Control Flow

April 21, 2018

C++ has several control flow options.

- if-else statements
- for loops
- while loops
- switch statements

1 Overview of relational & logical operators

Operator	Example	True Condition	False Condition
==	A == B	A equal to B	A is not equal to B
!=	A != B	A is not equal B	A equals B
>	A > B	A is greater than B	A is not greater than B
<	A < B	A is less than B	A is not less than B
>=	A >= B	A is greater or equal to B	A is not greater or equal to B
<=	A == B	A is less or equals to B	A is not less or equal to B

In [1]:
```
# include <string>
# include <iostream>
//instead of printing 0 and 1, create an array where
//0 = False, 1 = True
std::string TorF[] = {"False", "True"};

int a = 1;
int b = 2;
int c = 2;

//Print out the string values of each relational operation
std::cout<<"a < b is "<<TorF[a<b];
std::cout<<"\na > b is "<<TorF[a>b];
std::cout<<"\na != b is "<<TorF[a!=b];
std::cout<<"\nc >= b is "<<TorF[c>=b];
std::cout<<"\nc <= b is "<<TorF[c<=b];
```

```
a < b is True
a > b is False
a != b is True
c >= b is True
c <= b is True
```

2 Logic Operators

There are three logic operators in C++:

- and
- or
- not

Assume A = True, B = True, C = False, D = False

Operator	Example	True Condition
&&	**Logical And:** if both the outcomes are *True* the outcome of the operation is *True*	A && B = True A && C = False
||	**Logical Or:** if both the outcomes are *False* the outcome of the operation is *False*	C || D = False A || C = True
!	**Logical Not:** negates the logical condition	!(A && B) = False !(A||C) = False

In [2]:
```
# include <iostream>
# include <string>

int A = 1;
int B = 2;
int C = 2;
int D = 3;

//std::string TorF[] = {"False", "True"};// - I am commenting it out because I have used

//A true && false = false
std::cout<<"\n\n(A == C) && (A == D) is "<<TorF[(A ==C) && (A == D)];
//A true and true = true
std::cout<<"\n(B == C) && (B < D) is "<<TorF[(B == C) && (B < D)];

//The || operator
//A false || true = true
std::cout<<"\n\n(A == C) || (B == C) is "<<TorF[(A == C) || (B == C)];
//A false || false = false
```

```
        std::cout<<"\n(A == C) || (B > D) is "<<TorF[(A == C) || (B > D)];

        //The 'Not' operator
        // true = true
        // !(true) = false
        std::cout<<"\n\nA < B is "<<TorF[A<B];
        std::cout<<"\n!(A < B) is "<<TorF[!(A<B)];

        // false= false
        // !(false) = true
        std::cout<<"\n\nA == C is "<<TorF[A==C];
        std::cout<<"\n!(A == C) is "<<TorF[!(A==C)];
```

```
(A == C) && (A == D) is False
(B == C) && (B < D) is True

(A == C) || (B == C) is True
(A == C) || (B > D) is False

A < B is True
!(A < B) is False

A == C is False
!(A == C) is True
```

3 Conditional Statements

3.1 IF Statement

```
In [3]: int a_if = 0;
        if(a_if == 1)
        {
            std::cout<<"a_if is equal to 1.\n";
        }
```

```
In [4]: int a_if_2 = 1;
        if(a_if_2 == 1)
        {
            std::cout<<"a_if_2 is equal to 1.\n";
        }
```

```
a_if_2 is equal to 1.
```

3.2 IF-ELSE Statements

if(boolean expression) { //statements to execute if the boolean expression is true } else { //statements to execute if the boolean expression is false }

