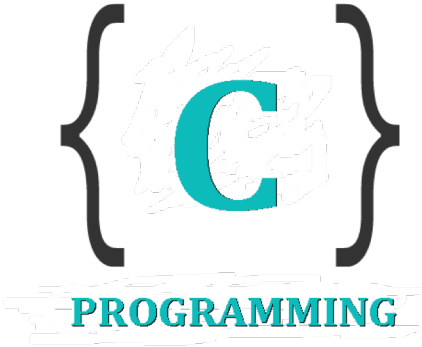

IT 105: Introduction to Programming

Dr. Manish Khare
Dr. Saurabh Tiwari

Lecture 5

Desirable Programming Style

- Clarity
  - The program should be clearly written.
  - It should be easy to follow the program logic.
- Meaningful variable names
  - Make variable/constant names meaningful to enhance program clarity.
    - ‘area’ instead of ‘a’
    - ‘radius’ instead of ‘r’
- Program documentation
  - Insert comments in the program to make it easy to understand.
  - Never use too many comments.
- Program indentation
  - Use proper indentation.
  - Structure of the program should be immediately visible.

Indentation Example #1 :: Bad Style

```
#include <stdio.h>
#define    PI    3.1415926
/* Compute the area of a circle */
main()
{
float   radius, area;
float   myfunc (float radius);
scanf ("%f", &radius);
area = myfunc (radius);
printf ("\n Area is %f \n", area);
}
```

```
float   myfunc (float r)
{
float   a;
a = PI * r * r;
return (a);      /* return result */
}
```

Indentation Example #1 :: Good Style

```
#include <stdio.h>
#define   PI   3.1415926
/* Compute the area of a circle */

main()
   {
      float   radius, area;
      float   myfunc (float radius);

      scanf ("%f", &radius);
      area = myfunc (radius);
      printf ("\n Area is %f \n", area);
   }
```

```
float   myfunc (float r)
   {
      float   a;
      a = PI * r * r;
      return (a);      /* return result */
   }
```

Indentation Example #2 :: Bad Style

```
#include <stdio.h>

/* FIND THE LARGEST OF THREE NUMBERS */

main()
{
int   a, b, c;
scanf (“%d %d %d”, &a, &b, &c);
if  ((a>b) && (a>c))   /* Composite condition check */
printf (“\n Largest is %d”, a);
else
if  (b>c)     /* Simple condition check */
printf (“\n Largest is %d”, b);
else
printf (“\n Largest is %d”, c);
}
```

Indentation Example #2 :: Good Style

```
#include <stdio.h>

/* FIND THE LARGEST OF THREE NUMBERS */

main()
    {
        int   a, b, c;
        scanf ("%d %d %d", &a, &b, &c);
        if  ((a>b) && (a>c))                    /* Composite condition check */
            printf ("\n Largest is %d", a);
        else
            if  (b>c)                           /* Simple condition check */
                printf ("\n Largest is %d", b);
            else
                printf ("\n Largest is %d", c);
    }
```

C instruction

- I hope before continuing to this lecture you must have written the basic printf() and scanf() programs. If you didn't then I strongly recommend you to do it.

- A program is nothing but a set of instruction. The program behaves as per the instructions that we give in it.

- Different instructions help us to achieve different task in a program

Type of Instruction

- There are basically three types of instruction in C
  - **Type declaration instruction** – This instruction is used to declare the type of variables used in a C program.
  - **Arithmetic instruction** – This instruction is used to perform arithmetic operations on constants and variables.
  - **Control instruction** – This instruction is used to control the sequence of execution of various statements in a C program

Type Declaration instruction

- This instruction is used to declare the type of variables being used in the program.
- Any variable used in the program must be declared before using it in any statement. T
- he type declaration statement is written at the beginning of main( ) function.
- Ex.:
  - int bas ;
  - float rs, grosssal ;
  - char name, code ;

- There are several subtle variations of the type declaration instruction
  - While declaring the type of variable we can also initialize it as shown below.
    - int i = 10, j = 25 ;
    - float a = 1.5, b = 1.99 + 2.4 * 1.44 ;
  - The order in which we define the variables is sometimes important sometimes not.
    - For example,
      - int i = 10, j = 25 ;
      - is same as int j = 25, j = 10
      - However, float a = 1.5, b = a + 3.1 ; is alright,
      - but float b = a + 3.1, a = 1.5 is not;
      - This is because here we are trying to use **a** even before defining it.

