Control Structure (Selection)

DM2111
C++ Programming

Introduction

Introduction	Array and Strings
Problem solving	Array and Strings
Basic elements of C++	Pointers
Basic elements of C++	Pointers
Statements	I/O operations
Repetition	Structs
Functions	Others
Functions	

Agenda

- Simple Statements
- Compound Statements
- if … else
- ?:
- switch

Statements

Most statements end with a semicolon

Beware of the semicolon!

The above is a *null statement*, and the cause of many sleepless nights for programmers.

Compound Statements

Aka **block**

Surrounded by curly braces

{ }

{ } is an *empty block*, equivalent to a null statement

Statements in block run sequentially

Statement Scope

Variables defined in the compound statement are not accessible outside of the statement.

```
int i = 0;
{
    int j = i;
    cout << i << ' ' << j;
}
i = 10; // i is still accessible here
j = 10; // j is not accessible here
```

Statement Scope

Variables can be defined in the control structure of *if*, *switch*, *while*, and *for* statements.

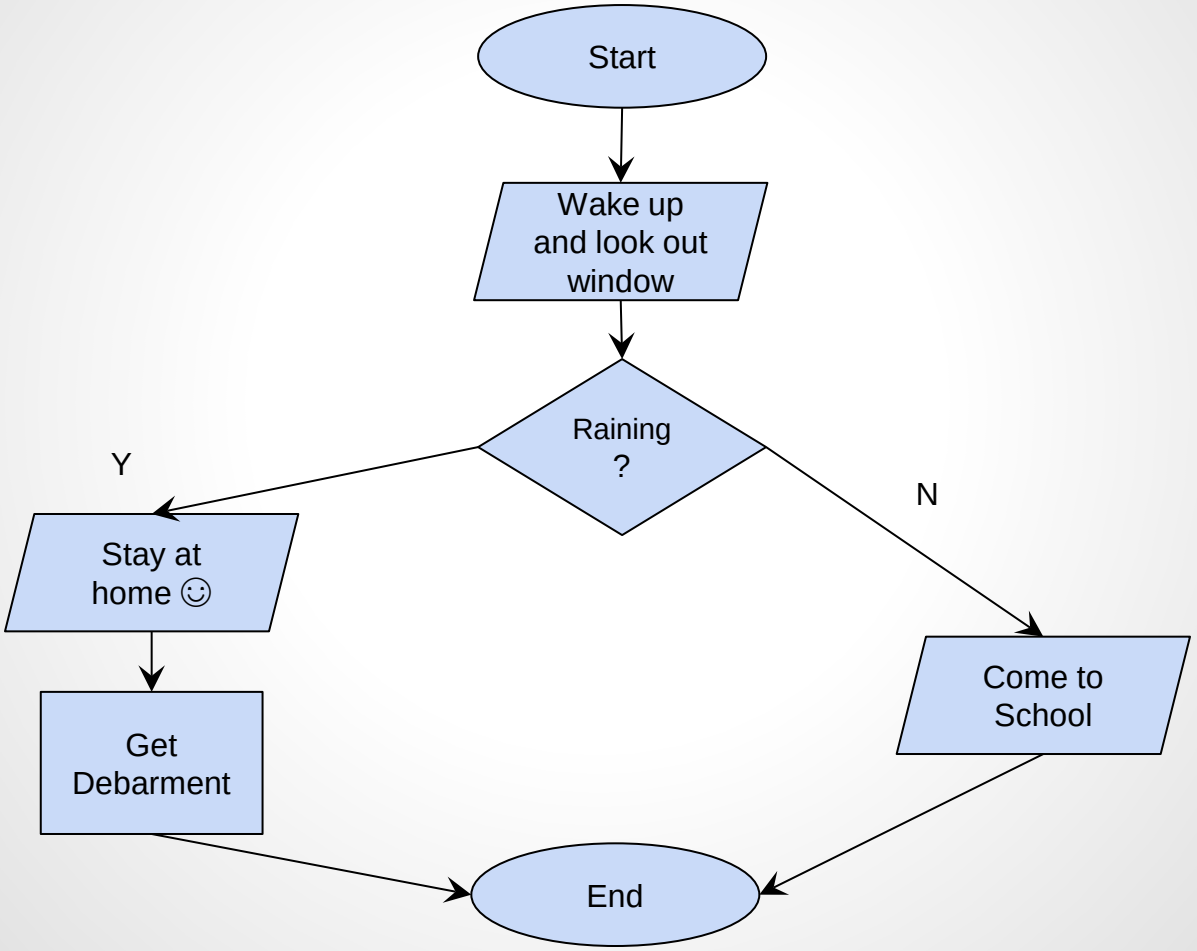
```
for (int i = 0; i < 10; ++i)
{
    cout << i;
}
i = 0; // i is not accessible here
```

