BACK TO C Basics

Version 1.0

By Malith Perera

Introduction

This document is focused on the basics of C programming concepts. It does not cover nitty-gritty details of C programming. But it is more than enough you to write a good C program after referring this.

What you have to do here is copy, paste and save the given code from your editor. Then compile and run. Inspect the code carefully till you get understand it. Impotent parts in the code are highlighted. I'm sure you will grab the essence. It's simple and easy. Same thing is repeating again and again.

Next do some changes to your code. As an example delete semicolon. Then save and compile. You will get errors. Analyze them carefully. Fix them back. Now you know what are these errors and why do they occur.

Here we use GCC compiler in Linux terminal with vim editor. It's free. So you can give it a try. Else you can use MingW compiler and VSCode as editor in Windows or GCC and XCode in Mac OS. Please search internet how to install, edit, compile and run C programs in your environment.

Play with your code and enjoy.

Happy Coding!

Support

This simple document is written in LibreOffice Writer which also a free and open source software. This is a free document you can use it, change it, enhance it as you like. If you find any error with this document you can fix it and save it on github using link below.

And also you can translate it to your native language. It will be a great help for other who speaks your language too. Anyone who translate, fix or design this document more colorful do not forget to include your names in editors page next to this. Please keep this document up to date and as simple as possible.

Thank you!

Document link:

Community Support

Please join with our community and you can ask any question what you get here.

Editors page

Edited by
Malith Perera

Designed by

Fixed by

Vim Editor Basic

Normal mode	Esc	
Insert mode	i	

In normal mode

Write (Save)	: w	
Select	v	
Copy	y	(or "+y)
Paste	p	(or "+p)
Delete	d	
Undo	u	
Redo	ctrl + r	

Note:

Though vim is a advanced powerful editor it's little hard to learn. And It also take some time to used to it. Anyway if you do not like it use any editor or an IDE as you like.

To add some enhanced features to your existing vim editor visit:
https://vim.spf13.com/

To download gvim for Windows:
https://www.vim.org/download.php

Hello World!

Your first C program.

```
#include <stdio.h>

int main()
{
    printf("Hello World!\n");

    return 0;
}
```

Copy above code

Terminal commands:

$ vim hello.c	(Open vim, Paste "+p, save :w)
$ gcc hello.c	(Compile)
$ ./a.out	(Run)

Comments

Comment your code

```
/* comment.c */
/*   In this program you will
     Print your name */

#include <stdio.h>

int main()
{
    printf("Hello Your name!\n"); // Write your name here

    return 0;
}
```

Comments are used to keep notes within your code. No any executable output given.

// one line comment

/* multi line comment */

Terminal commands:

```
$ vim comment.c
$ gcc comment.c
$ ./a.out
```

C data types

int	Integer (whole numbers)
float	Numbers with decimals
char	Character (one character)

Define a variable

```
int number1;

float  number2;

char character;
```

You can change variable name as you like above.

Define and initialize a variable value

```
int number1 = 7;

float  number2 = 1.25;

char character =  'A';  /* Only one character */
```

You can change or assign a variable value any time after defined.

Integer variable

Print integer variable

```c
#include <stdio.h>

int main()
{
    int number = 7;

    printf("Integer value: %d", number);// %d for int

    return 0;
}
```

Terminal commands:

```
$ vim var_int.c
$ gcc var_int.c
$ ./a.out
```

Float variable

Print float variable

```c
#include <stdio.h>

int main()
{
    float number = 4.5;

    printf("Float value: %f", number); // %f for float

    return 0;
}
```

Terminal commands:

```
$ vim var_float.c
$ gcc var_float.c
$ ./a.out
```

Char variable

Print char variable

```c
#include <stdio.h>

int main()
{
    char character = 'A';

    printf("Char value: %c", character); // %c for char

    return 0;
}
```

Terminal commands:

```
$ vim var_char.c
$ gcc var_char.c
$ ./a.out
```

for Loop

Print Hello 10 times

```c
#include <stdio.h>

int main()
{
    // From i=0 increase i one by one with i++ till i<10
    for (int i = 0; i < 10; i++)
    {
        printf ("Hello!\n");
    }

    return 0;
}
```

Terminal commands:

```
$ vim loop_for.c
$ gcc loop_for.c
$ ./a.out
```

while loop

Print Hello 10 times

```c
#include <stdio.h>

int main ()
{
    int i = 0;              // From i = 0

    while (i < 10)          // Repeat while i < 10
    {
        printf ("Hello!\n");
        i++;                // Increase i one with i++
    }

    return 0;
}
```

