Programming Fundamentals- I

Rao Muhammad Umer
Lecturer,
Web: raoumer.github.io
Department of Computer Science & IT,
The University of Lahore.

Qualification

- **MS Computer Science (2014-2016)**
  - Pakistan Institute of Engineering & Applied Sciences (PIEAS), Nilore, Islamabad, Pakistan.
- **BSc. Computer Systems Engineering (2010-2014)**
  - The Islamia University of Bahawalpur (IUB), Bahawalpur, Pakistan.
- Visit my personal website for more information about me on following link:

  raoumer.github.io

Administrative Stuff

- Course related stuff will be available soon on "**PIAZZA"** discussion forum
- We will use **MS Visual Studio 2010**
- I will provide you demo on **MS Visual Studio 2010**

Course Contents

- Basic components of a computer
- Programming languages
- Introduction to operating system
- Programming in C with examples and applications

Text Books

1. **The ‘C’ Programming Language**

   By: Kernighan and Ritchie

2. **Let Us ‘C’**

   By: Yashwant Kanitkar

Objective of the Course

- To learn computer fundamentals
- To learn C language.

Why Learn C?

- C is the base language of any other programming language.
- To be a Good Programmer, one must know fundamentals of C programming language.
- **Simulations** ----- lets us apply maths to the real world.

Let us Write First C Program

Data Type

- **Integer variables** (int)
  - For example:
    - Number of students
    - Number of tables
- **Floating point variables** (float)
  - For quantities which may contain decimal point such as distance, area, and temperature.
- **Character variables** (char)
- At this stage, we will focus only on **integers**.

Approach To Our First C Program

```
x = 5;
y = 7;
z = x + y;
```

x, y and z are integer variables

Our First C Program

```
#include <stdio.h>
#include “stdafx.h”
int main( )
{
        int x, y, z;
        x = 5;
        y = 7;
        z = x + y;
        printf(“%d", z);
        getchar();
        return 0;
}
```

Output of Program

12

Our First C Program

```
#include <stdio.h>
#include "stdafx.h“
int main( )
{
        int x, y, z;
        x = 5;
        y = 7;
        z = x + y;
        printf(“Sum is %d", z);
        getchar();
        return 0;
}
```

Output of Program

Sum is 12

Our First C Program

```
#include <stdio.h>
#include "stdafx.h“
int main( )
{
        int x, y, z, m;
        x = 5;
        y = 7;
        z = x + y;
        m = x -  y;
        printf("Sum is %d\n", z);
        printf("Difference is %d", m);
        getchar();
        return 0;
}
```

Output of Program

Sum is 12

Difference is -2

Escape Sequences

Sequence	Meaning
\a	Bell (alert)
\b	Backspace
\n	Newline
\t	Horizontal tab
\\	Backslash
\'	Single quote
\"	Double quotation

Escape Sequences

- The name reflects the fact that the backslash causes an **“ escape”** from the normal way characters are interpreted.
- In this case the n is interpreted **not as the character ‘n’** but as the new line character.

```
#include <stdio.h>
#include "stdafx.h“
int main( )
{
  int x, y, z, m;
  x = 5;
  y = 7;
  z = x + y;
  m = x - y;

  printf("Sum is %d\n Difference is %d", z, m);

  getchar();
  return 0;
}
```

What’s output?

Exercise

Try out various escape sequences in this program.

“Hello World”

```
//Most of the people write this program first
#include <stdio.h>
#include "stdafx.h“
int main ( )
{
        printf  ("Hello, World!\n");
        getchar();
        return  0;
}
```

More about printf()

- The printf function is used to output information (both data from variables and text) to standard output.
- It takes a format string and parameters for output.
- e.g. printf("The result is %d \n", b);
- printf("The result is %d and %d\n", a, b);
- printf("The result is %d and %d\n", a, b*6);

More about printf()

The format string contains:

- Literal text: is printed as is without variation
- Escaped sequences: special characters preceeded by \
- Conversion specifiers: % followed by a single character
  - Indicates (usually) that a variable is to be printed at this location in the output stream.
  - The variables to be printed must appear in the parameters to printf following the format string, in the order that they appear in the format string.

C doesn't care much about spaces

```
#include <stdio.h>
#include "stdafx.h"
int main ( )
{
printf ("Hello  World!\n");

return 0;
}
```

```
#include <stdio.h>
#include "stdafx.h"
int
main
(
)
{
printf
(
"Hello
World!\n"
)
;
return
0
;
}
```

Both of these programs are the same as the original as far as your compiler is concerned.