Statements are executed sequentially

```
#include <iostream>
using std::cout;
using std::endl;

void main (void) {
    int age = 0;

    cout << "Please enter your age: ";
    cin >> age;

    cout << "Your age is " << age << endl;
}
```

Conditional Statements

You make selections everyday, for example, coming to school



Conditional Statements

Conditional Execution

- Program execution is based on condition
- Conditions are represented by expressions

The `if` statement

```
if (condition)
   statement
```

The *if else* statement

```
if (condition)
   statement1
else
   statement2
```

Conditions?

Remember expressions?

Expressions return a value

Value of expressions is converted to `bool` to determine condition

```
i == j
2 > 5
true
6
int i = 9
raining == true
raining
```

Consider a grading system...

```
if (score < 50)
{
   cout << "Fail";
}
```

How about a pass?

if...else

```
if (score < 50)
{
    cout << "Fail";
}
else // >= 50
{
    cout << "Pass";
}
```

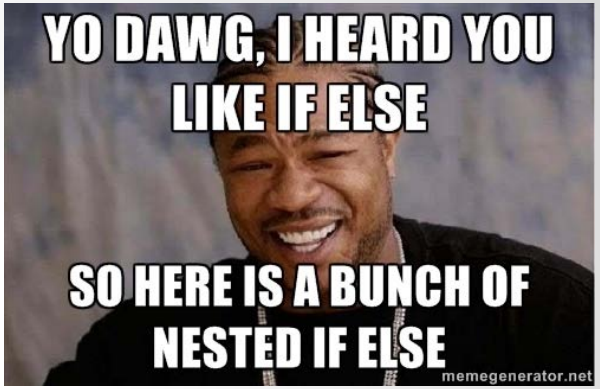
if...else

```
if (score < 50)
{
    cout << "Fail";
}
else // >= 50
    cout << "Pass";
    cout << " Well done!";
```

What is the problem here?

Nested `if` statements

```
if (kills == 2)
    cout << "Double kill!";
else if (kills == 3)
    cout << "Triple kill!";
else if (kills == 4)
    cout << "Ultra kill!";
else if (kills == 5)
    cout << "Rampage!";
else if (kills == 6)
    cout << "Killing spree!";
// and so on
```



What is the difference here?