Terminal commands:

```
$ vim loop_while.c
$ gcc loop_while.c
$ ./a.out
```

if selection

Select with if

```c
#include <stdio.h>

int main()
{
    int marks = 85;

    if (marks >= 75) // If marks greater than or equal to 75
    {
        printf ("A\n"); // print A
    }

    return 0;
}
```

Terminal commands:

$ vim select_if.c
$ gcc select_if.c
$ ./a.out

if else selection

Select what we want

```
#include <stdio.h>

int main()
{
    int marks = 85;

    if (marks >= 75) // If marks greater than or equal to 75
    {
        printf ("Pass\n");
    }
    else             // Else do this
    {
        printf ("Fail\n");
    }

    return 0;
}
```

Terminal commands:

$ vim select_if_else.c
$ gcc select_if_else.c
$ ./a.out

else if selection

Select what we want

```
#include <stdio.h>

int main()
{
     int marks = 85;

     if (marks >= 75)        // If marks greater than or equal to 75
     {
          printf ("A\n");
     }
     else if (marks >= 65)   // If marks greater than or equal to 65
     {
          printf ("B\n");
     }
     else                    // Else do this
     {
          printf ("F\n");
     }

     return 0;
}
```

Terminal commands:

```
$ vim select_else_if.c
$ gcc select_else_if.c
$ ./a.out
```

Array

Sequence of a variables

Think you need hundreds of variables for a calculation. It is difficult to give every variable a name and assign one by one. C gives you a way called arrays. Array is a variable that can hold sequence of many variable of same type at once.

Define an array

```
int marks[3]; // Student marks for 3 subjects

float temperatures[100]; // Hundred temperature values

char name[10]; // Able to hold up to 10 characters
```

Define and initialize array values

```
int marks[3] = {95, 85, 90};
```

Assign array values

```
int marks[3]; // Able to hold up to 3 integers

marks[0] = 90; // index begin with 0
marks[1] = 95;
marks[2] = 85;
```

Use array values

```
int total_marks = marks[0] + marks[1] + marks[2];
```

To deal with values of an array you should index it as above. Remember array index number is always begin with 0.

Array total

Calculate total marks

```c
#include <stdio.h>

int main()
{
    int marks[3] = {85, 90, 95};

    int total_marks = marks[0] + marks[1] + marks[2];

    printf("Total: %d", total_marks);

    return 0;
}
```

Terminal commands:

```
$ vim array_total.c
$ gcc array_total.c
$ ./a.out
```

Loop an array

Calculate total of an integer array

```
#include <stdio.h>

int main()
{
    int marks[3] = {85, 95, 90};
    int total_marks = 0;

    for (int i = 0; i < 3; i++ )
    {
        total_marks = total_marks + marks[i]; // or total_marks += marks[i];
    }

    printf ("Total: %d", total_marks);

    return 0;
}
```

Terminal commands:

```
$ vim array_total_marks.c
$ gcc array_total_marks.c
$ ./a.out
```

Structures

Compaq more than one variable types together

Define a structure

```
typedef struct student {
    int index;
    float average;
} Student;
```

Now you can use Student as a new data type like int or float to define variables.

```
Student st1;  // student 1
```

Here st1 variable has two member variables index and average. You can access and use them as below.

```
st1.index = 100;
st1.average = 88.8;
```

Structure student

Two students

```c
#include <stdio.h>

typedef struct student {
    int index;
    float average;
} Student;

int main()
{
    Student st1; // Student 1

    st1.index = 100;
    st1.average = 88.8;

    printf("index: %d average: %f\n", st1.index, st1.average);

    return 0;
}
```

Terminal commands:

$ vim struct_student.c
$ gcc struct_ student .c
$ ./a.out

Array of structure

Array of structure type

```
#include <stdio.h>

typedef struct student {
     int index;
     float average;
} Student;

int main()
{
     Student st[2];          // Array of 2 Student structures

     st[0].index = 100;      // First student
     st[0].average = 88.8;

     st[1].index = 101;      // Second student
     st[1].average = 90.2;

     printf("index: %d average: %f\n", st[0].index, st[0].average);
     printf("index: %d average: %f\n", st[1].index, st[1].average);

     return 0;
}
```

Terminal commands:

$ vim struct_example.c

$ gcc struct_example .c

$ ./a.out

Address

Address of a memory location

You can use & to get the address of a memory location.