- Another interesting point while declaring variables is
- Below instruction will work
  - int a,b,c,d;
  - a=b=c=10;
- However, the following statement would not work.
  - int a=b=c=d=10 ;
- We are trying to use b (to assign a) before defining it.

- I hope now you must have understand the importance of type declaration instruction.
- So we can only use variables after defining them. And these definition should be written at the starting of main() body.
- We cannot define variables anywhere else in the program. If you do so, it will result in an error.

Arithmetic Instruction

- C arithmetic instruction consists of a variable name on the left hand side of = and variable names & constants on the right hand side of =

- The variables and constants appearing on the right hand side of = are connected by arithmetic operators like +, -, *, and /
- Ex.:
  - int ad ;
  - float kot, deta, alpha, beta, gamma ;
  - ad = 3200 ;
  - kot = 0.0056 ;
  - deta = alpha * beta / gamma + 3.2 * 2 / 5

- Here,
  - *, /, -, + are the arithmetic operators.
  - = is the assignment operator.
  - 2, 5 and 3200 are integer constants.
  - 3.2 and 0.0056 are real constants.
  - ad is an integer variable.
  - kot, deta, alpha, beta, gamma are real variables.

- The variables and constants together are called ‘operands’ that are operated upon by the ‘arithmetic operators’ and the result is assigned, using the assignment operator, to the variable on lefthand side.

- A C arithmetic statement could be of three types. These are as follows:
  - Integer mode arithmetic statement
  - Real mode arithmetic statement
  - Mixed mode arithmetic statement

- Integer mode arithmetic statement –
  - This is an arithmetic statement in which all operands are either integer variables or integer constants. Ex.:
    - int i, king, issac, noteit ;
    - i = i + 1 ;
    - king = issac * 234 + noteit - 7689 ;

- Real mode arithmetic statement –
  - This is an arithmetic statement in which all operands are either real constants or real variables. Ex.:
    - float qbee, antink, si, prin, anoy, roi ;
    - qbee = antink + 23.123 / 4.5 * 0.3442 ;
    - si = prin * anoy * roi / 100.0 ;

- Mixed mode arithmetic statement –
  - This is an arithmetic statement in which some of the operands are integers and some of the operands are real. Ex.:
    - float si, prin, anoy, roi, avg ;
    - int a, b, c, num ;
    - si = prin * anoy * roi / 100.0 ;
    - avg = (a + b + c + num) / 4 ;

- It is very important to understand how the execution of an arithmetic statement takes place.

- Firstly, the right hand side is evaluated using constants and the numerical values stored in the variable names. This value is then assigned to the variable on the left-hand side.

- Though Arithmetic instructions look simple to use one often commits mistakes in writing them.

- Let us take a closer look at these statements. Note the these points carefully.

- C allows only one variable on left-hand side of =. That is, **z = k * l** is legal, whereas **k * l = z** is illegal.
- In addition to the division operator C also provides a modular division operator.
  - This operator returns the remainder on dividing one integer with another.
    - Thus the expression 10 / 2 yields 5, whereas, 10 % 2 yields 0.
  - Note that the modulus operator (%) cannot be applied on a float.
  - Also note that on using % the sign of the remainder is always same as the sign of the numerator.
    - Thus –5 % 2 yields –1, whereas, 5 % -2 yields 1.

- An arithmetic instruction is often used for storing character constants in character variables.
  - char a, b, d ;
  - a = 'F' ; b = 'G' ; d = '+' ;

- When we do this the ASCII values of the characters are stored in the variables.
- ASCII values are used to represent any character in memory.
- The ASCII values of ‘F’ and ‘G’ are 70 and 71

- No operator is assumed to be present. It must be written explicitly.
- In the following example, the multiplication operator after b must be explicitly written.
  - a = c.d.b(xy) usual arithmetic statement
  - b = c * d * b * (x * y) C statement

- Unlike other high level languages, there is no operator for performing exponentiation operation. Thus following statements are invalid.
  - a = 3 ** 2 ;
  - b = 3 ^ 2 ;
- If we want to do the exponentiation we can get it done this way:

```
#include <math.h>
main( )
{
int a ;
a = pow ( 3, 2 ) ;
printf ( “%d”, a ) ;
}
```

- Here pow( ) function is a standard library function.
- It is being used to raise 3 to the power of 2.
- #include is a preprocessor directive.
- It is being used here to ensure that the pow( ) function works correctly.
- We would learn more about standard library functions

Integer and Float Conversions

- In order to effectively develop C programs, it will be necessary to understand the rules that are used for the implicit conversion of floating point and integer values in C.

