Topic 1 – CS1A Review – P3

Selection
Chapter 5 in the shrinkwrap

Control Structures (4 types)

- Sequential
  - The program flow moves from one statement to the next in the order it exists in the code (top to bottom)
- Selection (Decision)
  - Selection structures make decisions and execute commands depending upon the decision
  - If statements or Switch statements

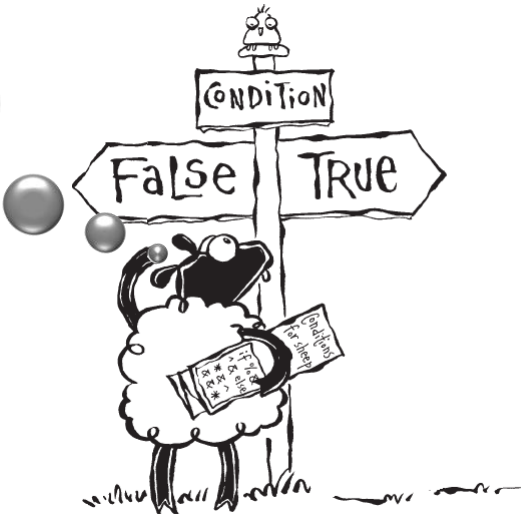
Control Structures (4 types) (2)

- Repetition
  - Used when something needs to be repeated
    - Could be a certain number of times
    - until a certain value has been reached or
    - a condition has been met
- Subprograms (functions)
  - A small piece of code that performs a specific task.

Today we will focus on Selection & Repetition
Don't worry → We'll cover functions later

Selection Structures

What if I only want some instructions to run some of the time?



Selection Structures

Selection
➔ Choosing between two or more alternative actions

- Run certain instructions based on some condition
- Conditions are based on Boolean Expressions
  - An expression that evaluates to 1 of 2 possibilities
  - Either True or False
- The computer evaluates a Boolean Expression and determines which instructions to execute based on the result
- Boolean expressions are formed using relational operators

Relational Operators

==	Equal
<	Less than
>	Greater than
<=	Less than or equal
>=	Greater than or equal
!=	Not Equal

NOTE: this is not the same as =,
= ← is an assignment

We use Relational Operators to compare values in Selection Statements
➔These will return a True (1) or False (0) value.

Examples of Boolean Expressions

What will be the result of these Boolean functions?

5 == 5

'a' < 'c'

4 + 3 > 10

10 != 20

6 <= 6

5 >= 9

'A' < 'Z'

'a' < 'Z'

If you compare characters using **relational operators**,
→It compares the **ASCII values**
(in this case 'a' has a greater ASCII value than 'Z'

SCII Chart for Printing Characters

Char	Decimal Value	Char	Dec	Char	Dec	Char	Dec
SP	32	8	56	P	80	h	104
!	33	9	57	Q	81	i	105
"	34	:	58	R	82	j	106
#	35	;	59	S	83	k	107
$	36	<	60	T	84	l	108
%	37	=	61	U	85	m	109
&	38	>	62	V	86	n	110
`	39	?	63	W	87	o	111
(	40	@	64	X	88	p	112
)	41	A	65	Y	89	q	113
*	42	B	66	Z	90	r	114
+	43	C	67	[	91	s	115
,	44	D	68	\	92	t	116
-	45	E	69	]	93	u	117
.	46	F	70	^	94	v	118
/	47	G	71	_	95	w	119
0	48	H	72	`	96	x	120
1	49	I	73	a	97	y	121
2	50	J	74	b	98	z	122
3	51	K	75	c	99	{	123
4	52	L	76	d	100	|	124
5	53	M	77	e	101	}	125
6	54	N	78	f	102	~	126
7	55	O	79	g	103	DEL	127

3-types of Selection Statements

- One-way Decisions
  - If the condition is true then execute some instructions
  - If the condition is false → don't do anything special
- Two-way Decisions
  - If the condition is true then execute some instructions
  - else (the condition is false) execute another set of instructions
- Multi-way Decisions
  - Nested If-Then or Nested If-Then-Else Statements
  - Many options…