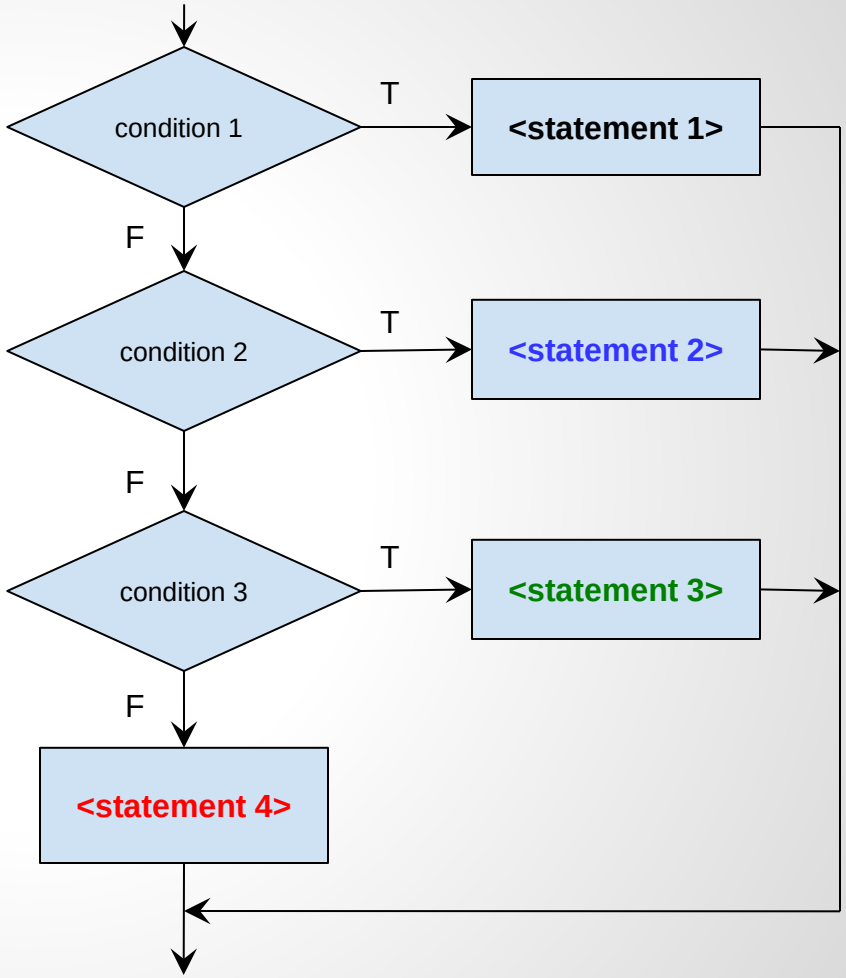
```
if (<condition 1>) {
    <statement 1>
}
else if (<condition 2>) {
    <statement 2>
}
else if (<condition 3>) {
    <statement 3>
}
else {
    <statement 4>
}
```

```
if (<condition 1>) {
    <statement 1>
}
if (<condition 2>) {
    <statement 2>
}
if (<condition 3>) {
    <statement 3>
}
else {
    <statement 4>
}
```

if...else

- Multiple selection

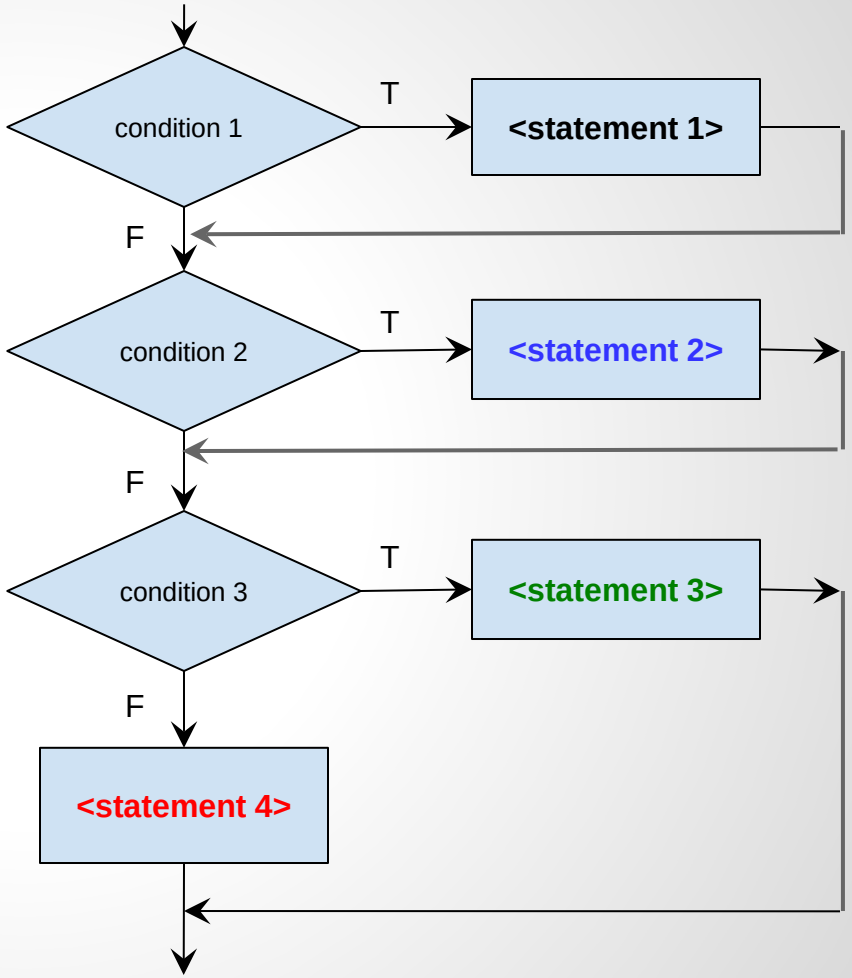
```
if (<condition 1>) {
    <statement 1>
}
else if (<condition 2>) {
    <statement 2>
}
else if (<condition 3>) {
    <statement 3>
}
else {
    <statment 4>
}
```



