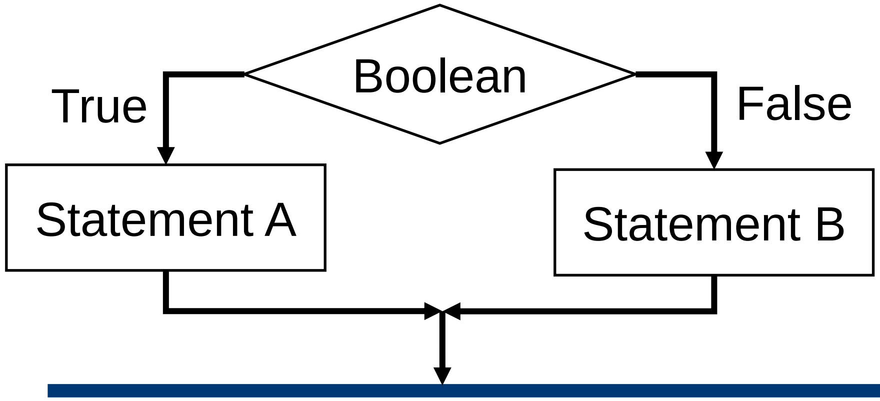
control flow with conditionals and loops

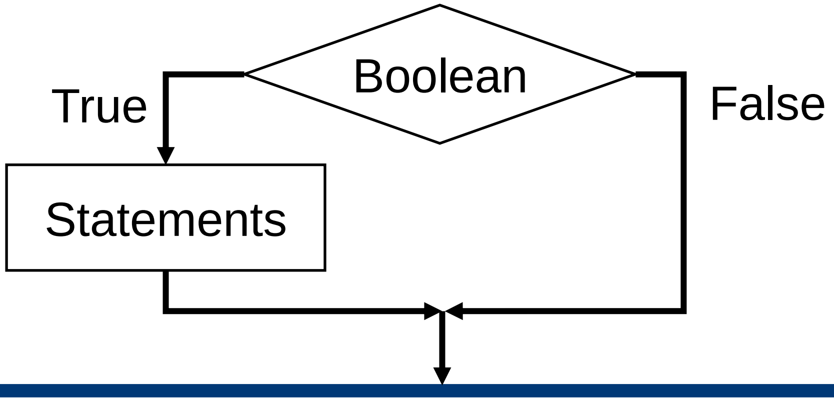
Conditionals: if statement

- The if statement
  - A common branching structure
  - Evaluate a boolean expression
    - If true, execute some statements
    - If false, execute other statements (Else option)

```
if(boolean expression){
  statement A // if true
} else {
  statement B // if false
}
```

```
if(boolean expression){
  statements // if true
}
```





Conditionals: if statement

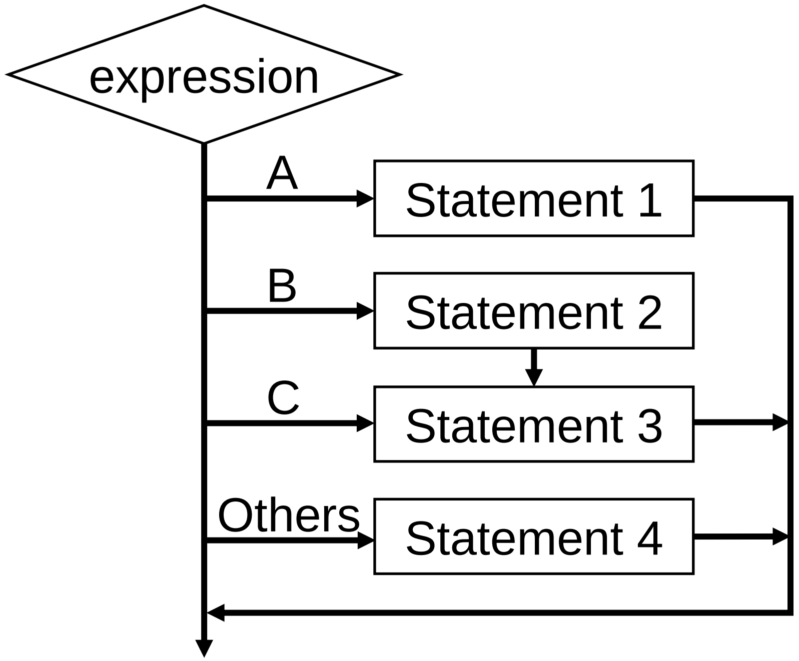
```
#include<iostream>
#include<cstdlib>
#include<ctime>
using namespace std;

int main (){
  srand(time(NULL));
  if((rand( ) % 2) == 1) cout << "Heads! \n";
  else cout << "Tails! \n";
}                                                    flip.cpp
```

