CS101 - Additional Help Material

Variable Declarations and Assignments:

```
int hello; //declares a variable hello
int Hello; //declares another variable Hello
Int hello; //error, Int is not a C++ type
Double amount; //error, Double is not a type
```

Example of illegal variable names

```
int number of cakes; // Has spaces
int 1number; // Begins with a number
int discount%; // Contains a symbol
double int; // Variable name cannot be a keyword, in this case int
int string; //  Although not a keyword, it will not allow us to
            //declare and use string variables
```

Some Valid Declaration with Initialization for int

```
int number = 14;
int year = 1996;
int age = 45;
```

Invalid Assignments for the above:

```
number = "14";
year = '2011';
age = "thirty";
age = 34.5;
```

Valid Declaration and assignments for double

```
double pi;
double e;
pi = 3.1415926535; // Valid
e = 2.71828; // valid
```

Invalid Assignments for the above:

```
pi = "3.141";
pi = ' 3.141 ';
```

Valid Declaration and assignments for char

```
char c;
char newline;
char code;
c = 'c';
newline = '\n';
code = 65; //max number is 127
```

Invalid Assignments for the above:

```
code = 456;
newline = "\n"; // double quotes is a string, not a char
```

Valid Declaration and assignments for bool

```
bool reply;
bool answer;

answer = true;
answer = false;
reply = 0;
reply = 1;
```

Invalid Assignments for the above:

```
answer = “true”;
reply = ‘0’; //value becomes true
```

Expressions Involving Arithmetic Operators

Integer Division

5 / 2 = 2
4 / 2 = 2
10/ 3 = 3
11 / 2 = 5

Floating Point division

5/2.0 = 2.5
4 / 2.0 = 2.0 // on the screen you see 2 printed
10.0/ 3 = 3.333333…
11.0 / 2.0 = 5.5

What makes this a decimal division? – Either the dividend or divisor MUST be a decimal.
Same rules apply to Addition, Subtraction and Multiplication.
If both values are of type int, the result is int.
If either is of type double, the result is double

Expression Evaluation:

The following is an assignment to Y of the value of ($ax^2 + bx + c$)

```
y = ( a * x * x ) + ( b * x ) + c;
```

If x has the value 5, and a=2, b=3 and c=-7 then y becomes = 50 + 15 - 7 = 58

ORDER OF PRECEDENCE

```
#include <simplecpp>

main_program {
   int a = 20;
   int b = 10;
   int c = 15;
   int d = 5;
   int e;

   e = (a + b) * c / d;       // ( 30 * 15 ) / 5
   cout << "Value of (a + b) * c / d is :" << e << endl ;

   e = ((a + b) * c) / d;    // (30 * 15 ) / 5
   cout << "Value of ((a + b) * c) / d is  :" << e << endl ;

   e = (a + b) * (c / d);   // (30) * (15/5)
   cout << "Value of (a + b) * (c / d) is  :" << e << endl ;

   e = a + (b * c) / d;     //  20 + (150/5)
   cout << "Value of a + (b * c) / d is  :" << e << endl ;

}
```

The output of the program is:

```
Value of (a + b) * c / d is :90
Value of ((a + b) * c) / d is  :90
Value of (a + b) * (c / d) is  :90
Value of a + (b * c) / d is  :50
```

Relational Operators

Reversing the order of the pair of symbols in the operators !=, >= and <= (by writing them as =!, => and =<, respectively) is normally a syntax error.

```
// Comparing integers using if statements, relational operators
// and equality operators.
#include <iostream> // allows program to perform input and output

using std::cout; // program uses cout
using std::cin; // program uses cin
using std::endl; // program uses endl

// function main begins program execution
int main()
{
   int number1; // first integer to compare
   int number2; // second integer to compare

   cout << "Enter two integers to compare: "; // prompt user for data
   cin >> number1 >> number2; // read two integers from user

   if ( number1 == number2 )
     cout << number1 << " == " << number2 << endl;

   if ( number1 != number2 )
      cout << number1 << " != " << number2 << endl;

   if ( number1 < number2 )
      cout << number1 << " < " << number2 << endl;

   if ( number1 > number2 )
      cout << number1 << " > " << number2 << endl;

   if ( number1 <= number2 )
      cout << number1 << " <= " << number2 << endl;

   if ( number1 >= number2 )
      cout << number1 << " >= " << number2 << endl;
} // end function main
```

Combining Relational Operators - Conjunction, Disjunction and Negation (aka Boolean Operators)

Arguments of a boolean operator can be one of 3 things:

1. bool variables
2. true, false or a number
3. a relational expression

Example 1 - Simple Usage and Precedence

```
#include <simplecpp>
main_program {
    bool a = false;
    bool b = true;
    bool c = 1;
    if (a && b) { // a AND b
        cout << "a && b is true" << endl;
    }
    if (a || b) { // a OR b
        cout << "a || b is true" << endl;
    }
    if (!a) { // NOT a
        cout << "a is false" << endl;
    }
    if (a && b || c) { // Equivalent to (a && b) || c
        cout << "a && b || c is true" << endl;
    }
    if (a && (b || c)) { // Equivalent to a && (b || c)
        cout << "a && (b || c) is true" << endl;
    }
    if (a && b && c) { // Equivalent to (a && b) && c, same holds true for OR
        cout << "a && b && c is true" << endl;
    }
    if (!a || b) { // Equivalent to (!a) || b
        cout << "!a || b is true" << endl;
    }
    if (!a && c) { // Equivalent to (!a) && c
```

```
        cout << "!a && c is true" << endl;
    }
    if (!(a || b)) { // Equivalent to !(a || b)
        cout << "!(a || b) is true" << endl;
    }
    if (!(a && c)) { // Equivalent to !(a && c)
        cout << "!(a && c) is true" << endl;
    }
}
```

Example 2 - Relational Operators + Boolean Variables Usage, Precedence

Precedence order : Boolean NOT > Arithmetic > Relational > Boolean AND > Boolean OR

```
#include <simplecpp>
main_program {
    bool a = false;
    bool b = true;
    int c = 5;
    float d = 7.5;
    if (d > 6.0 || a) {
        // Equivalent to (d > 6) || b
        cout << "d > 6.0 || a is true" << endl;
    }
    if (d - 2.0 > 6.0 && b) {
        // Equivalent to ((d - 2.0) > 6.0) && b i.e. (5.5 > 6.0) && b
        cout << "d - 2.0 > 6.0 && b is endl" << endl;
    }
    if (!c - 2 > 0 && b) {
        // Equivalent to ((!c) - 2 > 0) && b
        // !c evaluates to 0 if c != 0 and to 1 if c == 0
        cout << "!c - 2 > 0 && b is true" << endl;
    }
}
```

IF Statement

Placing a semicolon immediately after the right parenthesis after the condition in an if statement is often a logic error (although not a syntax error). The semicolon causes the body of the if statement to be empty, so the if statement performs no action, regardless of whether or not its condition is true. Worse yet, the original body statement of the if statement now becomes a statement in sequence with the if statement and always executes, often causing the program to produce incorrect results.

Indent the statement(s) in the body of an if statement to enhance readability.

1. **Use of semicolon.**

```
int age = 10;
if (age > 18)            // no semicolon here
{
        cout<<"adult.";
}
```

2.

Unlike **if** and **else if**, an **else** statement is not used to do condition checking. The purpose of **else** is to execute alternative code in the situation where the **if**, and **else if** conditions return false.

Example-

```
if ( a < b){
   // do something here;
}
else if (a > b){
   //do this other thing;
}
else {                          // else (a == b)
  // do something different;
}
```

3.

Use braces for multiple statements under **if**. Otherwise, only first sentence of them will be under body of **if.**

Example-

```
int age = 22;
if ( age < 18 )
        cout<<"below 18";
        cout<<"not an adult.";
```

This will behave like-

```
int age = 22;
if ( age < 18 )
{
        cout<<”below 18”;
}
cout<<”not an adult.”;
```

o/p: not an adult.

4. If you want to check whether two values are equal or not, make sure you have written relational operator **==**, not the assignment operator **=**. Otherwise, it will try to do the assignment job, and return TRUE after assigning successfully.

Example-

```
int a, b;
a = 4;
b = 3;
if ( a = b )
        cout<<”a and b are equal.”;
```

o/p: a and b are equal.

5. Semicolon after if

```
Int a=0, b=1;
if(a == b){
        cout<<”A equal to B”;
};
else{
        cout<<”A not equal B”;
}
```

There should not be any single statement between **if** and **else,** in above program, there is not a single statement but, a semicolon act as a statement.

Giving multiple semicolon after a statement is fine. You can write “int a;;;;;;; (any number of semicolon),” also you can write,

```
if(condition){
        //some statements
};;;;;;;
```

But if there is **else** with **if**, then giving semicolon after **if** is incorrect and will give error.

WHILE LOOP

1. Using uninitialized variable in condition

```
int count;
while (count < 100) {
        cout << count;
        count++;
}
```

Why doesn't my program enter the while loop?