3-types of Selection Statements

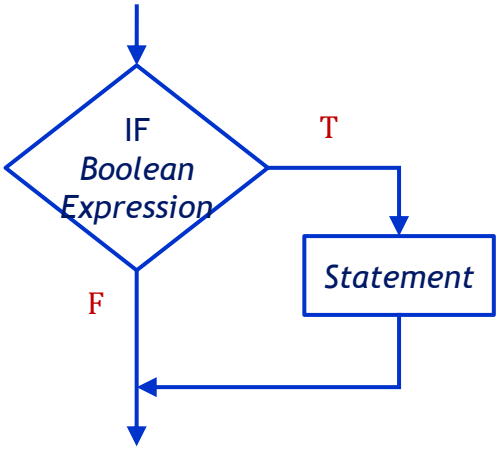
- One-way Decisions
  - If-Then Statements
- Two-way Decisions
  - If-Then-Else Statements
  - Conditional Statements
- Multi-way Decisions
  - Nested If-Then or Nested If-Then-Else Statements
    - Nested Conditional
  - If-Then-Else-If
  - Switch statements

If Statements

- If statements take different forms
  - For now we will focus on the 2 basic forms
    - If-Then
    - If-Then-Else
  - Both of these statements can be nested
- A simple "if-then statement" is a one-way stmt
  - One-way decisions
    - If a condition is true → execute some special instructions

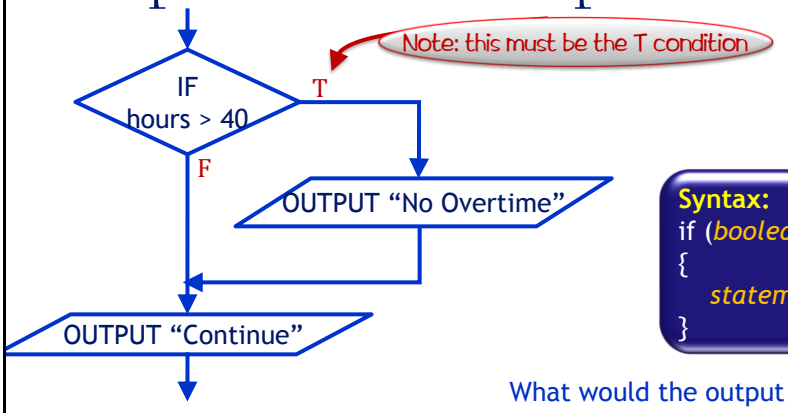
Syntax:
if (*boolean expression*)
{
  statements;
}

Flowchart for If-Then Statement



1. The Boolean expression is evaluated
2. If it evaluates to TRUE, then the True Instructions are executed
3. If it evaluates to FALSE the statement is ignored and the program continues with the next executable statement

Example: If-Then Example



Syntax:
if (*boolean expression*)
{
  statements;
}

```
if (hours <= 40)
{
    cout << "No overtime" << endl;
}
cout << "Continue" << endl;
```

What would the output be for…

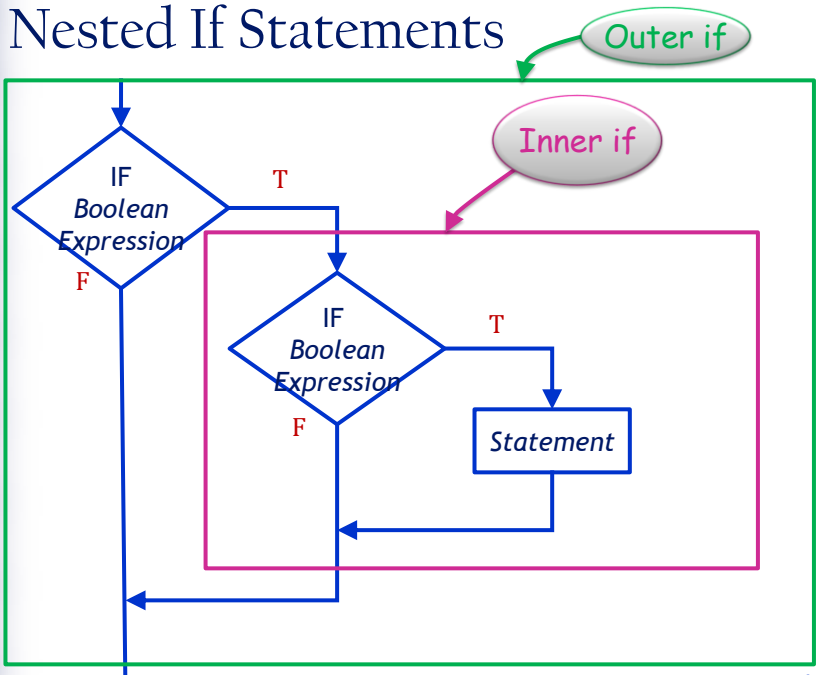
hours = 40?

hours = 120?

