Unit 7

'while' Loops

Control Structures

- We need ways of making **decisions** in our program
  - To repeat code until we want it to stop
  - To only execute certain code if a condition is true
  - To execute one segment of code or another
- Language constructs that allow us to make decisions are referred to as **control structures**
- The common ones are:
  - if statements
  - switch statements
  - while loops
  - for loops

Loops

- Loops are structures of code that may be repeated some number of times
- Examples:
  - Sum each student's grades (for all students in the class)
  - Search through a sequence of numbers for a particular value
  - Attend lecture ☺
- We need some condition to tell us when to stop looping, otherwise we'll repeat our code forever and never stop (a.k.a. an infinite loop)
- Several kinds of loops: 'while', 'do..while', and 'for'
  - We will focus on 'while' and 'do..while' in this unit

Why We Need Loops (1)

- Suppose we are writing a program for a simple turn-based guessing game where the user must guess a secret number
- If they guess incorrectly what should we do?

```
#include <iostream>
using namespace std;
int main()
{
  int guess;
  int secretNum = /* some code */
  cin >> guess;
  if(guess != secretNum) {
    /* What should we do here? */


  }
  else {
    cout << "You got it!" << endl;
  }
  return 0;
}
```

Why We Need Loops (2)

- What if they guess wrong a second time? What should we do?

```
#include <iostream>
using namespace std;
int main()
{
  int guess;
  int secretNum = /* some code */
  cin >> guess;
  if(guess != secretNum) {
    cin >> guess;
    if(guess != secretNum) {
       /* What should we do here? */
    }
    else {
       cout << "You got it!" << endl;
    }
  }
  else {
    cout << "You got it!" << endl;
  }
  return 0;
}
```

Why We Need Loops (2)

- We can never write enough `if` statements because someone might always use one more turn than we have `if` statements
- But we see there is a repetitive structure in this code
- Let's use a loop

```
#include <iostream>
using namespace std;
int main()
{
  int guess;
  int secretNum = /* some code */
  cin >> guess;
  if(guess != secretNum) {
    cin >> guess;
    if(guess != secretNum) {
      cin >> guess;
      if(guess != secretNum) {
       /* What should we do here? */
      }
      else {
        cout << "You got it!"<< endl;
      }
    }
    else {
       cout << "You got it!" << endl;
    }
  }
  else {
    cout << "You got it!" << endl;
  }
```