Address of a variable

```
int number;  // Variable
&number;     // Address of this variable
```

Address of an array

```
int students[20];         // Array
&student;                 // Address of array
&student[2];              // Address of third student in array
```

Address of a structure

```
Student student1;         // Structure student
&student1;                // Address of student1
```

Memory address

Array of memory locations

```c
#include <stdio.h>

typedef struct student {
    int index;
    float average;
} Student;

int main()
{
    int number = 5;                    // Variable
    int marks[3] = { 80, 90, 95 };     // Array
    Student st1;                       // Structure

    printf("Address of number   %p\n", &number);
    printf("Address of marks    %p\n", &marks);
    printf("Address of marks[2] %p\n", &marks[2]);
    printf("Address of st1      %p\n", &st1);

    return 0;
}
```

Terminal commands:

```
$ vim struct_example.c
$ gcc struct_example .c
$ ./a.out
```

Pointers

Point to memory location

Pointers can hold address of a memory location. If you know the address of a memory location you can use pointers to hold , access and pass that memory location.

Define a pointer

```
int* nptr;            // nptr is a pointer to an integer
                      // nptr can hold address of a integer variable
```

Asign

```
int number;
int* numberPtr;
numberPtr = &number; // &number is the address of number
```

Access value of a pointed variable

```
nptr = &number;

*nptr = *&nubmer;  // Add * left and right. Then *& cancels out.

*nptr = number;
```

Pointers

Pointer to an integer

```
#include <stdio.h>

int main()
{
     int number = 5;
     int* nPtr = &number; // nptr is a pointer to an integer

     printf("Address of number with &number : %p\n", &number);
     printf("Address of number in nPtr      : %p\n", nPtr);
     printf("Access number value with nPtr  : %d\n", *nPtr);

     return 0;
}
```

Terminal commands:

```
$ vim pointer_to_int.c
$ gcc pointer_to_int.c
$ ./a.out
```

Pointers

Pointer to an array

```
#include <stdio.h>

int main()
{
     int number[2] = {1, 2};

     int* nPtr = number; // See you need no & to get array address

     printf("Access array with pointer : %d, %d\n", nPtr[0], nPtr[1]);

     return 0;
}
```

Terminal commands:

```
$ vim pointer_to_int.c
$ gcc pointer_to_int.c
$ ./a.out
```

Pointers of void

Pointer of void

```
void fun2 (void* p)
{
   printf ("Value of a fun2 is %d\n", *((int*) p));
}


int main()
{
   int a = 7;
   void* p = &a;
   printf("Integer variable is = %d\n", *( (int*) p) );

   return 0;
}
```

Terminal commands:

```
$ vim pointer_to_int.c
$ gcc pointer_to_int.c
$ ./a.out
```

Functions

New function

```c
#include <stdio.h>

// define a new function
void New_Function()
{
    printf("Write above main function\n");
}

int main()
{
    New_Function(); // Calling New_Function

    return 0;
}
```

Terminal commands:

```
$ vim function_new.c
$ gcc function_new.c
$ ./a.out
```

Function Prototype

Make function portable

```
#include <stdio.h>

// Function prototype
void New_Function();

int main()
{
    New_Function(); // Calling New_Function

    return 0;
}

void New_Function()
{
    printf("Now you can write below main function\n");
}
```

Terminal commands:

```
$ vim function_new.c
$ gcc function_new.c
$ ./a.out
```

Function Parameters

Pass values to a function

```
#include <stdio.h>

void Add(int a, int b)
{
    int total = a + b;
    print ("Total: %d\n", total);
}

int main()
{
    Add(5, 10);

    return 0;
}
```

Terminal commands:

```
$ vim function_new.c
$ gcc function_new.c
$ ./a.out
```

Function Parameters

Return value from a function

```c
#include <stdio.h>

int Add(int a, int b)          // return type is int
{
    int total = a + b;
    return total;
}

int main()
{
    int tot = Add(5, 10); // tot get return of Add function

    printf("Total: %d\n", tot);

    return 0;
}
```

Terminal commands:

```
$ vim function_new.c
$ gcc function_new.c
$ ./a.out
```

Function Parameters

Pass and return variable from a function

```c
#include <stdio.h>

int Add(int a, int b)
{
    return a + b;             // add a + b and return
}

int main()
{
    int n1 = 5;               // variables
    int n2 = 10;

    int tot = Add(n1, n2);    // pass n1 and n2 to Add

    printf("%d + %d = %d\n", n1, n2, tot);

    return 0;
}
```