We SHOULD lay out our C program to make them look nice.

C doesn’t care much about spaces

- In the most general sense, **a statement is a part of your program that can be executed.**
- **An expression is a statement.**

```
a=a+1;
```

- **A function call is also a statement.**

```
printf("%d“, a);
```

- Other statements ……
- **C is a free form language,** so you may type the statements in any style you feel comfortable:

```
a=
a+
1;
```

- a = a + 1; a = 6; //line breaks can be anywhere

Punctuation

- Punctuations as semicolons, colons, commas, apostrophes, quotation marks, braces, brackets, and parentheses will also be used in C code.

- ; : , ‘ “ [ ] { } ( )

Compound Statements

- Sequences of statements can be combined into one with {...}

```
{
        printf ("Hello, ");
        printf ("world! \n");
}
```

C Statements

Some Suggestions

- DO: Use block braces on their own line.
  - This makes the code easier to read.
- DO: line up block braces so that it is easy to find the beginning and end of a block.
- AVOID: spreading a single statement across multiple lines if there is no need.
  - Try to keep it on one line.

Names of C Variables

Names of C Variables

- A good name for your variables is important
- Variables in C can be given any name made from numbers, letters and underscores which is not a keyword and does not begin with a number.
- Names may contain letters, digits and underscores
- The first character must be a letter or an underscore.
- First 31 characters are significant

  (too long name is as bad as too short).
- Are case sensitive:
  - abc is different from ABC
- Must begin with a letter or underscore and the rest can be letters, digits, and underscores.

Names of C Variables

```
present, hello, y2x3, r2d3, ...          /* OK */
_1993_tar_return                  /* OK but not good */
Hello#there                              /* illegal */
int                             /* shouldn't work */
2fartogo                                 /* illegal */
```

Names of C Variables

```
int a,b;
double d;
/* It is like cryptic */
```

```
int start_time;
int no_students;
double course_mark;
/* It is better */
```

Names of C Variables

Suggestions regarding variable names

- DO: use variable names that are descriptive
- DO: adopt and stick to a standard naming convention
  – sometimes it is useful to do this consistently for the entire software development site
- AVOID: variable names starting with an underscore
  – often used by the operating system and easy to miss
- AVOID: using uppercase only variable names
  – generally these are pre-processor macros (later)

Names of C Variables

- C keywords cannot be used as variable names.
- Sometimes called reserved words.
- Are defined as a part of the C language.
- Can not be used for anything else!
- Examples:
  - int
  - while
  - for

Keywords of C

- Flow control (6) – `if, else, return, switch, case, default`
- Loops (5) – `for, do, while, break, continue`
- Common *types* (5) – `int, float, double, char, void`
- *structures* (3) – `struct, typedef, union`
- Counting and sizing things (2) – `enum, sizeof`
- Rare but still useful *types* (7) – `extern, signed, unsigned, long, short, static, const`
- Evil keywords which we avoid (1) – `goto`
- Wierdies (3) – `auto, register, volatile`

Comments

Comments

- Can be used to write title of the program, author details etc.
- To guide a programmer. To write a note for function, operation, logic etc. in between a program.
- Non-executable statement

Comments

- Comments: /* This is a comment */
- Use them!
- Comments should explain:
  - special cases
  - the use of functions (parameters, return values, purpose)
- explain WHY your code does things the what it does.
- Can't be nested.

  e.g:- /* Hello /* abc */ Hi */   ERROR.

Comments

- Ideally, a comment with each variable name helps people know what they do.
- In Assignments and paper, I like to see well chosen variable names and comments on variables.

More on Comments

A few examples of comments

```
/* This program calculates area of a rectangle
This program is developed by Mr. XYZ */
length = 5; // in km
width = 3;    // in km
```

Scanf()

Reading Numeric Data with scanf

- Reading input from console
- scanf can be used like printf but to read instead of write.
- The scanf function is the input equivalent of printf
  - A C library function in the <stdio.h> library
  - Takes a format string and parameters, much like printf
  - The format string specifiers are nearly the same as those used in printf
- Examples:

  scanf ("%d", &x); /* reads a decimal integer */
- The ampersand (&) is used to get the “address” of the variable
  - If we used scanf("%d",x) instead, the value of x is passed. As a result, scanf will not know where to put the number it reads

