Computer Practicum 1

Introduction to C

Vida Groznik

Source: cprogramming.com, tutorialspoint.com, learn-c.org, fresh2refresh.com

Dennis M. Ritchie

(9 Sep. 1941 – 12 Oct. 2011)

- Created C programming language
- Created Unix OS with Ken Thompson
- Co-author of the book *The C Programming Language* (his co-author was Brian Kernighan)

- Awards: **Turing Award** (1983), IEEE Richard W. Hamming Medal (1990), Fellows of the Computer History Museum (1997), National Medal of Technology (1999), etc.

Short history of C

- C was invented at Bell Laboratories in 1972 by Dennis M. Ritchie
- C programming language features were derived from an earlier language called “B” (Basic Combined Programming Language – BCPL)
- C language was invented for implementing UNIX operating system
- In 1978, Dennis Ritchie and Brian Kernighan published the first edition “The C Programming Language” and commonly known as K&R C
- In 1983, the American National Standards Institute (ANSI) established a committee to provide a modern, comprehensive definition of C. The resulting definition, the ANSI standard, or “ANSI C”, was completed late 1988.

What do I need to run C programmes?

- You need the following two software tools available on your computer:
  - **text editor** and
  - the **C Compiler**.
- The source code written in source file is the human readable source for your programme. It needs to be "compiled", into machine language so that your CPU can actually execute the programme.
- The compiler compiles the source codes into final executable programmes.
- The most frequently used and free available compiler is the GNU C/C++ compiler, otherwise you can have compilers either from HP or Solaris if you have the respective operating systems.

- If you are using **Linux or UNIX**, check whether GCC is installed on your system:

```
$ gcc -v
```

- If the compiler is installed, you should see a message as follows:

```
Using built-in specs.

COLLECT_GCC=gcc

COLLECT_LTO_WRAPPER=/usr/lib/gcc/x86_64-linux-gnu/5/lto-wrapper

Target: x86_64-linux-gnu

Configured with: ../src/configure -v --with-pkgversion='Ubuntu 5.4.0-6ubuntu1~16.04.4' --with-........

Thread model: posix

gcc version 5.4.0 20160609 (Ubuntu 5.4.0-6ubuntu1~16.04.4)
```

If GCC is not installed, then you will have to install it yourself using the detailed instructions available at https://gcc.gnu.org/install/

How does a C programme source code look like?

```
1. #include <stdio.h>
2.
3. int main() {
4.    /* my first program in C */
5.    printf("Hello, World! \n");
6.
7.    return 0;
8. }
```

- **Line 1** - `#include <stdio.h>` is a preprocessor command, which tells a C compiler to include `stdio.h` file before going to actual compilation.
- **Line 3** - the main function where the programme execution begins.
- **Line 4** - This is a **comment**.
  Anything between `/*` and `*/` is not executed.
- **Line 5** - `printf(...)` is a function available in C which causes the message "Hello, World!" to be displayed on the screen.
- **Line 7** - terminates the `main()` function and returns the value `0`.

Compile and execute C programme

- Save the file as `hello.c`
- Open the terminal and go to the directory where your file is saved.
- Type `gcc hello.c` and press enter to compile your code. (Line 1)
- If there are no errors in your code, the terminal will take you to the next line and would generate `a.out` executable file.
- Now, type `a.out` to execute your programme. (Line 2)
- You will see the output `"Hello World"` printed on the screen. (Line 3)

```
1. $ gcc hello.c
2. $ ./a.out
3. Hello, World!
```

Tokens & Semicolons

A C programme consists of various **tokens**. They are either a keyword, an identifier, a constant, a string literal, or a symbol.

The following C statement consists of five tokens:

```
printf("Hello, World! \n");
```

The individual tokens are:

```
printf
(
"Hello, World! \n"
)
;
```

A **semicolon** is a statement terminator. That is, each individual statement must be ended with a semicolon.

It indicates the end of one logical entity.

Given below are two different statements:

```
printf("Hello, World! \n");
return 0;
```

Comments & Identifiers

Comments are a helping text in your C programme and are ignored by the compiler.

They start with /* and terminate with the characters */ as shown:

```
/* my first programme in C */
```

You cannot have comments within comments and they do not occur within a string or character literals.

A C **identifier** is a **name** used to identify a *variable*, *function*, or any other user-defined item.

An identifier starts with a letter A to Z, a to z, or an underscore '_' followed by zero or more letters, underscores, and digits (0 to 9).