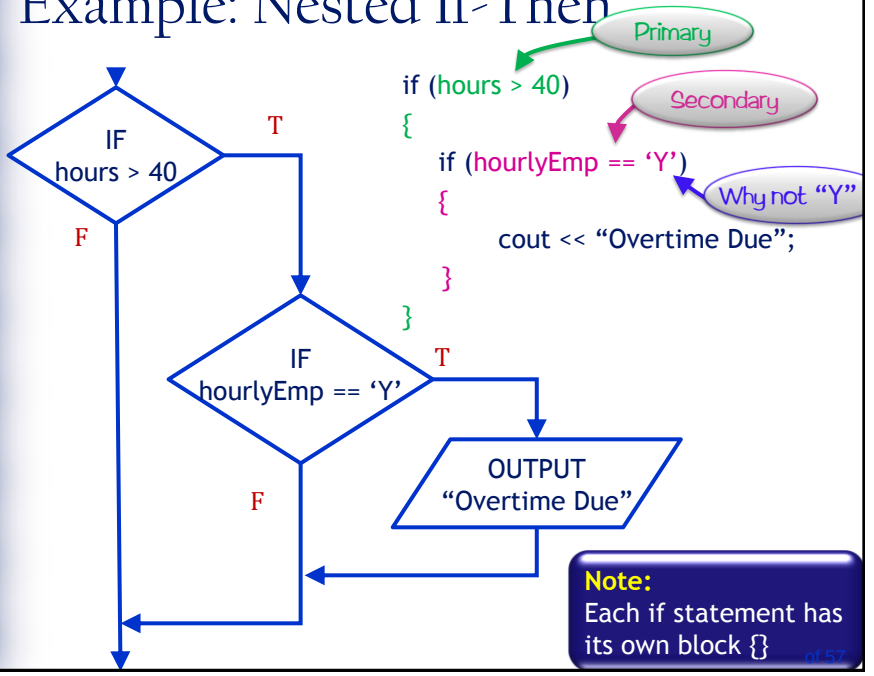
Nested If Statements

- Nested Selection Structure
  - Selection structure within another selection structure
  - Used when more than one decision must be made before an appropriate action can be carried out
  - For example, an If-Then statement that contains another If-Then statement (within the statement section)
- Primary Decision
  - always made by the outer selection structure
- Secondary Decision
  - Always made in the inner (or nested) selection structure

Nested If Statements



Example: Nested If-Then



If-Then-Else Statements

- Two-way Decisions
- Either execute one set of instructions or another
- Based on a Boolean expression

If the condition is true then

- Execute one set of instructions

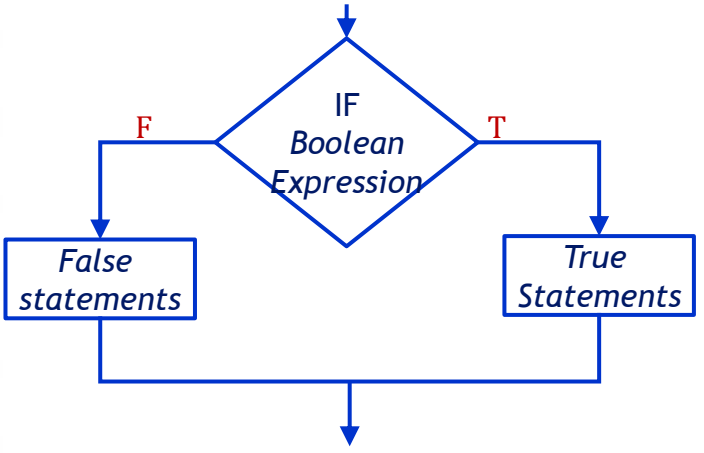
Else

- Execute another set of instructions

Syntax:

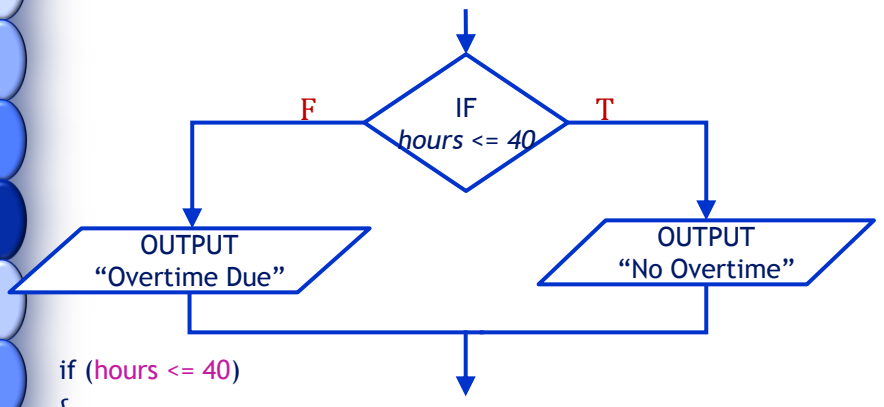
```
if (boolean expression)
{
   statement(s);
}
else
{
   statement(s);
}
```


Flowchart for If-Then-Else Stmt



1. The Boolean expression is evaluated
2. If it evaluates to TRUE, then statement1 is executed
3. If it evaluates to FALSE, then statement2 is executed

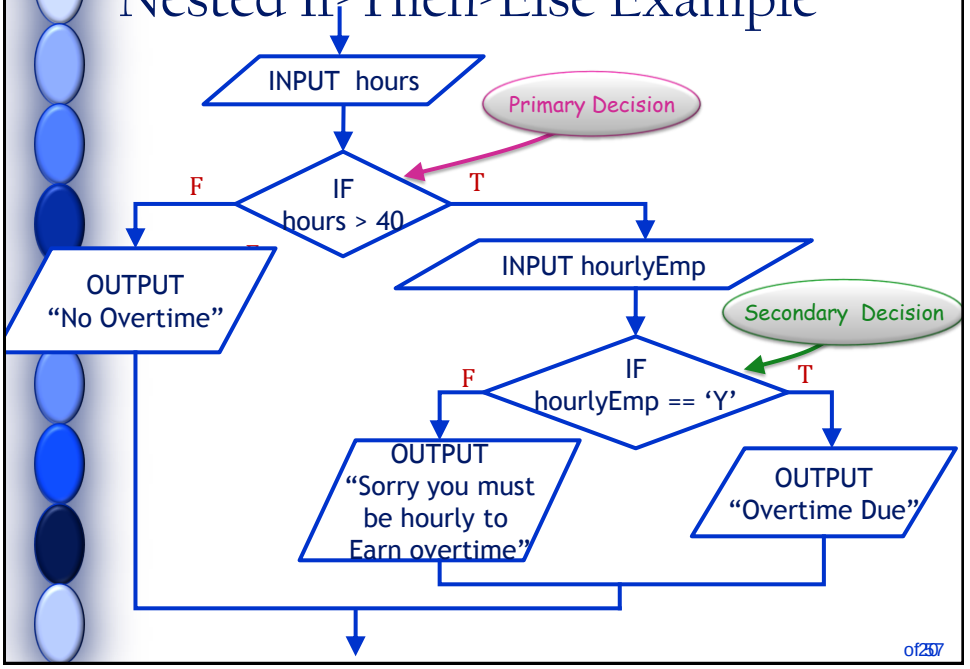
Flowchart for If-Then-Else Stmt




```
if (hours <= 40)
{
    cout << "No Overtime" << endl;
}
else
{
    cout << "Overtime Due" << endl;
}
```

If-Then-Else statements can also be nested

Nested If-Then-Else Example



```
cout << "Enter hours worked: ";
cin >> hours;
if (hours > 40)
{
    cout << "Are you an hourly employee?  ";
    cin.get(hourlyEmp);
    if (hourlyEmp == 'Y')
    {
        cout << "Overtime Due";
    }
    else
    {
        cout << "Sorry, you must be hourly to earn overtime";
    }
}
else
{
    cout << "No Overtime";
}
```

Outer If-Then-Else

Will this work?

