An Introduction to Programming

through C++

with

Manoj Prabhakaran

Lecture 2

Conditions

To do or not to do - that is the question

Based on material developed by Prof. Abhiram G. Ranade

So far

- Sequential execution of statements
- Loops and nested loops: `repeat(n) {` *body* `}`
- Variables, int type, arithmetic expressions (e.g., 360.0/n)
- Input/output (cout, cin)
- Comments and indentation

Today

- Conditional execution (execute only if some condition holds)
- More types. Meet "Boolean" variables!
- Expressions.
- Assignment.

Refer to Chapter 6 in the textbook

Adding a Condition

```
#include <simplecpp>
main_program {
  turtleSim();
   cout << “How many sides? ”;
   int nsides;
   cin >> nsides;

    repeat(nsides){
        forward(400/nsides);
        right(360.0/nsides);
    }

   getClick();
}
```

Adding a Condition

```
#include <simplecpp>
main_program {
  turtleSim();
   cout << “How many sides? (No more than 400 please!) ”;
   int nsides;
   cin >> nsides;
   // can we draw only if nsides ≤ 400?
    repeat(nsides){
        forward(400/nsides);
        right(360.0/nsides);
    }

   getClick();
}
```

Adding a Condition

```
#include <simplecpp>
main_program {
  turtleSim();
   cout << “How many sides? (No more than 400 please!) ”;
   int nsides;
   cin >> nsides;
   if (nsides <= 400) {   // draw if nsides ≤ 400
      repeat(nsides){
          forward(400/nsides);
          right(360.0/nsides);
      }
   }
   getClick();
}
```

If Statement

- Syntax:

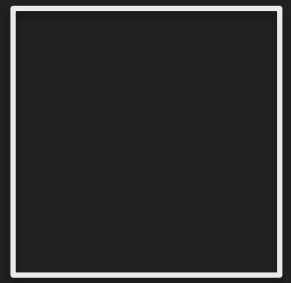
```
if (condition) { body }
```

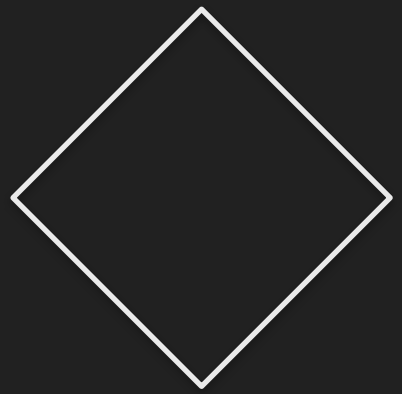
- Here, condition is an <u>expression</u> that takes value "true" or "false"
- Semantics: Body should be executed if condition is true
- Some examples of conditions
  - x <= y , x < y , x >= y , x > y , x == y , x != y

Note: It is == (rather than =)

Another Example

- Draw a square or a "diamond" (rotated square) based on what the user asks





Another Example

character type

```
cout << "Enter s for square, d for diamond: " ;
char input; cin >> input;
if (input == 'd')
  right(45);
repeat(4){ // draw a square in the direction turtle is facing
    forward(100);
    right(90);
}
```

character d, different from "d" (string d)

If the body is a single statement,
{ and } can be omitted **(use with care!)**

What happens if the user enters, say, x?

Another Example

```
cout << "Enter s for square, d for diamond: " ;
char input; cin >> input;
if (input == 'd')
   right(45);
if (input == 'd' || input == 's')
   repeat(4){ // draw a square in the direction turtle is facing
        forward(100);
        right(90);
   }
if (input != 'd' && input != 's')
    cout << “Invalid input. Exiting." << endl ;
```

condition 1 OR condition 2

This is a single statement!

condition 1 AND condition 2

End the line

Another Example

```
cout << “Enter s for square, d for diamond: " ;
char input; cin >> input;
if (input == 'd')
  right(45);
if (input == 'd' || input == 's')
  repeat(4){ // draw a square in the direction turtle is facing
      forward(100);
      right(90);
  }
else
   cout << “Invalid input. Exiting." << endl ;
```

condition 1 OR condition 2

This is a single statement!