3.3 IF-ELSE IF - ELSE Statements

if(boolean expression) { //statements to execute if the boolean expression is true } else if (boolean expression #2) { //statements to execute if the 'else if' boolean expression #2 is true }
else { //statements to execute if the boolean expressions //'if' and 'else if' are false }
Exercise:
Create a program, where a user inputs a guess and checks it whether it matches the target.
Answer:

```
In [5]: int main()
        {
            int TARGET = 33;
            int guess;
            std::cout<<"Guess a number between 0 - 100\n";
            std::cin>>guess;

            std::cout<<"You guessed: "<<guess<<"\n";


            if(guess < TARGET)
            {
                std::cout<<"Your guess is too low.\n";
            }
            else if(guess > TARGET)
            {
                std::cout<<"Your guess is too high.\n";
            }
            else
            {
                std::cout<<"Yay! You guessed correctly.\n";
            }
        }
```

```
In [6]: // assume that your guess is 20

        int TARGET_d = 33;
        int guess_d = 20;
        std::cout<<"Guess a number between 0 - 100\n";

        std::cout<<"You guessed: "<<guess_d<<"\n";


        if(guess_d < TARGET_d)
        {
```

```
        std::cout<<"Your guess is too low.\n";
    }
    else if(guess_d > TARGET_d)
    {
        std::cout<<"Your guess is too high.\n";
    }
    else
    {
        std::cout<<"Yay! You guessed correctly.\n";
    }
```

```
Guess a number between 0 - 100
You guessed: 20
Your guess is too low.
```

Exercise
Using if-else statements, write a program that will ask questions and select a pet for the user:
The questions:
"Do you want a pet with fur(f), feathers(t), or scales(s)?"
"You want a pet with scales. Do you want the pet to live in water(w), on land(l), or both(b)?
The pet choices should be: dog, bird, fish, gecko, frog

In [7]:
```
#include <iostream>

    int pet_selection()
    {
       char skin, location;
       std::cout<<"Would you like an animal with fur(f), feathers(t), or scales(s)?";
       std::cin>>skin;
       std::cout<<skin<<"\n";

       if(skin == 'f')
        {
            std::cout<<"Get a dog"<<"\n";
        }
        else if(skin == 't')
        {
            std::cout<<"Get a bird"<<"\n";
        }
        else if(skin == 's')
        {
            std::cout<<"Would you like an animal that lives in water(w), land(l), or both(b)
            std::cin>>location;
            std::cout<<location<<"\n";

            if(location == 'w')
                std::cout<<"Get a fish"<<"\n";
```

```
            else if(location == 'l')
                std::cout<<"Get a gecko"<<"\n";
            else if(location == 'b')
                std::cout<<"Get a frog"<<"\n";
            else
                std::cout<<"Enter water(w), land(l), or both(b)\n";
        }
        else
            std::cout<<"Please choose fur(f), feathers(t), scales(s)"<<"\n";
        return 0;
    }
```

3.4 Switch Statements

Now that we have talked about if-else statements, this may be a good time to talk about the king of 'if' statements, the switch statement.

Switch Statements

The format for a switch statement:

switch(expression) {) **case** constant-expression : statements;) break; (this is optional);) **case** constant-expression : statements;) break; (this is optional);) . . .) **default** : //optional) statements;) })