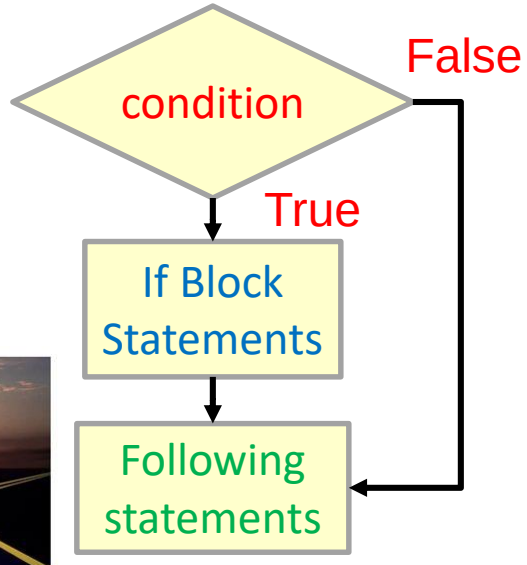
while Loops

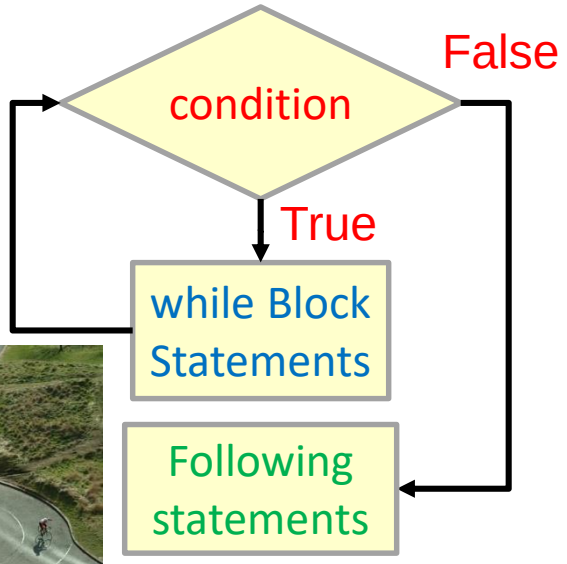
- A while loop is essentially a repeating 'if' statement

```
if (condition)
{
  // executed if condition1 is true
}
// following statements
```

```
while (condition)
{
  // executed if condition1 is true
} // go to top, eval cond1 again

// following statements
// only gets here when cond1 is false
```





while Loops

- A while loop is essentially a repeating 'if' statement

Condition: T T F

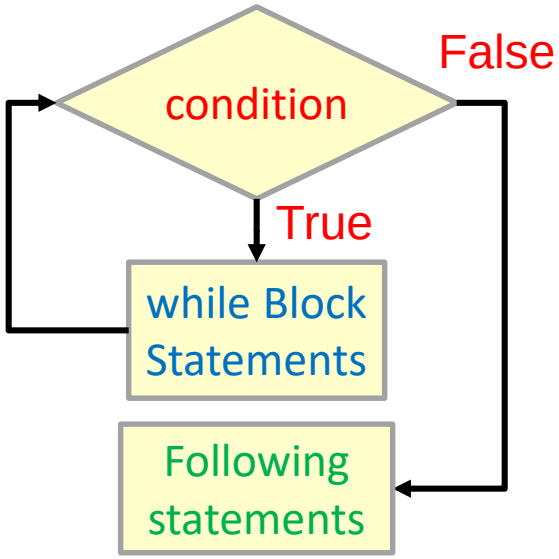
1 3 5

2 4

6

```
while (condition)
{
  // executed if condition1 is true
} // go to top, eval cond1 again

// following statements
// only gets here when cond1 is false
```



When Do I Use a While Loop (1)

- When you don't know in advance how many times something should repeat?
  - How many guesses will the user need before they get it right?

```
#include <iostream>
using namespace std;
int main()
{
  int guess;
  int secretNum = /* some code */
  cin >> guess;
  while(guess != secretNum) {
    cout << "Enter guess: " << endl;
    cin >> guess;
  }
  cout << "You got it!" << endl;
  return 0;
}
```

When Do I Use a While Loop (2)

- Whenever you see or use the word 'until' in a description
- Important Tip:
  - "until" = "while not"
  - Saying "keep guessing until you are correct" is the same as "keep guessing while you are not correct"

```
#include <iostream>
using namespace std;
int main()
{
  int guess;
  int secretNum = /* some code */
  cin >> guess;
  while(guess != secretNum) {
    cout << "Enter guess: " << endl;
    cin >> guess;
  }
  cout << "You got it!" << endl;
  return 0;
}
```

What Goes In an `while` Block

- What do we put in an `while` loop?
- ANYTHING!
  - Expressions & variable assignment
  - Function calls
  - Even other if..else statements

```
#include <iostream>
using namespace std;
int main()
{
  int guess;
  int secretNum = /* some code */
  cin >> guess;
  while(guess != secretNum) {
    cout << "Enter guess: " << endl;
    cin >> guess;
  }
  cout << "You got it!" << endl;
  return 0;
}
```