Terminal commands:

```
$ vim function_var.c
$ gcc function_var.c
$ ./a.out
```

Function Parameters

Pass variable with a pointer

```
#include <stdio.h>

int Add_One (int* nPtr)
{
    *nPtr += 1;          // *nPtr = number
}


int main()
{
    int number = 5;      // variables

    printf("number = %d\n", number);

    Add_One(&number);    // pass address of the number

    printf("number = %d\n", number);

    return 0;
}
```

Terminal commands:

```
$ vim function_var.c
$ gcc function_var.c
$ ./a.out
```

Function Parameters

Pass and array to a function

```
#include <stdio.h>

void Add_One(int arr[])
{
     for (int i=0; i<2; i++)
     {
          arr[i] += 1; // add one for each array variable
     }
}

int main()
{
     int age[2] = { 10, 15 };

     printf ("age: %d, %d\n", age[0], age[1]);

     Add_One(age);

     printf ("age: %d, %d\n", age[0], age[1]);

     return 0;
}
```

Terminal commands:

$ vim function_array.c
$ gcc function_array.c
$ ./a.out

Function Parameters

Pass array to a function with a pointer

```
#include <stdio.h>

void Add_One(int* arr)
{
     for (int i=0; i<2; i++)
     {
          arr[i] += 1; // You can index pointer like an array
     }
}

int main()
{
     int age[2] = { 10, 15 };

     printf ("age: %d, %d\n", age[0], age[1]);

     Add_One(age);

     printf ("age: %d, %d\n", age[0], age[1]);

     return 0;
}
```

Terminal commands:

$ vim function_array.c
$ gcc function_array.c
$ ./a.out

Function Pointers

Pass array to a function with a pointer

```
#include <stdio.h>

void fun2 (void* p)
{
   printf ("Value of a fun2 is %d\n", *((int*) p));
}


int main()
{
   int a = 7;
   void* p = &a;
   printf("Integer variable is = %d\n", *( (int*) p) );


   // fun_ptr is a pointer to function fun()
   void (*fun_ptr)(int) = &fun;

   /* The above line is equivalent of following two
   void (*fun_ptr)(int);
   fun_ptr = &fun;
   */

   // Invoking fun() using fun_ptr
```

```
    (*fun_ptr)(10);


    fun2 (&a);

    return 0;
}
```

Terminal commands:

```
$ vim function_array.c
$ gcc function_array.c
$ ./a.out
```

Function Arguments of void

Pass array to a function with a pointer

```
#include <stdio.h>

void fun2 (void* p)
{
   printf ("Value of a fun2 is %d\n", *((int*) p));
}

void fun2 (void* p)
{
   printf ("Value of a fun2 is %d\n", *((int*) p));
}


int main()
{
   int a = 7;
   void* p = &a;
   printf("Integer variable is = %d\n", *( (int*) p) );


   // fun_ptr is a pointer to function fun()
   void (*fun_ptr)(int) = &fun;

   /* The above line is equivalent of following two
```

```
    void (*fun_ptr)(int);
    fun_ptr = &fun;
    */

    // Invoking fun() using fun_ptr
    (*fun_ptr)(10);


    fun2 (&a);

    return 0;
}
```

Terminal commands:

```
$ vim function_array.c
$ gcc function_array.c
$ ./a.out
```

Macro

Replace text

```
#include <stdio.h>

// Define a text replacement macro
#define TEXT "print this text"

int main()
{
    printf ("%s\n", TEXT);

    return 0;
}
```

Terminal commands:

```
$ vim function_array.c
$ gcc function_array.c
$ ./a.out
```

Macro

Function like macro

```
#include <stdio.h>

#define SQUARE(a) ((a)*(a))
#define ADD(n1, n2) ((n1) + (n2))
#define PI 3.141
#define CIRCLE_AREA(r) (PI*(r)*(r))

int main()
{
    int sq = SQUARE(4);

    printf("Square         : %d\n", sq);

    printf("Add 2 numbers  : %d\n", (ADD(1,2)));

    int radius = 7;
    int circle_area = CIRCLE_AREA (radius);

    printf("Circle area    : %d\n", circle_area);

     return 0;
}
```

Terminal commands:

$ vim function_array.c
$ gcc function_array.c
$ ./a.out