بِسْمِ اللهِ الرَّحْمٰنِ الرَّحِيْمِ

(In the name of God, the Most Gracious, the Most Merciful.)

C

Starting to learn C, C is used in Operating systems, devices like tablets and printers, embedded in systems where programs run faster in limited memory( like in microwave, camera), etc.

C is also used to develop games, because of low latency.

Remember:

1. Many times in complex loops and recursions use print function to get understanding of the iterations.
2. May not use a semicolon after function definitions or control structures.
3. Must use a semicolon after struct, union, and enum declarations, as well as variable initializations
4. In c defining a variable in a scope is necessary, the scope is usually limited to the closest curly brackets.
5. Using scanf(“%c”) again and again must be done with fflush(stdin) in between to flush the enter key, otherwise the newline char may be stored as an element.
6. (\*ptr)=e2; may not work if ptr remains uninitialized.
7. A char can be always compared to a number like a<112 and a is of char type or converted directly.
8. Everything except 0 is treated as true in c. So if(int a=0) is always false and if(int a=234) is always true.
9. An array pointer is different from a char pointer even though they both point to a char(the first character of the array and the character that the pointer points to.).
10. Use percentage n after print to avoid printing garbage at the end.
11. C global variables may have 0 as their default value but the rest usually have garbage values by default.
12. Making a variable constant using const will make the programmer unable to change its value down the road.
13. NUL and null are different concepts.
14. If I use a function I in the for( loop definition) I still call it again and again for comparison, hence it’ll be more efficient to store the function value in a variable and use it.
15. Always have big no as strings.
16. Code smell

Problems with C:

1. Integer Overflow
   1. The counter after fully filling the no. of reserved bytes is reversed to 0.
2. Truncation
3. Floating-Point Imprecision
4. Default value of an integer may or may not be 0. Global variables are 0 by default but the local may have garbage values.

Actually a compiler does a lot more than just compiling, it does these processes:

1. Pre-processing
   1. The lines in C code which start with a hash are called Preprocessing Directives.
   2. These files mainly contain the function prototypes.
   3. In pre-processing, it finds and replaces the preprocessing directives into functions (used in the program) prototypes.
2. Compiling
   1. Though it is more used as a catch-all phrase to address all the 4 processes together
   2. The conversion of C language to assembly language takes place.
3. Assembling
   1. Takes the assembly code and converts it into 0s and 1s.
   2. Its side effect: Clang names assembly output(a.out).
4. Linking
   1. The preprocessor directive files and the code that we wrote gets out from the assembly as different sets of 1s and 0s.
   2. In this process, they are converted into a single code or one can say linked together.

The lines in C code which start with a hash are called Preprocessing Directives.

These files mainly contain the function prototypes.

Reverse engineering:

As there are more ways to do a single problem and many ways to implement the same thing and you lose sensible variable and function names in binary. This process in itself is very difficult. And if you are smart enough to understand the code of just from the 0s and 1s then you’re most probably smart enough to write that code yourself.

Debugging:

* Use printf, especially in loops printing the iteration that is taking places can be very helpful to track in the dry run.
* Learn a debugger
* Rubber Duck Debugging is a technique in which you try to explain the logic to an inanimate object.

Chapter 1: Variables, Constants & Keywords

* Variables
* In computer programming, a variable is an abstract storage location paired with an associated symbolic name, which contains some known or unknown quantity of data or object referred to as a value; or in simpler terms, a variable is a named container for a particular set of bits or type of data.
* A variable is an entity whose value can change.
* Variables are of many types:

| Type | Memory(in byte) |  |
| --- | --- | --- |
| bool | 1 | Easier even though we are wasting 7 of the bits. |
| int | 4 |  |
| long | 8 |  |
| float | 4 |  |
| double | 8 |  |
| char | 1 | Stores only a single character. |
| string | ? |  |
|  |  |  |

Rules for naming a variable:

1. First character is always an alphabet or underscore( \_ ).
2. No special symbols, blanks, commas, are allowed. However underscore and numbers are allowed.
3. Variable names are case sensitive.
4. Variables can’t be named after keywords/reserved words.
5. A good practice is to name variables on what they are used for, for increased readability.

Constants

An entity whose value doesn’t change

There are 3 types of constants:

1. Integer constants: 1,2,3,-1,-2
2. Real constants: -3.2.2, 7.0, 2.5
3. Character constants: ’t’, ‘(‘ , it must be enclosed within single inverted commas.

Keywords

These words are reserved words, whose meaning is already known to the compiler. There are 32 such keywords available in C.