if...else

- Multiple selection

```
if (<condition 1>) {
    <statement 1>
}
if (<condition 2>) {
    <statement 2>
}
if (<condition 3>) {
    <statement 3>
}
else {
    <statement 4>
}
```



Dangling else

How do we know which `if` does an `else` statement belongs to?

```
if (sufficientMinerals)
    if (sufficientVespeneGas)
        cout << "Building...";
else
    cout << "Not enough minerals.";
```

Dangling else

Braces are your friends, or curly brackets, whatever...

```
if (sufficientMinerals)
{
    if (sufficientVespeneGas)
        cout << "Building...";
}
else
    cout << "Not enough minerals.";
```

A programmer is going to the grocery store and his wife tells him, "Buy a gallon of milk, and if there are eggs, buy a dozen."

So the programmer goes, buys everything, and drives back to his house. Upon arrival, his wife angrily asks him, "Why did you get 13 gallons of milk?"

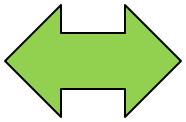
The programmer says, "There were eggs!"

?: Conditional Operator

```
<cond> ? <expr1> : <expr2>;
```

- If <cond> evaluates to true, <expr1> is evaluated; otherwise <expr2> is evaluated

```
larger = a > b ? a : b;
```



```
if (a > b)
    larger = a;
else
    larger = b;
```

```
int age;
cin >> age;

cout << (age > 12? "adult" : "kid") << endl;
```

switch

```
switch (<expression>)
{
  case <constant1>:
     <statement 1>;
     break;
  case <constant2>:
     <statement 2>;
     break;
  .
  .
  .
  default:
     <default statement>;
}
```

- `switch` is used to simplify multiple if's
- `switch` expression has to be integral
- Expression is matched against `case` values
- If a match is found, statements execute until `break` or end of switch
- If no match is found, `default` statements will execute

switch

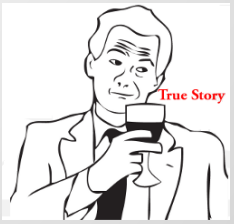
```
switch (kills)
{
    case 2: cout << "Double kill!";
            break;
    case 3: cout << "Triple kill!";
            break;
    case 4: cout << "Ultra kill!";
            break;
    case 5: cout << "Rampage!";
            break;
    case 6: cout << "Killing spree!";
            break;
    default: cout << "Something got killed!";
}
```

Control flow within a switch

Statements executes across `case` labels until a `break` is encountered

```
switch (grade)
{
    case 'A':
    case 'B':
    case 'C':
    case 'D':
    case 'E': ++passes;
              break;
    default: ++failures;
}
```

forgetting `break` is a common source of bugs



default

What is this `default` that you speak of?

```
switch (grade)
{
    case 'A':
    case 'B':
    case 'C':
    case 'D':
    case 'E': ++passes;
              break;
    default: ++failures;
}
```

If nothing else matches, `default` will execute. kthxbye