In [8]:
```
#include <iostream>

int switch_example()
{
    int menuItem = 1;

    std::cout<<"What is your favorite winter sport?: \n";
    std::cout<<"1.Skiing\n2: Sledding\n3: Sitting by the fire";
    std::cout<<"\n4.Drinking hot chocolate\n";
    std::cout<<"\n\n";

    switch(menuItem)
    {
        case(1): std::cout<<"Skiing?! Sounds dangerous!\n";
                 break;
        case(2): std::cout<<"Sledding?! Sounds like work!\n";
                 break;
        case(3): std::cout<<"Sitting by the fire?! Sounds warm!\n";
                 break;
        case(4): std::cout<<"Hot chocolate?! Yum!\n";
                 break;
        default: std::cout<<"Enter a valid menu item";
    }

    char begin;
    std::cout<<"\n\nWhere do you want to begin?\n";
```

```
            std::cout<<"B. At the beginning?\nM. At the middle?";
            std::cout<<"\nE. At the end?\n\n";
            begin = 'M';

            switch(begin)
            {
                case('B'): std::cout<<"Once upon a time there was a wolf.\n";
                case('M'): std::cout<<"The wolf hurt his leg.\n";
                case('E'): std::cout<<"The wolf lived happily everafter\n";
            }
            return 0;
        }
```

```
In [9]: int menuItem = 1;
        switch(menuItem)
            {
                case(1): std::cout<<"Skiing?! Sounds dangerous!\n";
                         break;
                case(2): std::cout<<"Sledding?! Sounds like work!\n";
                         break;
                case(3): std::cout<<"Sitting by the fire?! Sounds warm!\n";
                         break;
                case(4): std::cout<<"Hot chocolate?! Yum!\n";
                         break;
                default: std::cout<<"Enter a valid menu item";
            }
```

```
Skiing?! Sounds dangerous!
```

```
In [10]: char begin;
         begin = 'M';

         switch(begin)
         {
             case('B'): std::cout<<"Once upon a time there was a wolf.\n";
             case('M'): std::cout<<"The wolf hurt his leg.\n";
             case('E'): std::cout<<"The wolf lived happily everafter\n";
         }
```

```
The wolf hurt his leg.
The wolf lived happily everafter
```

Recall, **break** statements are optional. In this first example, notice, there are no break statements between the cases. This means when a menu item is selected the case statement associated with that choice is executed, as well as each case statement that comes after it.

```
In [11]: #include <iostream>
```

```
int holiday_package()
{
    char menuItem;
    std::cout<<"Choose your holiday package:\n";
    std::cout<<"L: luxury package\nS: standard package\n";
    std::cout<<"B: basic package ";

    std::cin>>menuItem;
    std::cout<<menuItem<<"\n";
    std::cout<<"The "<<menuItem<<" package includes:\n";

    switch(menuItem)
    {
        case 'L':
        {
            std::cout<<"\tSpa Day\n";
            std::cout<<"\tSailboat Tour\n";
        }
        case 'S':
        {
            std::cout<<"\tCity Tour\n";
            std::cout<<"\tComplimentary Happy Hour\n";
        }
        case 'B':
        {
            std::cout<<"\tAirport Transfers\n";
            std::cout<<"\tComplimentary Breakfast\n";
            break;
        }
        default:
            std::cout<<"Please select the L,S,B package.\n";
    }
    return 0;
}
```

Now I would like you to do a switch statement with breaks between the cases. Create a program that asks the user for two float numbers. Then asks the user if they would like to:

- add the numbers
- subtract the numbers
- multiply the numbers
- divide the numbers

The program should then print the numbers with the chosen operation and the solution. For example:

```
The user enters: 4 5 +
The program will output: 4 + 5 = 9
```

In [12]:
```
/*Now I would like you to do a switch statement with breaks
**between the cases. Create a program that asks the user for
**two float numbers. Then asks the user if they would like to:
**add the numbers, subtract the numbers, multiply the numbers,
**divide the numbers.
**The program should then print the numbers with the chosen
**operation and the solution.
*/

#include <iostream>

int calculations()
{
    float in1, in2;
    std::cout<<"Enter two numbers:\n";
    std::cin>>in1;
    std::cin>>in2;
    std::cout<<"Enter the operation '+','-','*','/':\n";
    char op;
    std::cin>>op;
    switch(op)
    {
            case '+':
            {
                std::cout<<"The user enters: "<<in1<<"\t"<<in2<<"\t + \n";
                std::cout<<"The program will output: "<<"in1"<<" + "<<in2<<" = "<<in1 +
            }
            case '-':
            {
                std::cout<<"The user enters: "<<in1<<"\t"<<in2<<"\t - \n";
                std::cout<<"The program will output: "<<"in1"<<" - "<<in2<<" = "<<in1 -
            }
            case '*':
            {
                std::cout<<"The user enters: "<<in1<<"\t"<<in2<<"\t * \n";
                std::cout<<"The program will output: "<<"in1"<<" * "<<in2<<" = "<<in1 *
            }
            case '/':
            {
                std::cout<<"The user enters: "<<in1<<"\t"<<in2<<"\t / \n";
                std::cout<<"The program will output: "<<"in1"<<" / "<<in2<<" = "<<in1 /
            }
    }
    return 0;
}
```

4 Loops

We are using loops when we want to conduct an iterative process.