What Goes In an `while` Condition

- What do we put in a `while` condition?
- ANYTHING.
  - The compiler will interpret what is in the parentheses as a Boolean
    - 0 = false
    - Non-0 = true

```
int main()
{
  int x, y, val;
  bool done;

  // Uses Boolean result of comparison
  while( x > 0 )    { /* code */ }

  // Uses value of bool variable.
  //  Executes if done == false.
  while( !done )     { /* code */ }

  // Interprets number as a bool
  //  Executes if val is non-zero
  while( val )      { /* code */ }

  // Interprets return value as bool
  //  Executes if the min is non-zero
  while( min(x,y) ) { /* code */ }

  return 0;
}
```

Hand Tracing (1)

- Trace through the code and show all changes to x and y for:
  - x = 24
  - y = 18

```
int main()
{
  int x, y;
  cin >> x;
  while( (x % 2) == 0){
    x = x/2;
  }

  cin >> y;
  while(y > 0){
    if( y >= 10 ){
      y -= 5;
    }
    else if( y >= 5 ){
      y -= 3;
    }
    else {
      y -= 1;
    }
  }
  return 0;
}
```

Hand Tracing (2)

- Trace through the code and show all changes to x and y for:
  - x = 27
  - y = 6

```
int main()
{
  int x, y;
  cin >> x;
  while( (x % 2) == 0){
    x = x/2;
  }

  cin >> y;
  while(y > 0){
    if( y >= 10 ){
      y -= 5;
    }
    else if( y >= 5 ){
      y -= 3;
    }
    else {
      y -= 1;
    }
  }
  return 0;
}
```

Exercises 1

- cpp/while/whilen
- cpp/while/sum50
- cpp/while/blastoff

do..while Loops (1)

- while loops have a sibling known as do..while loops
- do..while loops
  - Start with keyword do
  - Followed by the body of code to be executed repeatedly in brackets { }
  - Ends with `while` condition and semicolon (;)
- do..while loops will execute the body at least once

```
int main()
{
  int x, y, val;
  bool quit;

  // a while loop
  while( x < val )
  {
    /* body of code */
  }

  // a do..while loop
  do
  {
    /* body of code */
  } while( x < val );

  return 0;
}
```

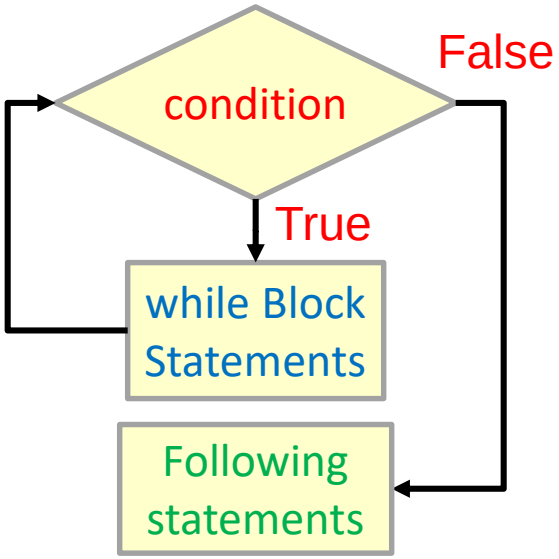
do..while Loops (2)

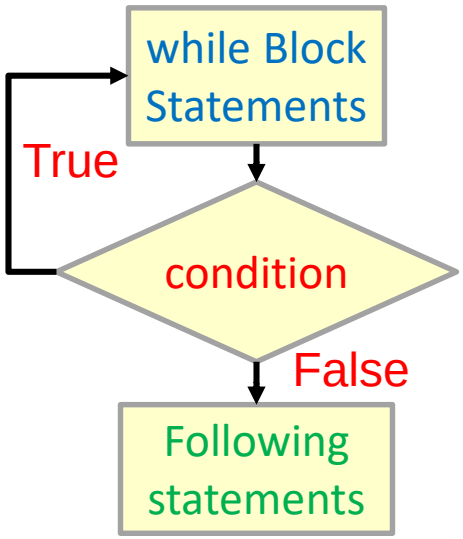
- do..while loops check the condition after executing at least once and repeat if the condition is true

```
while (condition)
{
  // executed if condition1 is true
} // go to top, eval cond1 again

// following statements
// only gets here when cond1 is false
```

```
do
{
  // executed at least once
} while (condition);// go to 'do' (top)
                    //if cond1 evals to true
// following statements
// only gets here when cond1 is false
```