In C++, by default, variables are not initialized to zero, the variable "count" was assigned a memory location with garbage data. Garbage data can be anything, that may violates your condition or can satisfy the condition, for better practice, always initialize the variable with initial value (depends on your logic).

2. Using semicolon after while loop

```
Int a=0;
while(a++ < 10);
        cout<<a<<endl;
```

Why does it just output only 10, why not 1,2,3….9 ?

If you put semicolon after **while loop,** it means, your while loop doesn't have any body, i.e. your while loop is empty.

So, at **a = 10,** while loop will terminate and statement next to it i.e. **cout<<a<<endl;** will get executed.

For correct output, program should be,

```
Int a = 0;
while(a < 10) <-- ; removed
        cout<<a<<endl;
```

3. While loop should contain a statement which changes the value of variable that used in condition checking,

Example:

```
int a=0;
while(a < 10){
        cout<<a;
}
```

This while loop will execute infinite number of times.

```
int a = 0;
while(a < 10){
	cout<<a;
	a++;   <-- added
}
```

4. Using single equal to check condition

```
char again = ‘Y’;
while(again = ‘Y’){
	cout<<”Continue?(Y/N)”;
	cin>>again;
}
```

This program runs infinite number of times because, = is assignment operator and it always assigns ‘Y’ to again, and result of this statement is **true**. Use == instead.

```
char again = ‘Y’;
while(again == ‘Y’){
	cout<<”Continue? (Y/N)”;
	cin>>again;
}
```

5. **DO-WHILE LOOP**

Don’t forget to put semicolon after while in do-while loop,

Example:

```
do{
	//statements
}while(condition); <- important
```

FOR LOOP

1. **Use of Semicolon:**

```
for(i=1;i<=10;i++)          // no semicolon after brackets
{
        sum1=sum1+i;
        sum2=sum2+i*i;
}
```

2. **Two semicolons within brackets:**

Code before first semicolon will be treated as initialization, code in between two semicolon is condition, code after second semicolon is increment/decrement. See **flow of for loop** for details.

```
for ( init; condition; increment )
{
        // statement(s);
}
```

3. Always keep in mind- how many times the loop is executing.

```
for( int a = 1; a < 20; a++ )       // this loop will be executed 19 times.
{
         cout << "value of a: ";
 }
```

4. Take care of precedence of operators. Use brackets in this cases.

ARRAYS

1. **Difference between variables declaration and array declaration,**

int arr; <- here arr is variable

and

Int arr[10]; <- arr is an array of size 10.

2. **Give size of an array while declaration,**

```
int arr[];          // gives error,
int arr[10];        // allocates the memory for 10 integers with name arr
```

3. **Arrays are initialized with zero by default.**

```
int arr[10];
cout<<”Value of arr[2] is: ”<< arr[2];
```

o/p: Value of arr[2] is: 0

4. **Overstepping array boundaries**

```
int arr[10];
int i=0;
for(i=0;i<=10;i++){
        cout<<a[i]<<” “;
}
```

Above program will compile successfully, but it will give **unpredictable value, which will be a garbage value.**

FUNCTIONS

1. Undeclared Function:

```
int main(){
        void fun();
}
void fun(){
        // functio body
}
```

You will get an error **“fun : identifier not found ”**, compiler does not know what **fun** is.

```
void fun();     // function declaration also known as function prototype
int main(){
        fun();  // function calling
}

void fun(){     //function definition
        //function body
}
```

2. Function inside function is not allowed

```
void fun();
int main(){
        void fun(){
                cout << ”Allowed”;
        }
}
```

Here, you are defining function inside function, which is not allowed and produces error. Instead call function from another function and define function outside all functions.

3. Return Type of function

```
void fun();
int main(){
        cout << fun();
}
void fun(){
        return 5;
}
```

Here, return type of **fun()** is void i.e. fun() will not return any type of value and you are trying to print this value, which will give error. For this, return type of the function should be anything except **void.**

```
int fun();
int main(){
        cout << fun();
}
int fun(){
        return 5;
}
```

Output will be 5.

4. Extra Semicolon

```
int fun();
int main(){
        fun();
}

int fun();          // ; will produce error
{
        return 0;
}
```

Semicolons go at the end of complete statements. The following are not complete statements.

5. Different return types in function declaration and definition

```
int fun();
int main() {
   fun();
}
void fun(){       //return type is different from return type in prototype
   // function body
}
```

Return type of a function must be same as in declaration. Here, **fun()** has **int** return type but in definition it is written as **void** which is mismatched.