C **does not allow punctuation characters** such as @, $, and % within identifiers.
C is a **case-sensitive** programming language. Thus, Manpower and manpower are two different identifiers in C.

Keywords

The list shows the reserved words in C.

These reserved words may not be used as constants or variables or any other identifier names.

auto	else	long
switch	break	enum
register	typedef	case
extern	return	union
char	float	short
unsigned	const	for
signed	void	continue
goto	sizeof	volatile
default	if	static
while	do	int
struct	_Packed	double

Data types

Data types in c refer to an extensive system used for declaring variables or functions of different types.

The type of a variable determines how much **space** it occupies in storage and how the bit pattern stored is interpreted.

The array types and structure types are referred collectively as the aggregate types. **The type of a function specifies the type of the function's return value.**

	Types & Description
1	**Basic Types** They are arithmetic types and are further classified into: • integer types and • floating-point types.
2	**Enumerated types** Arithmetic types and are used to define variables that can only assign certain discrete integer values throughout the program.
3	**The type void** The type specifier void indicates that no value is available.
4	**Derived types** They include: • Pointer types, • Array types, • Structure types, • Union types and • Function types.

Integer types

The following table provides the details of standard integer types with their storage sizes and value ranges.

To get the exact size of a type or a variable on a particular platform, you can use the `sizeof` operator. The expressions `sizeof(type)` yields the storage size of the object or type in bytes.

Type	Storage size	Value range
char	1 byte	-128 to 127 or 0 to 255
unsigned char	1 byte	0 to 255
signed char	1 byte	-128 to 127
int	2 or 4 bytes	-32,768 to 32,767 or -2,147,483,648 to 2,147,483,647
unsigned int	2 or 4 bytes	0 to 65,535 or 0 to 4,294,967,295
short	2 bytes	-32,768 to 32,767
unsigned short	2 bytes	0 to 65,535
long	4 bytes	-2,147,483,648 to 2,147,483,647
unsigned long	4 bytes	0 to 4,294,967,295

Floating-point types

The following table provide the details of standard floating-point types with storage sizes and value ranges and their precision.

The header file `float.h` defines macros that allow you to use these values and other details about the binary representation of real numbers in your programs.

Type	Storage size	Value range	Precision
float	4 byte	1.2E-38 to 3.4E+38	6 decimal places
double	8 byte	2.3E-308 to 1.7E+308	15 decimal places
long double	10 byte	3.4E-4932 to 1.1E+4932	19 decimal places

The void type

The void type specifies that no value is available. It is used in three kinds of situations:

1. **Function returns as void**
   There are various functions in C which do not return any value or you can say they return void. A function with no return value has the return type as void. For example, `void exit (int status);`
2. **Function arguments as void**
   There are various functions in C which do not accept any parameter. A function with no parameter can accept a void. For example, `int rand(void);`
3. **Pointers to void**
   A pointer of type void * represents the address of an object, but not its type. For example, a memory allocation function void `*malloc( size_t size );` returns a pointer to void which can be casted to any data type.

Variables

A variable is a name given to a storage area that our programmes can manipulate. Each variable in C has a specific **type**, which determines the **size and layout** of the variable's memory; the **range of values** that can be stored within that memory; and the set of **operations** that can be applied to the variable.

The name of a variable can be composed of letters, digits, and the underscore character. It must begin with either a letter or an underscore. Upper and lowercase letters are distinct because C is case-sensitive.

The basic variable types:

Type	Description
char	Typically a single octet(one byte).
int	The most natural size of integer for the machine.
float	A single-precision floating point value.
double	A double-precision floating point value.
void	Represents the absence of type.

Variable definition and initialization

A variable **definition** tells the compiler **where and how much storage** to create for the variable. A variable definition **specifies a data type** and contains a **list** of one or more **variables** of that type as follows:

```
type variable_list;
```

Examples:

```
int    i, j, k;
char   c, ch;
float  f, salary;
double d;
```

Variables can be **initialized** (assigned an initial value) in their declaration. The initializer consists of an equal sign followed by a constant expression as follows:

```
type variable_name = value;
```

```
extern int d = 3, f = 5;  // declaration of d & f.
int d = 3, f = 5;         // definition and
                          // initializing d and f.
byte z = 22;              // definition and
                          // initializes z.
char x = 'x';             // the variable x has
                          //the value 'x'.
```

Scope of the variables – local variable

- The scope of local variables will be within the function only.
- These variables are declared within the function and can't be accessed outside the function.
- In the example, m and n variables are having scope within the main function only. These are not visible to test function.
- Like wise, a and b variables are having scope within the test function only. These are not visible to main function.

