C++ CONDITIONS AND LOOPS

USING BOOLEAN EXPRESSIONS

expression – a combination of variables, operations, and values that yields a result value.

Mathematical expression examples:

```
total / numWinners
```

```
Winnings = total / numWinners; //statement
```

```
(rate * 40) + (1.5 * rate) * (hours - 40)
```

```
grossPay = (rate * 40) + (1.5 * rate) * (hours - 40) ; //statement
```

Boolean expression – an expression that can be thought of as being true or false.

```
hours <= 40
```

- A Boolean expression can be evaluated in the same way that an arithmetic expression is evaluated
- The difference is it can only produce 2 values

USING BOOLEAN EXPRESSIONS

Flow of control:

- If/else
- switch
- while
- do/while
- for

```
<   // less than
> //greater than
<=  //less than or equal to
>=  //greater than or equal to
==  //equal to
!=  //not equal to
```

```
&& //logical AND
||  //logical OR
!  //NOT
```

USING BOOLEAN EXPRESSIONS

```
if (hours <= 20) //20 or less hours
{
    grossPay = rate * 40;
    cout << "\nYou worked part time this week!";
}
else if (hours <= 40)//21 to 40 hours
     //curly braces are not required, but may make it easier to read
    grossPay = rate * 40;
else //more than 40 hours
```

BRANCHING MECHANISM

Branching mechanism – any programming construct that chooses one from a number of alternate actions

When using control structures (if, while, for, etc) use curly braces for more than one item(statement)

Dangling else problem – The compiler may pair the else with the wrong if

```
if (hours <= 20) //20 or less hours
{
     grossPay = rate * 40;
     cout << "\nYou worked part time this week!";
}
else if (hours <= 40)//21 to 40 hours
      //curly braces are not required, but may make it easier to read
     grossPay = rate * 40;
else //more than 40 hours
{
      //nested if, will only printf if it is more than 65
      //curly braces are not required, but may make it easier to read
     if (hours > 65)//more than 65 hours
          cout << "\nYou worked way too many hours this week!";
     grossPay = rate * 40;
     overTimeHours = (hours - 40);
     grossPay = (rate * 40) + (1.5 * rate) * overTimeHours;
     cout << "\nTime and a half rate is $" << (1.5 * rate) << endl;
}
```

SWITCH

Another type of multiway branch used for variable equality

```
switch (grade) {
     case 'A':
     case 'a':
          cout << "Excellent!\n";
          gradepoints += 4.0;
          break;
     case 'B':
     case 'b':
          cout << "Very good!\n";
          gradepoints += 3.0;
          break;
     case 'C':
     case 'c':
          cout << "Passing.";
          gradepoints += 2.0;
          break;
     default:
          cout << "That is not a valid letter grade!";
}
```

LOOPS

Loop – any program construct that repeats a statement or sequence of statements a number of times

Iteration – each repetition of the loop

while – controlling Boolean expression (test) before the loop

do / while - controlling Boolean expression (test) at the end of the loop

for - controlling Boolean expression (test) before the loop

initialization

test (controlling Boolean expression)

update

INCREMENT AND DECREMENT OPERATORS

```
int num = 2;

num++; //as a statement

//the expression num++ will first return num and then add one
num++ //as an expression

//the expression ++num will add one before
++num //as an expression
```

INCREMENT AND DECREMENT OPERATORS

```
How many times will this loop iterate?

    const int max1 = 5;

    int main()
    {
        int count = 0;

        cout << "\ncount is " << count << " before the loop";
        while (count++  < max1) //test then add
         //++count  //add then test
        {
            cout << "\ncount is " << count;
        }

        return  0;
}
```

```
//count ++

count is 0 before the loop
count is 1
count is 2
count is 3
count is 4
count is 5
count is 6 after the loop
```

```
//++count

count is 0 before the loop
count is 1
count is 2
count is 3
count is 4
count is 5 after the loop
```

Branching (if)

```
//assume variables have values and this is inside a function body

if (hours > 40)
{
    //overtime calculation
    overTimeHours = (hours - 40);
    grossPay = (rate * 40) + (1.5 * rate) * (hours - 40);
}
```

Branching (example: if/ else)

```
//assume variables have values and this is inside a function body

if (hours > 40)
{
    //overtime calculation
    overTimeHours = (hours - 40);
    grossPay = (rate * 40) + (1.5 * rate) * (hours - 40);
}
else
    //no overtime
    grossPay = rate * 40;
```

Looping (example: while)

```
//assume this is inside a function body

int countdown;
cout << "How many greetings do you want? ";
cin >> countdown;  //initialization

//while loop example
while (countdown > 0) ////Boolean expression (test)
{
    cout << "Hello! ";
    countdown--; //update
}
```

FLOW OF CONTROL

- **Looping (**example: **do/while**)

```
//assume this is inside a function body

char ans;

//do/ while loop example
do {
       cout << "Welcome!\n";
       cout << "Do you wnat another greeting?\n"
            << "Press y for yes, n for no, \n"
            << "and then press return: ";
       cin >> ans; //update

} while (ans == 'y' || ans == 'Y'); //test
```

BLOCKS OF CODE AND LOCAL VARIABLES

For loops: (initialization; test; update) //separated by semicolons

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

int main()
{
    int sum = 0;
    for (int n = 1; n <= 10; n++) //n is a local variable to the for loop
    {
        sum += n;
        cout << "\nThe current sum is " << sum;
    }
    cout << "\nThe value of n is " << n; //error n is local to the for loop BLOCK

    return  0;
}
```

DESIGNING LOOPS

- initialization
- test (controlling Boolean expression)
- update

When selecting a loop, first design the loop using pseudocode.

A **for loop** is best for loops that change by an equal amount for each iteration

If there are circumstances in which the loop body may not be executed at all, a **while loop** is the best choice

A **do/while loop** will always execute once

DESIGNING LOOPS

***Testing a loop** – each loop should be tested with inputs for at least*
zero iterations**, **one iteration**, and **maximum iterations

Methods for ending an input loop

1. List headed by size – example, ask the user how many times
2. Ask at each iteration – example, continue ? (y or n)
3. List ended with a sentinel value
4. Running out of input

General loop ending techniques:

1. Count controlled loops – determines the number before beginning the loop
2. Ask before iterating – again?
3. Exit on a flag condition – a flag is a variable that changes value to indicate some event has taken place

It is very common to have "off by one" loops