If-Else Statement

- Syntax:

```
if (condition) { body 1 } else { body 2 }
```

- This could be thought of as a more efficient version of

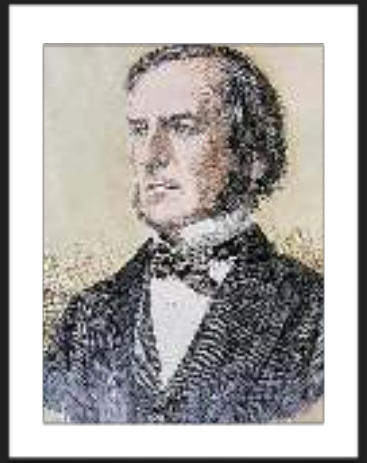
```
if (condition)  { body 1 }
if (!condition) { body 2 }
```

  - But then the condition is evaluated twice and may have "side-effects" (later)

- The condition can be an "expression" like

(*condition1* || *condition2*)

Boolean Variables and Expressions



- A British mathematician George Boole proposed a system of logical variables in 1847, which has been widely used in logic and programming: Boolean algebra
- Like integer variables take values ..., -2, -1, 0, 1, 2, ... boolean variables take values `true`, `false`
- Like there are operations +, * for integers, boolean variables allow operations like AND, OR, XOR, NOT
- C++ has a type `bool` for boolean variables

Boolean Variables and Expressions

```
bool x,y
x && y
x || y
x != y
!x
```

- x AND y is true if and only if both x and y are true
- x OR y is true if and only if at least one of x and y is true
- x XOR y is true if and only if exactly one of x and y is true
- NOT x is true if and only if x is false
- Can use boolean variables to store the truth value of conditions and use them later

```
bool valid;
valid = (input == 's' || input == 'd');
...
if (!valid)
   cout << “Invalid input!" << endl ;
```

Assignment (storing)

Assignment Expression

- Syntax: *variable* = *expression*
- Typically *variable* and *expression* must have the same *type*
- Example (for int): `int x; cin >> x;` `x` = `x*x - 1`;
- Semantics: Evaluate *expression* and then store the result in *variable*
- A quirk: Assignment itself is an expression. `x=(y=z)` is valid!
- The simplest kind of expressions: a constant (e.g., 360) or a variable by itself. Complex expressions are built using operators.
  - Integer operators: E.g., `+` `-` `*` `/` `%` (more on them later)

Assignment Expression

- What does the following code do?

```
main_program {
    turtleSim();
    int x=5; // x is assigned an initial value
    repeat(100) {
        forward(x); right(90);
        x=x+5; // add 5 to x, and store it back in x
    }
    hide();
    getClick();
}
```

Boolean Expressions in C++

- To evaluate the expression `condition1 || condition2` first condition1 is evaluated
  - <u>if `condition1` is true</u>, the entire expression is taken to be true, <u>without evaluating `condition2`</u>
  - otherwise, `condition2` is evaluated and its value becomes the value of the expression
- Similarly, while evaluating `condition1 && condition2`, if `condition1` is false, the entire expression is taken to be false, <u>without evaluating `condition2`</u>

Ternary Conditional Operator

- What does the following piece of code do?

```
int x, y, z;
cin >> x; cin >> y;
if (x > y) {
  z = x;
}
else {
  z = y;
}
```

More succinct equivalent version

```
int x, y, z;
cin >> x; cin >> y;
z = ( x>y ? x : y ) ;
```

condition ? *expr1* : *expr2*

Chaining if-else

```
int x, d;
bool found=false;
cin >> x;
if (x == 1 || x == -1) {
  cout << x << " has no prime fac
} else if (x%2 == 0) {
  d = 2; found = true;
} else if (x%3 == 0) {
  d = 3; found = true;
} else if (x%5 == 0) {
  d = 5; found = true;
} else
  cout << "Smallest prime factor of " << x << " is more than 5." << endl;
if (found) cout << "Smallest prime factor of " << x << " is " << d << endl;
```

```
if (condition1) {
  ...
} else if (condition2) {
  ...
} else if (condition3) {
  ...
} else {
  ...
}
```

```
if (condition1) {
  ...
} else {
  if (condition2) {
    ...
  } else {
    if (condition3) {
      ...
    } else {
      if (condition4) {
        ...
      } else {
        ...
      }
    }
  }
}
```

Drive the Turtle Around


```
bool done = false;
cout << "Enter f for forward, r for right, l for left, q to quit: ";
repeat(100) {
    if (!done) {
        char input;
        cin >> input;
        if (input == 'f' || input == 'F')  forward(100);
        else if (input == 'r' || input == 'R') right(90);
        else if (input == 'l' || input == 'L') left(90);
        else if (input == 'q' || input == 'Q') done = true;
        else cout << "Invalid input. Ignoring." << endl;
    }
}
```

Exercises

- Drive the turtle around, but if it tries to go outside the 500x500 box, refuse the command and print an error message
  - Hint: Keep track of the x and y coordinates of the turtle

- Simulate two turtles in the same window, accepting commands for them alternately (turtle pointer shows the turtle about to move)
  - Keep track of two (simulated) turtles' positions, and which one is the actual turtle. When switching turtles, penUp and move from one to the other.

- While simulating two turtles, refuse moves resulting in a collision
  - Can you further allow moves of arbitrary step sizes and angles, via commands of the form "r 45", "f 30" etc.? Note that now collisions can occur in the middle of a move, and you should refuse such moves too.
  - use `float` type for real numbers