- These are mentioned below
  - An arithmetic operation between an integer and integer always yields an integer result.
  - An operation between a real and real always yields a real result.
  - An operation between an integer and real always yields a real result.
  - In this operation the integer is first promoted to a real and then the operation is performed. Hence the result is real.

Operation	Result	Operation	Result
5 / 2	2	2 / 5	0
5.0 / 2	2.5	2.0 / 5	0.4
5 / 2.0	2.5	2 / 5.0	0.4
5.0 / 2.0	2.5	2.0 / 5.0	0.4

Type Conversion in Assignments

- It may so happen that the type of the expression and the type of the variable on the left-hand side of the assignment operator may not be same.
- In such a case the value of the expression is promoted or demoted depending on the type of the variable on left-hand side of =.

- Consider the following assignment statements.
  - int i ;
  - float b ;
  - i = 3.5 ;
  - b = 30 ;
- Here in the first assignment statement though the expression's value is a float (3.5) it cannot be stored in i since it is an int.
- In such a case the float is demoted to an int and then its value is stored. Hence what gets stored in i is 3.
- Exactly opposite happens in the next statement. Here, 30 is promoted to 30.000000 and then stored in b, since b being a float variable cannot hold anything except a float value.

- if a complex expression occurs, still the same rules apply.
- For example, consider the following program fragment.
  - float a, b, c ;
  - int s ;
  - s = a * b * c / 100 + 32 / 4 - 3 * 1.1 ;
- Here, in the assignment statement some operands are ints whereas others are floats.
- As we know, during evaluation of the expression the ints would be promoted to floats and the result of the expression would be a float.
- But when this float value is assigned to s it is again demoted to an int and then stored in s.

- Observe the results of the arithmetic statements shown in below. It has been assumed that **k** is an integer variable and **a** is a real variable.

Arithmetic Instruction	Result	Arithmetic Instruction	Result
k = 2 / 9	0	a = 2 / 9	0.0
k = 2.0 / 9	0	a = 2.0 / 9	0.2222
k = 2 / 9.0	0	a = 2 / 9.0	0.2222
k = 2.0 / 9.0	0	a = 2.0 / 9.0	0.2222
k = 9 / 2	4	a = 9 / 2	4.0
k = 9.0 / 2	4	a = 9.0 / 2	4.5
k = 9 / 2.0	4	a = 9 / 2.0	4.5
k = 9.0 / 2.0	4	a = 9.0 / 2.0	4.5

Hierarchy of Operations

- While executing an arithmetic statement, which has two or more operators, we may have some problems as to how exactly does it get executed.
- For example, does the expression 2 * x - 3 * y correspond to (2x)-(3y) or to 2(x-3y)?
- Similarly, does A / B * C correspond to A / (B * C) or to (A / B) * C?
- To answer these questions satisfactorily one has to understand the 'hierarchy' of operations.
- The priority or precedence in which the operations in an arithmetic statement are performed is called the hierarchy of operations

Priority	Operators	Description
1st	* / %	multiplication, division, modular division
2nd	+ -	addition, subtraction
3rd	=	assignment

- Within parentheses the same hierarchy as mentioned in figure is operative.
- If there are more than one set of parentheses, the operations within the innermost parentheses would be performed first, followed by the operations within the second innermost pair and so on

- Determine the hierarchy of operations and evaluate the following expression:

  i = 2 * 3 / 4 + 4 / 4 + 8 - 2 + 5 / 8

Stepwise evaluation of this expression is shown below:

i = 2 * 3 / 4 + 4 / 4 + 8 - 2 + 5 / 8	
i = 6 / 4 + 4 / 4 + 8 - 2 + 5 / 8	operation: *
i = 1 + 4 / 4 + 8 - 2 + 5 / 8	operation: /
i = 1 + 1+ 8 - 2 + 5 / 8	operation: /
i = 1 + 1 + 8 - 2 + 0	operation: /
i = 2 + 8 - 2 + 0	operation: +
i = 10 - 2 + 0	operation: +
i = 8 + 0	operation : -
i = 8	operation: +