Example

```
#include <stdio.h>
#include "stdafx.h“
int main ()
{
      int num1, result_square;
      printf ("Enter an integer value please: ");
      scanf ( "%d",  &num1);
      result_square = num1*num1;
      printf ("\n = Square of your entered number is %d\n\n",
      result_square);
      getchar();
      return 0;
}
```

Reading Numeric Data with scanf

- Reading more than one variable at a time:
  - For example:

```
int n1, n2, n3;
scanf("%d%d%d",&n1,&n2,&n3);
```

  - Use white spaces to separate numbers when input.

```
5 10  22
```

- In the format string:
  - You can use other characters to separate the numbers

```
int no_students, no_chairs;
scanf(“Number of Students=%d, Number of Chairs=% d",
&no_students, &no_chairs);
```

    - You must provide input like:

```
Number of Students= 30, Number of Chairs= 35
```

Example

```
#include <stdio.h>
#include "stdafx.h“
int main( void )
{
      int value1, value2, sum, product ;
      printf(“Enter two integer values: “) ;
      scanf(“Value 1 = %d, Value 2 = %d”, &value1, &value2) ;
      sum = value1 + value2 ;
      product = value1 * value2 ;
      printf(“Sum is = %d \n\n Product = %d\n”, sum, product) ;
      getchar();
      return 0 ;
}
```

The scanf statement

```
int number, check;
check= scanf ("%d",&number);
//Error
```

```
int number, check;
scanf ("%d",&number);
check= number;
//Correct
```

Expressions and Operators

Expressions and Operators

- In the most general sense, a statement is a part of your program that can be executed.
- An expression is a statement.
- Examples:

```
x = 4;
x = x + 1;
printf("%d",x);
```

- Two types:
  - Function calls
  - The expressions formed by data and operators
- An expression in C usually has a value
  - except for the function call that returns void.

Arithmetic Operators

Operator	Symbol	Action	
Addition	+	Adds operands	x + y
Subtraction	-	Subtracts from first	x - y
Negation	-	Negates operand	-x
Multiplication	*	Multiplies operands	x * y
Division	/	Divides first by second (integer quotient)	x / y
Modulus	%	Remainder of divide op	x % y

- `(x % y)` gives the remainder when `x` is divided by `y`
- `remainder= x%y;` (ints only)

Assignment Operator

- The assignment operator =

  x = 3

  – It assigns the value of the right hand side (rhs) to the left hand side (lhs).

  – The value is the value of the rhs.

- For example:

```
x = ( y = 3 ) +1; /* y is assigned 3 */
/* the value of (y=3) is 3 */
/* x is assigned 4 */
```

Compound Assignment Operator

- Often we use “update” forms of operators
  - x=x+1, x=x*2, ...
- C offers a short form for this:

Operator	Equivalent to
x + = y	x = x + y
x *= y	x = x * y
y -= z + 1	y = y - (z + 1)
a /= b	a = a / b
x += y / 8	x = x + (y / 8)
y %= 3	y = y % 3

```
// demonstrates arithmetic assignement operators
#include <stdio.h>
int main()
{
    int  ans = 27;
    ans += 10; //same  as: ans = ans + 10;
    printf(" %d, ",ans);
    ans -= 7; //same  as: ans = ans - 7;
    printf(" %d, ",ans);
    ans *= 2; //same  as: ans = ans * 2;
    printf(" %d, ",ans);
    ans /= 3; //same  as: ans = ans / 3;
    printf(" %d, ",ans);
    ans %= 3; //same  as: ans = ans % 3;
    printf(" %d,  \n",ans);
    return 0;
}
```

Increment and Decrement Operators

Increment and Decrement

- Increment and decrement operators.
  - Increment: ++
    - ++x is the same as : (x = x + 1 or x + = 1). It increases the value of x by 1
  - Decrement: -- (similar to ++)
    - --x is the same as : (x = x – 1 or x - = 1). It decreases the value of x by 1

Increment and Decrement

Pre-fix and Post-fix

- ++i means increment i then use it
- i++ means use i then increment it

```
int i= 6;
printf ("%d\n",i++);   /* Prints 6 sets i to 7 */
```

Note this important difference

```
int i= 6;
printf ("%d\n",++i);   /* prints 7 and sets i to 7 */
```

It is easy to confuse yourself and others with the difference between `++i` and `i++` - it is best to use them only in simple ways.

All of the above also applies to `--`.