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

First C program:

#Include<stdio.h>

int main(){

printf(“Hello, I am learning C with Harry”);

return 0;

}

1. #Include: #include is a way of including a standard or user-defined file in the program and is mostly written at the beginning of any C program. The #include preprocessor directive is read by the preprocessor and instructs it to insert the contents of a user-defined or system header file in our C program. Header files contain a set of predefined standard library functions. The .h is the extension of the header files in C.
2. int main(){
   1. Here int indicates that the output of the main function is an integer.
   2. The main function is predefined in C telling the compiler to execute instructions inside it.
   3. () or (Void) can be used to tell that the main function requires no input.
3. {} begins and ends a function and its definition.
4. ; a semicolon is always used to terminate a statement in C. However there are few scenarios when we can write a running program without a semicolon.
5. return 0 gives the output 0 to the main function(must be an integer) telling the compiler that you have successfully reached the very end of the program. If you use return 1, the program will be executed but still report an error. If return is used in a function definition, it breaks right there and the statements following it are not processed (pre mature return). The program may work without returning 0 in C99 or later. Usually 0 means false in C but this is 0 so that error code 1 to infinity are defined.
6. It’s important to remember that the instructions are case sensitive and they are executed line-by-line.

Comments

Comments are used to clarify something about the program in the plain language.

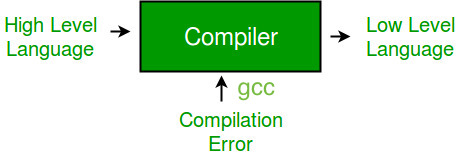
It is a way to add notes to a program.

They aren’t executed but are ignored.

1. Single-line comment: // This is a comment
2. Multi-line comment: /\* this is a multi-line comment\*/

Compilation and Execution:

A compiler is a computer program which converts a C program into machine language so that it can be easily understood by the computer.



Library functions:

The Standard Function Library in [C](https://www.geeksforgeeks.org/c-language-set-1-introduction/) is a huge library of sub-libraries, each of which contains the code for several [functions](https://www.geeksforgeeks.org/functions-in-c/). In order to make use of these libraries, link each library in the broader library through the use of [header files](https://www.geeksforgeeks.org/header-files-in-c-cpp-and-its-uses/). The actual definitions of these functions are stored in separate library files, and declarations in header files. In order to use these [functions](https://www.geeksforgeeks.org/functions-in-c/), we have to include the header file in the program. Below are some header files with descriptions:

| S No. | Header Files | Description |
| --- | --- | --- |
| 1 | <assert.h> | It checks the value of an expression that we expect to be true under normal circumstances.  If the expression is a nonzero value, the assert macro does nothing. |
| 2 | <complex.h> | A set of functions for manipulating complex numbers. |
| 3 | <float.h> | Defines macro constants specifying the implementation-specific properties of the  floating-point library. |
| 4 | <limits.h> | These limits specify that a variable cannot store any value beyond these limits, for example-  An unsigned character can store up to a maximum value of 255. |
| 5 | <math.h> | The math.h header defines various mathematical functions and one macro. All the Functions  in this library take double as an argument and return double as the result. |
| 6 | <stdio.h> | The stdio.h header defines three variable types, several macros, and various  function for performing input and output. |
| 7 | <time.h> | Defines date and time handling functions. |
| 8 | <string.h> | Strings are defined as an array of characters. The difference between a character array  and a string is that a string is terminated with a special character ‘\0’. |

printf()

* Syntax: printf(“This is %d, i);
* printf allows you to choose the following types for variables:
  + %c : Character
  + %f: Float
  + %i: integer
  + %li: Long integer
  + %s: string
* printf(“%.5f\n”, z) prints till the 5th digit after the decimal,

scanf()

* Syntax: scanf(“%d %d %d”, &I, &j, &k);
* & means address of, it means that the supplied value should be copied to the address which is indicated by variable i.
* To ask again & again for an conditioned input:

int n;

do{ n=scanf(“%i”, &n);}

while(n<I); //“n” is mentioned outside due to the scope limitations.

Chapter 2: Instructions and Operators

Expressions return a value even if it’s void, while statements don’t evaluate to something but have side effects.