- Determine the hierarchy of operations and evaluate the following expression:

kk = 3 / 2 * 4 + 3 / 8 + 3

Stepwise evaluation of this expression is shown below:

kk = 3 / 2 * 4 + 3 / 8 + 3	
kk = 1 * 4 + 3 / 8 + 3	operation: /
kk = 4 + 3 / 8 + 3	operation: *
kk = 4 + 0 + 3	operation: /
kk = 4 + 3	operation: +
kk = 7	operation: +

- So far we have seen how the computer evaluates an arithmetic statement written in C.
- But our knowledge would be incomplete unless we know how to convert a general arithmetic statement to a C statement.
- C can handle any complex expression with ease.

Algebric Expression	C Expression
$a \times b - c \times d$	a * b – c * d
$(m + n)\ (a + b)$	(m + n) * (a + b)
$3x2 + 2x + 5$	3 * x * x + 2 * x + 5
$\frac{a+b+c}{d+e}$	(a + b + c) / (d + e)
$\left[\frac{2BY}{d+1} - \frac{x}{3(z+y)}\right]$	2 * b * y / (d + 1) – x / 3 * (z + y)

Associativity of Operators

- When an expression contains two operators of equal priority the tie between them is settled using the associativity of the operators.

- Associativity can be of two types—Left to Right or Right to Left.

- Lets understand this with the help of example. Consider the expression: a = 3 / 2 * 5 ;

- Here there is a tie between operators of same priority, that is between / and *. This tie is settled using the associativity of / and *. But both enjoy Left to Right associativity. Therefore firstly / operation is done followed by *.

Control Instructions in C

- As the name suggests the ‘Control Instructions’ enable us to specify the order in which the various instructions in a program are to be executed by the computer.
- In other words the control instructions determine the ‘flow of control’ in a program.
- There are four types of control instructions in C.
  - Sequence Control Instruction
  - Selection or Decision Control Instruction
  - Repetition or Loop Control Instruction
  - Case Control Instruction

- The Sequence control instruction ensures that the instructions are executed in the same order in which they appear in the program.

- Decision and Case control instructions allow the computer to take a decision as to which instruction is to be executed next.

- The Loop control instruction helps computer to execute a group of statements repeatedly

Exercise

- Evaluate the following expressions and show their hierarchy
  - g = big / 2 + big * 4 / big - big + abc / 3 ;
    - (abc = 2.5, big = 2, assume g to be a float)
  - on = ink * act / 2 + 3 / 2 * act + 2 + tig ;
    - (ink = 4, act = 1, tig = 3.2, assume on to be an int)
  - s = qui * add / 4 - 6 / 2 + 2 / 3 * 6 / god ;
    - (qui = 4, add = 2, god = 2, assume s to be an int)
  - s = 1 / 3 * a / 4 - 6 / 2 + 2 / 3 * 6 / g ;
    - (a = 4, g = 3, assume s to be an int)

- Convert the following equations into corresponding C statements

(a) $$Z = \frac{8.8\,(a+b)\,2/c - 0.5 + 2a/(q+r)}{(a+b)*(1/m)}$$

(b) $$X = \frac{-b + (b*b) + 2\ \ 4ac}{2a}$$

(c) $$R = \frac{2v + 6.22\ \ (c+d)}{g+v}$$

(d) $$A = \frac{7.7b\,(xy+a)/c - 0.8 + 2b}{(x+a)\ \ (1/y)}$$

What will be output of the following program

```
# include <stdio.h>
main( )
{
    int i = 2, j = 3, k, l ;
    float a, b ;
    k = i / j * j ;
    l = j / i * i ;
    a = i / j * j ;
    b = j / i * i ;
    printf( "%d %d %f %f", k, l, a, b ) ;
}
```

- What will be output of the following program

```
# include <stdio.h>
main( )
{
    int a, b, c, d;
    a = 2 % 5;
    b = -2 % 5;
    c = 2 % -5;
    d = -2 % -5;
    printf( "%d %d %d %d", a, b, c, d) ;
}
```

- What will be output of the following program

```
# include <stdio.h>
main( )
{
      float a =5,  b = 2;
      int c, d;
      c = a % b;
      d = a / 2;
      printf( "%d \n", d) ;
}
```