Output:

```
values : m = 22 and n = 44
values : a = 50 and b = 80
```

```c
#include<stdio.h>
void test();

int main()
{
   int m = 22, n = 44;
     // m, n are local variables of main function
     /*m and n variables are having scope within this main
     function only. These are not visible to test funtion.*/
     /* If you try to access a and b in this function,
      you will get 'a' undeclared and 'b' undeclared error */
   printf("\nvalues : m = %d and n = %d", m, n);
   test();
}

void test()
{
   int a = 50, b = 80;
     // a, b are local variables of test function
     /*a and b variables are having scope within this test
     function only. These are not visible to main function.*/
     /* If you try to access m and n in this function, you
     will get 'm' undeclared and 'n' undeclared error */
   printf("\nvalues : a = %d and b = %d", a, b);
}
```

Scope of the variables – global variable

- The scope of global variables will be throughout the program. These variables can be accessed from anywhere in the program.
- This variable is defined outside the main function. So that, this variable is visible to main function and all other sub functions.

```c
#include<stdio.h>
void test();int m = 22, n = 44;
int a = 50, b = 80;

int main()
{
   printf("All variables are accessed from main function");
   printf("\nvalues: m=%d:n=%d:a=%d:b=%d", m,n,a,b);
   test();
}

void test()
{
   printf("\n\nAll variables are accessed from" \
   " test function");
   printf("\nvalues: m=%d:n=%d:a=%d:b=%d", m,n,a,b);
}
```

Output:

```
All variables are accessed from main function
values : m = 22 : n = 44 : a = 50 : b = 80
All variables are accessed from test function
values : m = 22 : n = 44 : a = 50 : b = 80
```

Scope of the variables – environment variable

- Environment variable is a variable that will be available for all C applications and C programs.
- We can access these variables from anywhere in a C program without declaring and initializing in an application or C program.
- The inbuilt functions which are used to access, modify and set these environment variables are called environment functions.
- There are 3 functions which are used to access, modify and assign an environment variable in C. They are:
    1. `setenv()`
    2. `getenv()`
    3. `putenv()`

Reading input

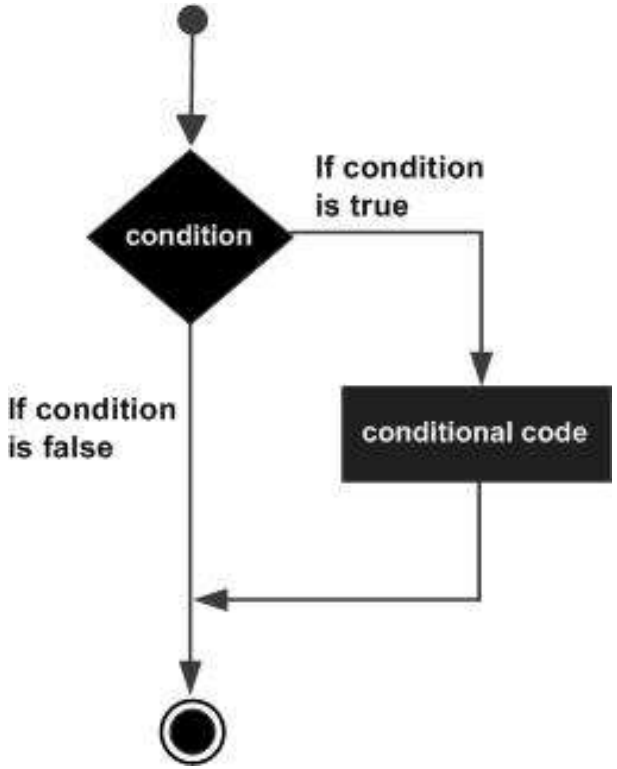
To read in a value, we use `scanf` function and `printf` to read it back out.

```c
#include <stdio.h>
int main()
{
  int this_is_a_number;

  printf("Please enter a number: ");
  scanf("%d", &this_is_a_number);
  printf("You entered %d", this_is_a_number);
  getchar();
  return 0;
}
```

if – statement

In “if” control statement, respective block of code is executed when condition is true.