Type of Instructions:

1. Type declaration instructions:
   1. In C we must declare the datatype of the variable before using it.
   2. Eg: int a,b,c,d,e=10, f,g,h=a-1;
2. Arithmetic Instructions:
   1. Eg: (3\*2)+1
   2. +,-,/,\*, and % is used.
   3. % is a modular division operator which returns remainder. But it cannot be applied on floats and always returns a sign as same as of the numerator.
   4. There is no operator to perform exponential in C. But can use pow(x,n) from <math.h>, which returns float value.
   5. 4^5 =1 as caret in C denotes the XOR bitwise operator.
   6. Type Conversion:
   7. bool -> char -> short int -> int -> unsigned int -> long -> unsigned -> long long -> float -> double -> long double
   8. If we force int a=3.5; a would become 3, similarly if an operator acts between float and int it will return a float.
   9. Operator precedence though is from left to right and used with a table, but it always seems to respect bodmas.
   10. It would be more appropriate to say that 6.0/2-2 returns a double data type, than to say that it returns a float.
3. Control Instructions:
   1. Determines the flow of control in a program.
      1. Sequence Control Instructions: All other than the other three.
      2. Decision Control Instructions: if, else.
      3. Loop Control Instructions: while, while-do, for.
      4. Case Control Instructions: switch.

Chapter 3: Conditional and Case Control Instructions

1. If else statements.
2. Switch statements.

If-Else statements

if(condition1){

statements if condition 1 is true;

} //Note: No semicolon here, as else is related to the if.

else if(condition 2){

statements which are executed if and only if condition 1 is false and condition 2 is true.

} //again no semicolon here

else {

statements if both conditions are false

}

Relational Operators: ==(equality check), <=,>=,<,>,!=(not equal to).

Logical operators: &&(and), ||(or), ! (Not)

Operator precedence:

| Priority | Operator |
| --- | --- |
| 1st | ! |
| 2nd | \*,/,% |
| 3rd | +,- |
| 4th | <,>,<=,>= |
| 5th | ==,!= |
| 6th | && |
| 7th | || |
| 8th | = |

Conditional Operator (?):

* (Condition)? Expression if condition is true:expression if condition is false;
* Where : and ? are ternary operators.
* Limitation: The 2 returned values must be of the same type in cpp, if printed directly.

Switch:

* Switch-case is used when we have to make a choice between a number of alternatives for a given variable.

switch(Integer-expression){

case c1:

code;

break;//this break is necessary to get out of the switch if case 1 is satisfied.

case c2:

code;

break;

default:

code;

}

* If break isn’t used: If any case matches, all the rest will be executed as well.
* Char value can be used for cases using ASCII.
* Can use any order for cases.

Q: what will be the output of this program?

int a=10;

if (a=11)

printf(“I am 11”);

else

printf(“I am not 11”):

Solution:

* I am 11, a=11 assigns 11 into an and hence the first printf works.
* If a==11 would have been there the 2nd printf would have been carried out.
* Single statements can be carried out without the curly brackets.
* Every non zero number is treated as true in C language.
* If we use a=0 instead, I am not 11 will be printed.

Type Casting

float l;

int i;

l=32/ (float) i;

Chapter 4: Loop Control Instructions

* I++ or I=+1in C is returns I and adds 1(byte or integer) in it. Similarly for I—, I=-5, ++I, —I.
* Using I operator = number will return I and perform I =I operator number, these are called as compound assigned operators.
* The increments may be read from right to left and hence the output can be very weird.

while(condition){

statements;

}

do{

Statements;

}while(condition);

// This loop runs at least one time.

for(initialise;test;increment){

statements;

}

for(;;){ //usually (initialization; condition; incrementation)

printf("%i\n", i++);

if(i==5) break;

} // This also works and this is very useful for making infinite loops.

The break statement is to be used to get out of a loop.

The continue statement is to be used to skip the rest of an iteration and move to the next iteration.

Note: We are using ; inside for as we can use , for defining more than one variable or condition within the brackets.

How to get random numbers in a c program?

#include<stdio.h>

#include<stdlib.h>

#include<time.h>

int main(){

srand(time(0)); // This seeds the random function using an algorithm which outputs different numbers with different inputs.

printf(“%d”, rand()); //Time at 0 changes time at every second

return 0;

}

Problems and solutions with it:

1. To get the rand function to return within a range use remainder(%).
2. If repeated in the same program, used within a loop or the program runs more than once in a second,then we must delay it, so that the time is changed, and hence the output of the rand() function.

Checked till here.

Chapter 5: Functions and Recursions

#include<stdio.h>

void display(): // function prototype, telling the compiler that we will define it later in the program.

// void denotes that the function returns nothing.

int main(){

int a;

display(); // function call

return 0;

}

void display(){ //function definition

printf(“I am display\n”);

}

Note: If a function is called main is suspended and the control goes to the called function, when the instructions end the control goes back to main.

Functions are used for:

1. To avoid rewriting the same logic again and again.
2. to keep track of what we are doing in a program.
3. to test and check logic independently.