- The if statement
  - If `rand()` returns an odd number, print Heads
  - If `rand()` returns an even number, print Heads
- `srand(time(NULL));`
  - Initialize the random number generator using the argument passed as seed (`time(NULL)`)

Conditionals: switch statement

- The switch statement
  - Tests a value against a set of constants
  - Evaluates switch expression
    - If it matches one of cases, executes some statements
    - If it matches none, executes default statements or does nothing
  - Without `break`, executes following case statements
  - Case constants must be distinct

```
switch(expression){
  case A: statement 1;
          break;
  case B: statement 2;
  case C: statement 3;
          break;
  default: statement 4;
}
```



Conditionals: switch statement

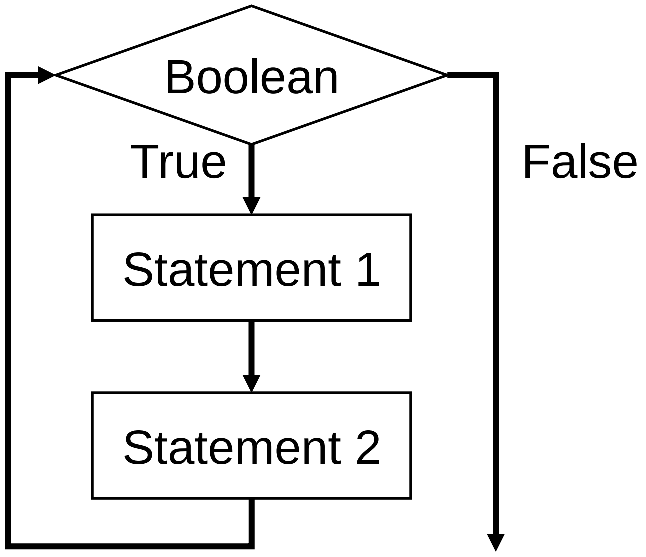
```
bool accept(){
  cout << "Do you want to proceed (y or n)?\n";
  char answer = 0;
  cin >> answer; // read answer
  switch (answer) {
    case 'y':  return true;
    case 'n':  return false;
    default:  cout << "I'll take that for a no.\n";
              return false;
  }
}
```

- Function accept
  - Ask a question and execute a switch statement
  - If user types `y`, returns true
  - If user types `n`, returns false
  - If user types the others, prints a message and returns false

Loops: while loop

- The while loop
  - A common repetition structure
  - Operation
    - Check a boolean expression
    - Execute a sequence of statements
    - Repeat

```
while(boolean expression){
  // loop body
  statement1;
  statement2;
}
```