Inner If-Then-Else

Comparing Floating Point Values

- Floating point values are a little trickier to compare than integers
- This is because of the way they are stored in memory
  - They are rounded so it is rare that they will evaluate to be the same (even if you evaluate them to be the same)
- Thus, we just want to check if they are "close enough" to call them equal
- One method
  - First calculate the absolute value of the difference of the two numbers
  - Second, check it against some very small epsilon value

Example

```
const float EPSILON = 0.00001;
float val1, val2;

if (fabs(val1 - val2) < EPSILON)
{
   cout << "values are equal";
}
```

EPSILON is some value which we determine will be within a "close enough" margin
In this case it is 0.00001

fabs
→A C++ library function that returns the absolute value of a floating point expression.

Comparing c-strings

Cstrings are stored in an array

- Remember an array is a contiguous area of storage where each element has the same data type
- cstrings are an array of characters

- When you access an array you are working with the address of the array → Not the value in the address
- When you make the following comparison:

  if(stringOne == stringTwo)

  you are comparing the addresses →not the values
  → **the addresses will never be the same**

strcmp function

strcmp(*c-string1*, *c-string2*);

- Compares the contents of string1 and string 2
- Returns:

Return value	if ASCII VALUES are such that
0	string1 == string2
Integer < 0	string1 < string2
Integer > 0	string1 > string2

NOTE: String comparisons should be of the same **size**
→ **can't do**
char str1[8];
char str2[10];
If (str1 == str2) <<- can't compare these

Example

```
char stringOne[10];
char stringTwo[10];

cout << "Enter the first string: ";
cin.getline(stringOne, 10);

cout << "Enter the second string: ";
cin.getline(stringTwo, 10);

if(strcmp(stringOne, stringTwo) == 0)
{
   cout << "The strings are the same";
}
```

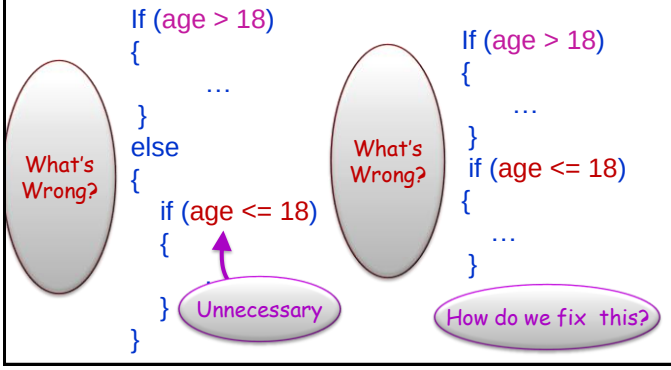
Common Syntax Errors in If Structures

Syntax errors:

- Forgetting the parenthesis
  - Eg. If (hours > 40) → NOT if hours > 40
- Putting a ";" at the end of the first line
  - Eg. if (hours > 40) → NOT if (hours > 40);
    - **This statement is correct it just won't do anything**

Common Logic Errors in If Structures

- Logic errors are commonly made as a result of the following mistakes:
  - Reversing the primary and secondary decisions
  - Redundancy
    - Using an unnecessary nested selection structure
  - Using if-then instead of if-then-else



Conditional Operator

- Shortcut to create a simple if-then-else statement

Syntax:
(condition? true_statements: false_statements);

Example:

```
overTime = (hrsWkd > 40 ? (hrsWkd-40)*rate*1.5: 0.0);
```

NOTE: the assignment goes first

Condition to be Tested precedes the?