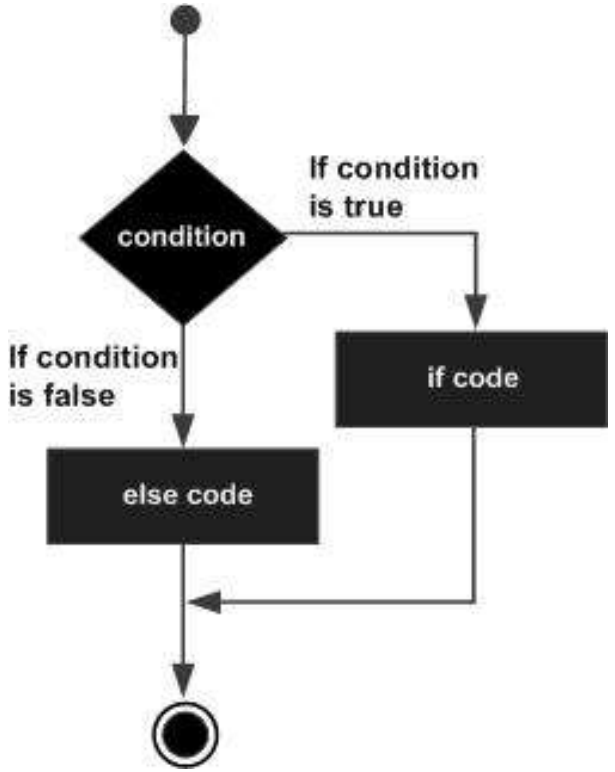

```
1  int main()
2  {
3    int m=40,n=40;
4    if (m == n)
5    {
6      printf("m and n are equal");
7    }
8  }
```

Output:

```
m and n are equal
```

if-else – statement

In if-else control statement, group of statements are executed when condition is true. If condition is false, then else part statements are executed.




```c
#include <stdio.h>
int main()
{
  int m=40,n=20;
  if (m == n)
  {
    printf("m and n are equal");
  }
  else
  {
    printf("m and n are not equal");
  }

}
```

Output:

```
m and n are not equal
```

if-else if-else – statement

- In “nested if” control statement, if condition 1 is false, then condition 2 is checked and statements are executed if it is true.
- If condition 2 also gets failure, then else part is executed.

```c
#include <stdio.h>
int main()
{
  int m=40,n=20;
  if (m>n)
  {
    printf("m is greater than n");
  }
  else if(m<n)
  {
    printf("m is less than n");
  }
  else
  {
    printf("m is equal to n");
  }
}
```

Output:

```
m is greater than n
```

for loop

In for loop control statement, loop is executed until condition becomes false.

```
for (exp1; exp2; expr3)
{
  statements;
}
```

Where:

exp1 – variable initialization

(Example: i=0, j=2, k=3)

exp2 – condition checking

(Example: i>5, j<3, k=3)

exp3 – increment/decrement

(Example: ++i, j–, ++k)

```
1  #include <stdio.h>
2
3  int main()
4  {
5    int i;
6
7    for(i=0;i<10;i++)
8    {
9        printf("%d ",i);
10   }
11
12 }
```

Output:

```
0 1 2 3 4 5 6 7 8 9
```

while loop

In while loop control statement, loop is executed until condition becomes false.

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

Where condition might be a>5, i<10

```
1   #include <stdio.h>
2
3   int main()
4   {
5     int i=3;
6
7     while(i<10)
8     {
9        printf("%d\n",i);
10       i++;
11    }
12
13  }
```

Output:

```
3 4 5 6 7 8 9
```

do-while loop

In do-while loop control statement, while loop is executed irrespective of the condition for first time. Then 2nd time onwards, loop is executed until condition becomes false.

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

Where condition might be a>5, i<10

```
#include <stdio.h>

int main()
{
  int i=1;

  do
  {
      printf("Value of i is d\n",i);
      i++;
  }
  while(i<=4 && i>=2);
}
```

Output:

```
Value of i is 1
Value of i is 2
Value of i is 3
Value of i is 4
```

Exercise 1

Write a programme which prints out the sum of the numbers `a (int)`, `b (float)` and `c (double)`.

Exercise 2

Write a programme which tells you whether a number, entered by a user is a prime number or not.

Exercise 3

Write a programme which asks a user to enter an integer n and calculate a factorial of that number (n!).

Check whether a user has entered a positive number. If the number is not positive, print an error and return value 1.

Exercise 4

Write a programme which prints out first n Fibonaci numbers.

The variable n should be provided by a user.

Exercise 5

Write a programme which checks whether an entered number is a palindrome.

Exercise 6

Write a progamme which is provided with three numbers by a user.

1. Find the largest of given 3 numbers.
2. Find the smallest of the given 3 numbers.
3. Find a sum of given 3 numbers.
4. Find an average of given 3 numbers.