Loops: while loop example

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

int main (){
  int n; // last power of two to print
  cin >> n;
  int i = 0;  // loop control counter
  int v = 1;  // current power of two
  while (i <= n) {
    cout << v << endl;
    i = i + 1;
    v = 2 * v;
  }
}                                          power.cpp
```

- Calculate powers of 2 that are less than 2^n
  - Increase `i` from `0` to `n`
  - Double `v` each time

Loops: while loop example

- Implement Square Root
  - `sqrt(n)` in `cmath` library
  - *Newton-Raphson method* to compute the square root of n
    - Initialize $t_0 = n$
    - Repeat until $t_i = n / t_i$, up to desired precision
    - set t_{i+1} to be the average of t_i and n / t_i
  - Example: `sqrt(2)`

i	t_i	$2/t_i$	t_{i+1}(Average)
0	2.0	1.0	1.5
1	1.5	1.3333333	1.4166667
2	1.4166667	1.4117647	1.4142157
3	1.4142157	1.4142114	1.4142136
4	1.4142136	1.4142136	1.4142136

Loops: while loop example

- Implement Square Root
  - *Newton-Raphson method* to compute the square root of n

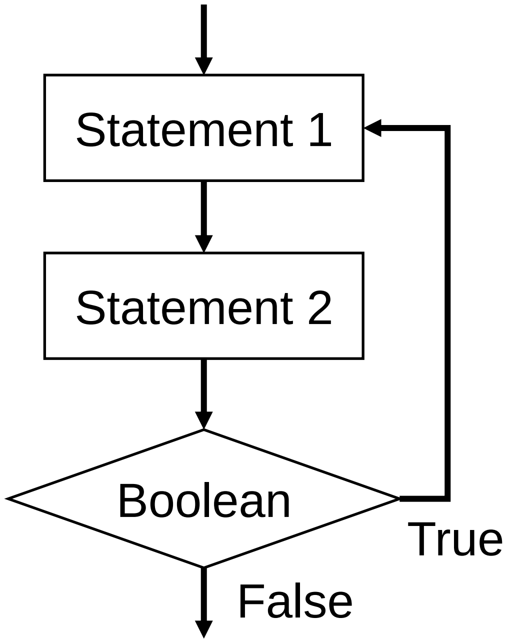
```
#include<iostream>
#include<cmath>
using namespace std;

int main (){
  double error = 1E-5;
  double n, t;
  cin >> n;
  t = n;
  while (abs(t - n/t) > error) {
    t = (t+n/t)/2.0;
  }
  cout << "Sqrt of " << n << " is " << t << "!\n";
}
```

sqrt.cpp

Loops: do-while loop

- The do-while loop
  - Another common repetition structure
  - Operation
    - Execute a sequence of statements once
    - Check a boolean expression
    - Execute a sequence of statements
    - Repeat

```
do {
  // loop body
  statement1;
  statement2;
} while(boolean expression)
```



Loops: do-while loop

- *Newton-Raphson method*

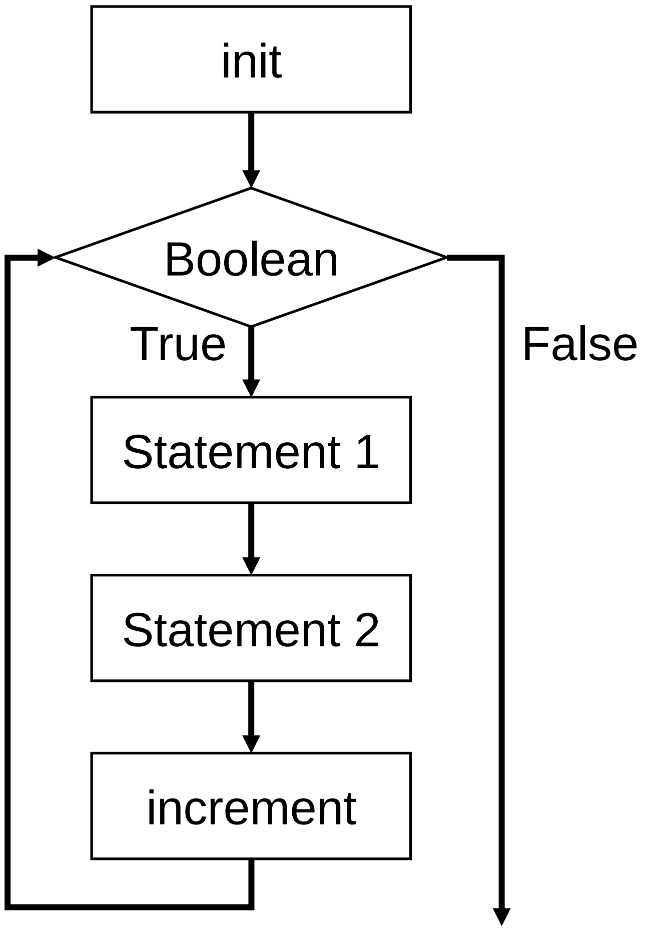
```
#include<iostream>
#include<cmath>
using namespace std;