If true this statement is executed

If false this statement is executed

What would the value of overtime be i

hrsWkd = 40 and rate = 10 ? 0.0

hrsWkd = 60 and rate = 10 ? 300.0

Note:
This can also be used with a cout << statement

Conditional Operator Example

```
if (hrsWkd > 40)
   overTime = (hrsWkd - 40) * rate * 1.5;
else
   overTime = 0.0;
```

```
overTime = (hrsWkd > 40 ? (hrsWkd - 40)* rate * 1.5 : 0.0);
```

- You can use this anywhere you can use an expression
  - an assignment statement (see above)
  - or a cout statement

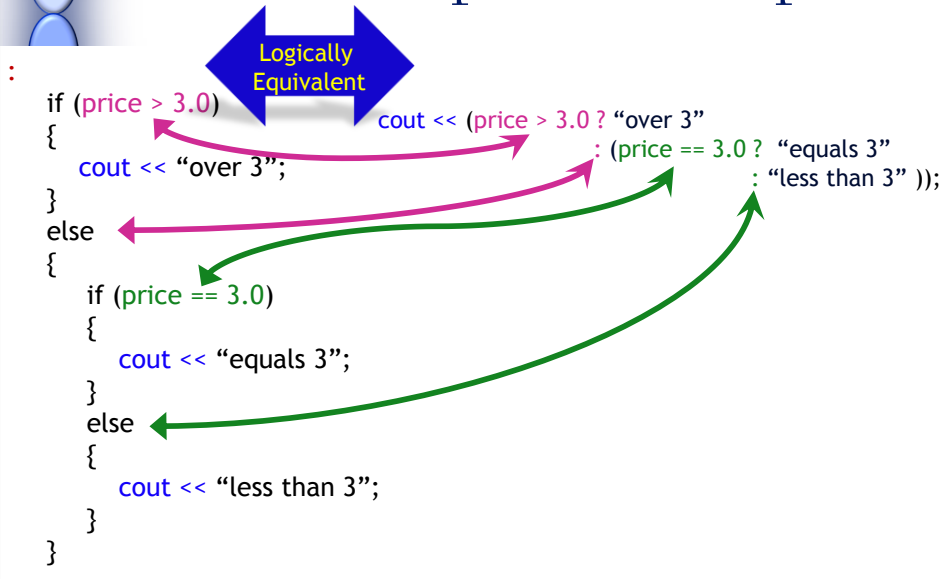
```
cout << (hrsWkd> 40 ? (hrsWkd - 40)* rate * 1.5 : 0.0);
```

You can nest them too

Let's say hourly employees get 1½ * their rate for overtime, but other employees just get regular pay.

```
overTime =   hrsWkd > 40  ? (hrlyEmp ==‘Y’ ?  (hrsWkd - 40)* rate * 1.5
                                                : (hrsWkd - 40)* rate)
                          : 0.0;
```


Conditional Operator Example



Exercises

```
int age
float price;
cout << "Age: ";
cin >> age;
```

Given the above declarations answer the following.

1. Use the conditional operator in an assignment statement that assigns 12.50 to a variable called price if the age is less than or equal to 5 and 25.50 otherwise.

2. Use the conditional operator in a cout statement that outputs 12.50 if the age is less than or equal to 5, 25.50 if the age is less than or equal to 12 and 37.50 otherwise without assigning it to the variable called price. (NOTE: You will need to nest them here)

3-types of Selection Statements

- One-way Decisions
  - If-Then Statements
- Two-way Decisions
  - If-Then-Else Statements

  OR
  - ***Conditional Statements***
- Multi-way Decisions
  - Nested If-Then or Nested If-Then-Else Statements

  OR
  - ***If-Then-Else-If***
  - ***Switch statements***

Today we will focus on **if-then-else-if**, **switch statements**, & the **conditional operator**.

If-Then-Else-If

Nested if's are one way of coding multi-way decisions.

If-Then-Else-If statements are also multi-way decisions.