4.1 For Loops

for loops in C++ have the following form:

```
for ( declaration : range ) statement;
for (initialization; condition; increase) statement;
```

In [13]:
```
// Basic 10-element integer array.
int x[10] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

for( int i : x )
{
    std::cout<<"i = " << i << "\n";
}
```

```
i = 1
i = 2
i = 3
i = 4
i = 5
i = 6
i = 7
i = 8
i = 9
i = 10
```

In [14]:
```
for(int i=0; i< 11;i++)
    {
        std::cout<<"i = "<<i<<"\n";
    }
```

```
i = 0
i = 1
i = 2
i = 3
i = 4
i = 5
i = 6
i = 7
i = 8
i = 9
i = 10
```

Exercise:

1. Write a program that asks a user for five numbers.
2. Print out the sum and average of the five numbers.

```
In [20]: int input_sum()
         {
             float sum = 0;
             float average = 0;
             float num;
             for(int i = 1; i<6; i++)
             {
                 std::cout<<"please input the "<<i<<"th number\n";
                 std::cin>>num;
                 sum += num;
             }
             average = sum/5.0;
             std::cout<<"The sum of the numbers is: "<<sum<<"\n";
             std::cout<<"The average of the numbers is: "<<average;
         }
```

```
input_line_33:1:5: error: redefinition of 'input_sum'
int input_sum()
    ^input_line_32:1:5: note: previous definition is here
int input_sum()
    ^


        Interpreter Error:
```

4.2 While Loop

C++ has two forms of the while loop:

`while loops`

`do .. while loops`

C++ while loops syntax is:

4.2.1 while loops

`while(condition) {  statements; }`

If the condition is true, the statements within the curly brackets are executed. If the condition is false, the statements are not executed.

```
In [21]: int entry = 0;

         //Condition is true
         while(entry <=5)
         {
             std::cout<<"incrementing entry = "<<entry<<"\n";
             entry++;
         }
```

```
incrementing entry = 0
incrementing entry = 1
incrementing entry = 2
incrementing entry = 3
incrementing entry = 4
incrementing entry = 5
```

```
In [22]: //Condition is false

         while(entry < 5)
         {
             std::cout<<"decrementing entry = "<<entry<<"\n";
             entry--;
         }
```

4.2.2 do-while loops

```
do {      statements; }while(condition );
```

The statements are executed the first time through the loop. Then the condition is checked. This is slightly different than the while loop. Which, you may recall, required the condition to be true **BEFORE** entering the loop for the first time.

```
In [23]: int count = 0;
         do
         {
             std::cout<<"Count = "<<count<<"\n";
             count++;
         }while(count < 5);
```

```
Count = 0
Count = 1
Count = 2
Count = 3
Count = 4
```

```
In [24]: int otherCount = 6;
         //This do..while loop will execute once
         do
         {
             std::cout<<"othercount = "<<otherCount<<"\n";
             otherCount++;
         }while(otherCount < 5);
```

```
othercount = 6
```

In the programming quiz, use a while loop to prompt the user to guess a target number. Tell the user if the guess is too high or too low. The user enters 'q' or guesses the target number to end the program.