int main (){
  double error = 1E-5;
  double n, t, t2;
  cin >> n;
  t = n;
  t2 = n;
  do {
    t = (t + t2)/2.0;
    t2 = n/t;
  } while (abs(t - t2) > error);
  cout << "Sqrt of " << n << " is " << t << "!\n";
}                                                    sqrt2.cpp
```

Loops: for loop

- The for loop
  - Another common repetition structure
  - Operation
    - Execute initialization statement
    - Check boolean expression
    - Execute sequence of statements
    - Execute increment statement
    - Repeat

loop continuation condition

```
for(init; boolean exp; increment){
  // loop body
  statement1;
  statement2;
}
```



Loops: for loop example

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

int main (){
  int n, i, v; // n: last power of two to print
  // i: loop control counter, v: current power of two
  cin >> n;
  for (i = 0, v = 1; i <= n; i = i + 1) {
    cout << v << endl;
    v = 2 * v;
  }
}                                                   power2.cpp
```

- Calculate powers of 2 that are less than 2^n
  - for loop is used instead of the while loop in power.cpp
  - `i` and `v` are initialized in the for loop (init)
  - `i` is increased in the for loop (increment)

Loops: for loop example

- What does the following program print?

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

int main (){
  int f = 0, g = 1;
  for (int i = 0; i <= 10; i++) {
    cout << f << endl;
    f = f + g;
    g = f - g;
  }
}
                                              for_quiz.cpp
```

Nesting Conditionals and Loops

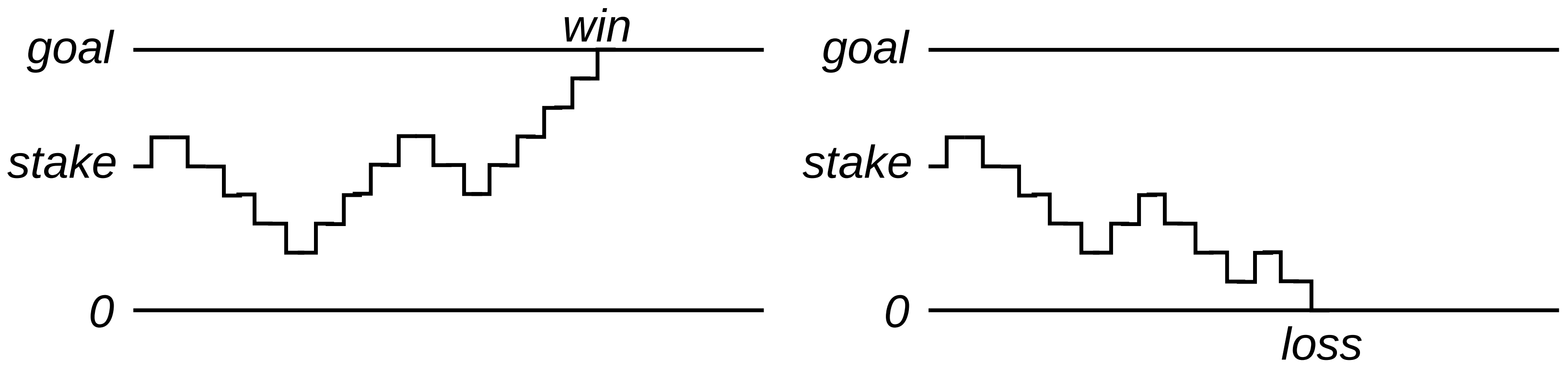
- Nesting:
  - Use a conditional or a loop within a conditional or a loop
  - Enables complex control flows