Executes if the first logical expression is true → Otherwise it drops to the next else if

This executes if the 1st logical expression was false and the 2nd logical expression was true → Otherwise it drops to the next else if and so on

This executes if all the previous logical expressions evaluated to false

```
Syntax:
if ( logical expression )
{
    stmtT1;
}
else if ( logical expression )
{
    stmtT2;
}
. . .
else if ( logical expression )
{
    stmtTN;
}
else
{
    stmtF;
}
```


Example – If-Then-Else-If

Lets say you wanted to output what class a user is in based on a variable called classCode.

classCode is of type char and represents the following values:

F → freshman
S → sophomore
J → junior
R → senior

```
if (classCode == 'F')
{
    cout << "You are a freshman!" << endl;
}
else if (classCode == 'S')
{
    cout << "You are a sophomore!" << endl;
{
else if (classCode == 'J')
{
    cout << "You are a junior!" << endl;
}
else if (classCode == 'R')
{
    cout << "You are a senior!" << endl;
}
else
{
    cout << "Invalid class Code!" << endl;
}
```

It is a good practice to handle unexpected inputs

Switch Statement

- Allows for multi-way selection
- Eliminates the need for many nested ifs

Syntax:

```
switch ( expression )
{
        case constant-expression : statement ;
        ...
        default : statement;
}
```

- If the expression evaluates to the constant- expression then the appropriate statement(s) is executed
- Otherwise the default statement is executed

Switch Example

```
switch (classCode)
{
    case 'F' : cout << "You are a freshman!" << endl;
               break;
    case 'S' : cout << "You are a sophomore!" << endl;
               break;
    case 'J' : cout << "You are a junior !" << endl;
    case 'R' : cout << "You are a senior!" << endl;
               break;
    default : cout << "You entered an invalid code";
               break; // not necessary but a good habit
}
```

What will happen if classCode == 'J'?

Break Statement

- The break statement forces a block of code to exit (or terminate).
- If you don't break in a switch statements all of the statements succeeding a case will execute!
- This can be useful if you want the same code to execute under multiple cases (or situations).

WARNING:
Switch statements are the only time you should use the break statement.
It is considered bad practice to use it in a loop

Break Statement

- Break statement prevents the case statement from following through.
- It can be useful in some situations

```
switch (classCode)
{
    case 'F' :
    case 'f' : cout << "You are a freshman!" << endl;
    break;
    case 'S' :
    case 's' : cout << "You are a sophomore !" << endl;
    break;
    etc.
```

Break Statement

- The break statement forces a block of code to exit (or terminate).
- If you don't break in a switch statements all of the statements succeeding a case will execute!
- This can be useful if you want the same code to execute under multiple cases (or situations).

WARNING:
Switch statements are the only time you should use the break statement.
It is considered bad practice to use it in a loop or if statement!!

Final Notes

- Can an if-then-else (or else-if) structure can replace any switch statement?
- Can a switch statement replace any if-then-else-if structure?
- Switch statements are based off the same variable
  - Best used if there are many unique conditions
- Which statement should you use for a two-way statement?

Which should you use?

- If you need to compare a range of values (e.g. > 5.. between 5 and 10)
- If you need to compare values of several variables
- If you need to compare for equality of different values against one variable…

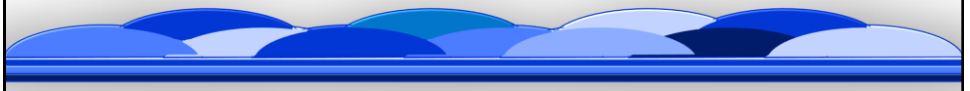
NOTE: The statements within any selection structure should be unique to that condition!

Unique statements only please

What is wrong with this?

```
if (classCode == 'F')
{
    studentCount = studentCount + 1;
    cout << "You are a freshman!" << endl;
}
else if (classCode == 'S')
{
    studentCount = studentCount + 1;
    cout << "You are a sophomore!" << endl;
{
else
{
     studentCount = studentCount + 1;
    cout << "You are a junior!" << endl;
}
```

Boolean Operators



Logical Boolean Operators & Expressions

Boolean Operators are used to form complex decision statements

Function	Syntax	# of operands
NOT	!	Unary
AND	&&	Binary
OR	||	Binary

We use these operators in conjunction with relational operators (<, >, ==, etc.)

George Boole:
Developed Boolean logic and published it in 1847 in his book, "The Mathematical Analysis of Logic"
This led to the field of symbolic logic.

NOT (!)

NOT (denoted as !)