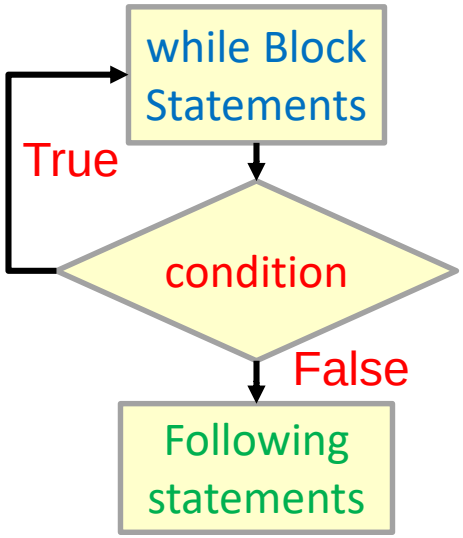


do..while Loops (3)

- do..while loops check the condition after executing at least once and repeat if the condition is true

Condition: 1 3 5 T
2 4 6 T
7 F

```
do
{
  // executed at least once
} while (condition);// go to 'do' (top)
                    //if cond1 evals to true
// following statements
// only gets here when cond1 is false
```

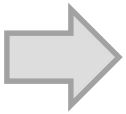


When Should I Use do..while

- We generally prefer while loops
- We can use do..while loops when we know we want to execute the code at least one time (and then check at the end)
- Even then…
  - See next slide

Converting do..while to while Loops

```
do
{
  cin >> guess;
} while (guess != secretnum);
cout << "You got it!" << endl;
```



```
cin >> guess;
while (guess != secretnum)
{
  cin >> guess;
} // go to top, eval cond1 again
cout << "You got it!" << endl;
```

We need to get one guess at least and then determine if we should repeat. This seems a natural fit for the do..while structure but we can easily mimic this behavior with a normal while loop.

We can duplicate the body of the loop once before we start the loop.

```
guess = secretnum + 1;
while (guess != secretnum)
{
  cin >> guess;
} // go to top, eval cond1 again
cout << "You got it!" << endl;
```

We can set our variables to ensure the while condition is true the first time.

Exercises 2

- cpp/while/dowhilen
- cpp/while/goldilocks

Common Loop Mistakes

- Failing to update the variables that affect the condition
- Assignment rather than equality check
- Off by one error
- Often leads to **infinite loops**
  - When you run your program it will not stop
  - **Use Ctrl+c to force quit it**

```
int i=0, n=10;
while (i < n)
{
  cout << "Iteration " << i << endl;
  // Oops forgot to change i
}
cout << "Done" << endl;
```

```
int i=0, n=5;
while (i = n) // oops, meant i==n
{
  cin >> i;
}
cout << "Done" << endl;
```

```
int i=0;
// want to print "Hi" 5 times
while (i <= 5) // oops, meant i < n
{
  cout << "Hi" << endl;
  i++;
}
```

Flags: A Common while Structure

- A Boolean flag
  - Two values: true or false
  - Pattern: Initialize to a value that will cause the while loop to be true the first time and then check for the ending condition in an if statement and update the flag
  - Up to you to determine the meaning of the flag (e.g. done or again)

```
int guess, secretNum;
bool done = false;
while ( ! done )
{
  cin >> guess;
  if(guess == secretNum) {
    done = true;
  }
}
cout << "You got it!" << endl;
```

```
int guess, secretNum;
bool again = true;
while ( again )
{
  cin >> guess;
  if(guess == secretNum) {
    again = false;
  }
}
cout << "You got it!" << endl;
```