```
for (int i = 0; i < trials; i++){
  int t = stake;
  while (t > 0 && t < goal){
    if (rand() %2 == 1) t++;
    else t--;
  }
  if (t == goal) wins++;
}
```

Nesting Example: Gambler's ruin

- Gambler's ruin
  - Gambler starts with $*stake* and places $*1* fair bets until going broke or reaching $*goal*
  - What are the chances of winning?
  - How many bets will it take?

- One approach: Monte Carlo simulation
  - Flip digital coins and see what happens
  - Repeat and compute statistics



Nesting Example: Gambler's ruin

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

int main(){
  int stake, goal, trials, wins = 0;
  cout << "Put stake, goal and trials\n";
  cin >> stake >> goal >> trials; // Get parameters

  srand(time(NULL));
  for (int i = 0; i < trials; i++) { // Count wins among trials.
    // Do one gambler's ruin experiment.
    int t = stake;
    while (t > 0 && t < goal)
      if (rand() % 2 == 1) t++; // flip coin and update
      else t--;
    if (t == goal) wins++;
  }
  cout << wins << " wins of " << trials << endl;
}
```

gamblers_ruin.cpp

Nesting Example: Gambler's ruin

- Gambler's ruin
  - Fact: Probability of winning = stake ÷ goal
  - Ex. 20% chance of turning $500 into $2500, 500/2500 = 20%

- The fact can be proved mathematically.
- For more complex scenarios, computer simulation is often the best plan of attack

```
% ./gamblers_ruin.exe
Put stake, goal and trials
500 2500 1000
191 wins of 1000

% ./gamblers_ruin.exe
Put stake, goal and trials
500 2500 1000
184 wins of 1000
```

Error Handling: Error

- Programming is a process of finding and fixing mistakes
  - Syntax error: Illegal program
    - Compiler error messages help locate syntax errors
  - Semantic error: Legal but wrong program
    - Run program to identify problem
    - Add print statements if needed to produce trace.
  - Performance error: Correct but inefficient program
    - Run program to find the performance bottleneck
    - Tune the performance

```
while (i <= n) {
  i = i + 1;
  v = v + v
}
```

; is missing

Syntax error

{ is missing

```
while (i <= n)
  i = i + 1;
  v = v + v;
```

} is missing

Semantic error

```
while (i <= n) {
  i = i + 1;
  v = v * 2;
}
```

* is slower than +

Performance error

Error Handling: Assertion

- Assertion
  - A logical expression that is assumed to be true
  - Expresses what happens if it is false

```
Assertion failed: expression, file filename, line line number
```

  - Disabled if a macro with the name NDEBUG has already been defined

```
#ifdef NDEBUG
#define assert(condition) ((void)0)
#else
#define assert(condition) /*implementation defined*/
#endif
```

Error Handling: Assertion

- Assertion Example
  - Check if 2+2 is 4 and 2+2 is 5
  - Program aborts at the second assert

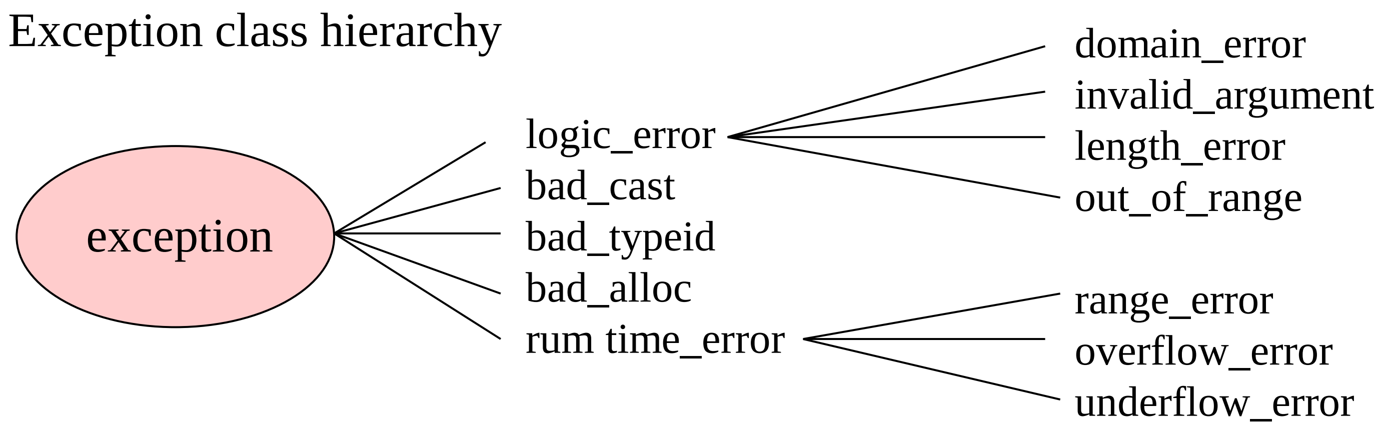
```
#include <iostream>
// uncomment to disable assert()
// #define NDEBUG
#include <cassert>

int main(){
    assert(2+2==4);
    std::cout << "Execution continues past the first assert\n";
    assert(2+2==5);
    std::cout << "Execution continues past the second assert\n";
}
```