- When it precedes a single expression it gives the opposite result.
- Unary operator (only needs 1 operand)

Example

!(hrsWkd <= 40)

hrsWkd > 40

Truth Table

Value	after !
T(1)	F(0)
F(0)	T(1)

Negating the relational operators

! (a < b) ⇔ a >= b

! (a >= b) ⇔ a < b

! (a == b) ⇔ a != b

AND (&&)

AND (Denoted as &&)

- Combines two logical expressions and requires that both expressions be true for the entire expression to be true

Example

num1 <= 10 && num1 != 5

This expression would be TRUE only if

num1 <= 10 AND num1 does not = 5

exp1 (num1 <= 10)	exp2 (num1 != 5)	&&
T(1)	T(1)	T(1)
T(1)	F(0)	F(0)
F(0)	T(1)	F(0)
F(0)	F(0)	F(0)

Only time && is TRUE is if they both are true

OR (||)

OR (Denoted as ||)

- combines two logical expressions and states that if either or both of the expressions are TRUE, the entire expression is TRUE

Example

num1 < 1 || num1 > 10

This expression would be FALSE only if

num1 >= 1 AND num1 <= 10

exp1 (num1 < 1)	exp2 (num1 > 10)	||
T(1)	T(1)	T(1)
T(1)	F(0)	T(1)
F(0)	T(1)	T(1)
F(0)	F(0)	F(0)

Only time || is FALSE is if they both are false

DeMorgan's law

- Lets say that exp1 & exp2 are boolean expressions so they evaluate to TRUE of FALSE

Note: The Symbols Change

!(exp1 || exp2) ⇔ !exp1 && !exp2

!(exp1 && exp2)⇔ !exp1 || !exp2

NOTE: When using Boolean operators the expression must evaluate to T or F

DeMorgan's Law #1

!(exp1 || exp2) ⇔ !exp1 && !exp2

ex1	ex2	ex1 || ex2	!(ex1||ex2)	!ex1	!ex2	!ex1 && !ex2	!ex1 || !ex2
T	T						
T	F						
F	T						
F	F						

Note: These values are equivalent

Exercise - Now it is your turn:
Using a truth table show DeMorgan's 2nd law:
!(exp1 && exp2) ⇔ !exp1 || !exp2

DeMorgan's Law #2

!(exp1 && exp2) ⇔ !exp1 || !exp2

ex1	ex2	ex1 && ex2	!(ex1 && ex2)	!ex1	!ex2	!ex1 || !ex2	!ex1 && !ex2

Exercises

Rewrite the expressions distributing the !

- !(num1 == num2)

  ⇔

- !(num1 == num2 || num1 == num3)

  ⇔

  ⇔

- !(num1 == num2 && num3 > num4)

  ⇔

  ⇔

Math notation is not C++ syntax

5 < x < 15 is okay in math

This is not equivalent to 5 < x < 15 in C++

How would C++ view this?

Write the equivalent in C++

C++ uses Short-Circuit Evaluation

- **Short Circuit Evaluation** refers to how a language evaluates logical expressions
- Left to Right order
- When using an AND (&&) operator evaluation **stops** as soon as FALSE condition is found
- When using an OR (||) operator evaluation **stops** as soon as a TRUE condition is found

Declaring a Boolean Variable

The Boolean data type can be assigned one of two values: true or false.

Syntax:
bool *variableName*;

EXAMPLE

```
int main()
{
   bool dataOK;
   int n1, n2;

   dataOK =  n1 < 4 & n2 !=3 ;

   if (dataOK)
   {
      cout << "all is good";
   }
   else
   {
      cout >> "bad data";
   }
}
```

How would you check if dataOK was false?

Declaring a Boolean Variable

```
int in1, in2, in3, in4;
bool bl1, bl2;
…
bl1 = in1 > in2;
bl2 = in3 > in4;
if (bl1 && bl2)
{
   cout << "The bl1 & bl2 are true") << endl;
}
else if (bl1 || bl2)
{
   cout << "Either the bl1 is true or the bl2 is true" << endl;
}
cout << bl1<< "\t" << bl2;
```

What would output if →
in1 = 32, in2 = 4, in3 = 45, in4 = 5

in 1 = 32, in2 = 4, in3 = 5, in4 = 45

in1 = 4, in2 = 4, in3 = 5, in4 = 45

in1 = 4, in2 = 43, in3 = 45, in4 = 5

Order of Precedence (Expanded)