In [25]:
```
/*Goal: In the programming quiz, use a while loop to prompt
**the user to guess a target number.
**Tell the user if the guess is too high or too low.
**The user enters -1  or guesses the target number to end
**the program.
*/

int gues_game()
{
    //use 55 as the number to be guessed
    int target = 100;

    int guess = -1;
    std::cout<<"Guess a number between 0 and 1000: ";
    while((guess != target))
    {
        std::cin>>guess;
        std::cout<<guess<<"\n";
        if(guess > target)
        {
            std::cout<<"Your guess is too high\n";
        }
        else if(guess < target)
        {
            std::cout<<"Your guess is too low\n";
        }
        else
            std::cout<<"You guessed the target!\n";
        if(guess == -1)
        {
            std::cout<<"Goodbye!";
            break;
        }
    }


    return 0;
}
```

An alternative solution to this problem is using a random number generator.

In [26]:
```
#include <iostream>
#include<sstream>
#include <time.h> //added for the random number generator seed
#include <cstdlib>//added to use the rand function

int random_guess()
{
```

```
    int target;
    std::string userString;
    int guess = -1;

    srand(time(NULL));  //set the seed for the random number generator
    target = rand() %1000 + 1; //generate the 'random' number

    while(guess != target)
    {
        std::cout<<"Guess a number between 0 and 1000: ";
        std::getline (std::cin,userString);
        //convert to an int
        std::stringstream(userString) >> guess;
        std::cout<<userString<<"\n";
        if(guess > target)
            std::cout<<"Your guess is too high\n";
        else if(guess < target)
            std::cout<<"Your guess is too low\n";
        else
            std::cout<<"You guessed the target!\n";

        //Note I had to use double quotes around "q"
        //because it is a string
        if(userString == "q")
        {
            std::cout<<"good bye!";
            std::cout<<"The number was "<<target<<"\n";
            break;
        }
    }
    return 0;
}
```

4.3 Infinite Loops

But why?

Sometimes it is desired to create an infinite loop. For example, in embedded systems a infinite loop is often used for the main task.

To create an infinite loop using a for loop:

```
for( ; ;) {      std::cout<<"This for loop will run forever\n"; }
```

To create an infinite loop using a while loop:

```
while(1) {      std::cout<<"This while loop will run forever\n"; }
```

4.4 Exit a Loop

To exit a loop before it completes its normal sequence, we need to use control statements.

The two most commonly used are:

- break
- continue

The break statement:
The break statement will **END** the loop and begin executing the first statement that comes **AFTER** the end of the loop.
The continue statement:
The continue statement will **FORCE** the **NEXT** iteration to be executed.

In [27]:
```
/*Goal: understand the break and conitnue statements*/

#include <iostream>


int break_continue()
{
    int a = 0;
    while(a < 5)
    {
        std::cout<<"a = "<<a<<"\n";
        a++;
        if(a == 3)
            break;
    }
    std::cout<<"The first statement after the first while loop\n\n";


    while(a < 15)
    {
        a++;
        if(a == 10)
        {
            std::cout<<"\tWhen a=10, go back to the top of the loop";
            std::cout<<"\n\tThis means a=10 is skipped.\n";
            continue;
        }
        std::cout<<"After continue a = "<<a<<"\n";
    }
    return 0;
}
```

In [28]:
```
int a_br = 0;
while(a_br < 5)
{
    std::cout<<"a_br = "<<a_br<<"\n";
    a_br++;
    if(a_br == 3){
        break;
```

```
            }
        }
        std::cout<<"The first statement after the first while loop\n\n";
```

```
a_br = 0
a_br = 1
a_br = 2
The first statement after the first while loop
```

```
In [29]: while(a_br < 15)
         {
             a_br++;
             if(a_br == 10)
             {
                 std::cout<<"\tWhen a_br=10, go back to the top of the loop";
                 std::cout<<"\n\tThis means a_br=10 is skipped.\n";
                 continue;
             }
             std::cout<<"After continue a_br = "<<a_br<<"\n";
         }
```

```
After continue a_br = 4
After continue a_br = 5
After continue a_br = 6
After continue a_br = 7
After continue a_br = 8
After continue a_br = 9
        When a_br=10, go back to the top of the loop
        This means a_br=10 is skipped.
After continue a_br = 11
After continue a_br = 12
After continue a_br = 13
After continue a_br = 14
After continue a_br = 15
```