assert.cpp

```
Execution continues past the first assert
Assertion failed: (2+2==5), function main, file assert.cpp, line 9.
Abort trap: 6
```

Error Handling: Exception

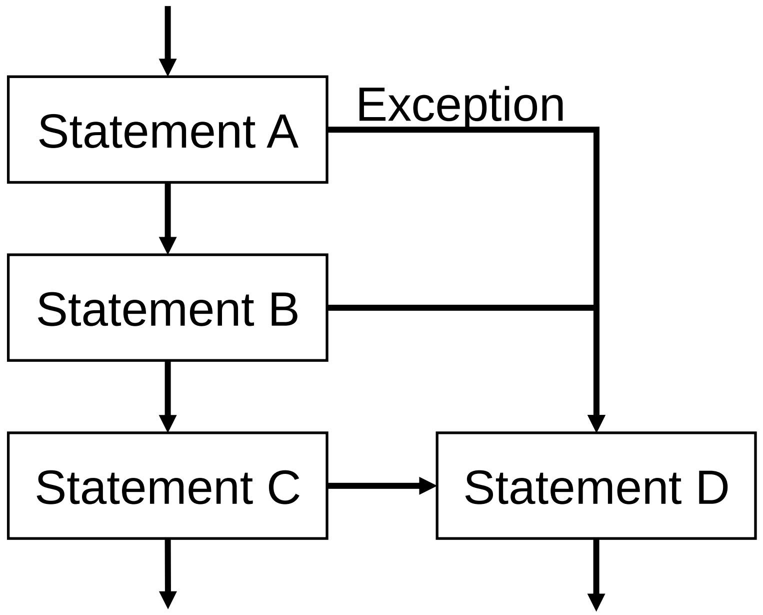
- Exception
  - A way to react to exceptional circumstances
  - Transfers control to special functions called handlers
  - Exceptional circumstances
    - underflow, overflow, division by zero, ...
- Exception classes



Error Handling: Exception

- Exception Handling: `try-catch` blocks
  - `try` block
    - A portion of code is placed under exception inspection
  - `catch` block
    - Exception handler that manages the exceptional circumstances
    - Placed immediately after the try block
  - throw expression
    - Represents the occurrence of an error

```
try {
   Statement A;
   Statement B;
   Statement C;
} catch (...) {
   Statement D;
}
```



Error Handling: Exception

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

double division(int a, int b) {
   if( b == 0 ) throw "Division by zero condition!";
   return (a/b);
}

int main () {
   int x = 50;
   int y = 0;
   double z = 0;

   try {
      z = division(x, y);
      cout << z << endl;
   } catch (const char* msg) {
     cerr << msg << endl;
   }
   return 0;
}                                                         exception.cpp
```

Summary: Flow

- Conditionals
  - IF statement
  - Switch-case statement
- Loops
  - While loop
  - Do While loop
  - For loop
- Nesting conditionals and loops
- Error Handling
  - Error
  - Assertion
  - Exception