Parameters are the placeholders while the arguments are the actual values.

A function only returns one value but can change more values if given by address.

Chapter 6: Pointers

Chapter 7: Arrays

Array is a collection of similar elements. It’s a sequence of values which are back to back in memory(no gaps or fragmentation).

An array of arrays can be defined by “char asdf[][10];” ; it means that inside the asdf are collections of strings which don’t exceed 10 in count.

Doubt: Can arrays store different data types?

David: short answer: No, long answer: Sort of.

Chapter - 8 Strings

A string is a 1-D character array terminated by a NUL character (‘\0’ (backslash zero)).

A string is always made up of individual char types; they can store integers as char types. But int str[]=“1234” will give you an error and if forced int str[]={1,5,6,'4', '/0'}; this will convert ‘4’ and ‘/0’ into ascii values.

All functions in <string.h> are designed for a character pointer and for an array till a NUL char; having a NUL char in between will make the function think it as the end of the string.

We can always change the NUL character of a string and convert into a non string array, hence the printf function malfunctions, if done so.

They are stored in the contiguous memory and the first pointer is the string name, or the defining pointer.

Initialising strings:

1. char s[]={’T’, ‘A’, ‘B’, ‘I’, ’S’, ‘H’, ‘\0’}; //Can’t be reinitialized.
2. char t[]=“TABISH’; //Can’t be reinitialized.
3. char \*ptr=“TABISH”; // Can be reinitialized any number of times and as much big.
4. char s[30];

\\ can’t use this: s="TABISH”; as this as well can’t be reinitialized.

Understand the difference between 3 and 4.

Printing and Taking strings from the user:

printf(“%s”, str); //Stops the cursor at the end of the string, by default.

puts(str); //Moves the cursor to the next line, by default.

scanf(“%s”, ptr); //Can’t take multi-word string put printf can take strings with spaces.

gets(str); //Can take multi-word string, but one at a time unlike scanf.

Both automatically add the NUL char at the end, if the defined string is long enough.

Library functions for strings:

Using #<include<string.h>

strlen()

strlen() doesn’t count the NUL character while sizeof() does.

If we say string[6], it can store 6 not NUL characters but then it will be treated as a non string array.

strcpy()

This function copies the content of second string into first string passed to it.

strcpy(target, source); // Target now contains the string saved in source.

the target string should have enough capacity to store the source string.

strcat()

This function is used to concatenate two strings and stores it in the first string without space.

strcmp()

This function is used to compare two strings. It returns 0 if strings are equal.

If they are not equal it may compare the ascii value of the first mismatching character by first-second rule or is +1 or -1 found in the same way, this is different depending upon the compilers.

Using char s[] /char \*s as argument for function will change the original string.

Q: Write a program to take string as an input from the user using %c and %s. Confirm that the strings are equal.

#include<stdio.h>

#include<string.h>

int main(){

char b[34], c[34];

int i=0;

scanf("%s", b);

do{

fflush(stdin); // Very important: Had to do so, so that the scanf doesn’t take enter as an input.This flushes the character in stdin.

scanf("%c", &c[i++]);

}while(c[i-1]!='\n');

c[i-1]='\0';

if(strcmp(b,c)==0){printf("YES");};

return 0;

}

Chapter 9: Structures

* Structures hold dissimilar data.
* Defining a structure outside of any function(also outside the main function) allows it to be used anywhere in the program, across different functions, files, or modules.
* Struct employee is like a custom datatype, it can also be used to define a return data type.
* Must use a semicolon after a structure.
* The structure data type stores the values contiguously.
* An array pointer is different from a char pointer even though they both point to a char(the first character of the array and the character that the pointer points to.).
* A structure can be passed into a function like void show( struct employee e320) but if done so is passed by value and not by reference.

Syntax:

#include<stdio.h>

#include<string.h>

struct employee{

int code;

float salary;

char name[10]; //Can't directly assign string, must use strcpy().

};

int main(){

struct employee e1, e2={341, 234.23, "Dsd Gsdw"}, e3={0}, e[30], \*ptr;

e1.code=123;

e1.salary=1324.23;

strcpy(e1.name, "Xyz Abc");

e[5].code=238;

ptr=&e2; //Not (\*ptr)=e3 first, as for this the compiler tries to get \*ptr which is uninitialized.

(\*ptr)=e3; // can also use arrow operator: (\*ptr).code is same as ptr->code

//(\*ptr).code=999; Can also use this.

printf("%d\n", (\*ptr).code);

printf("%s\n", e1.name);

return 0;

}

typedef

Used to create an alias name for data types in c and is more commonly used with structures.

typedef struct complex {

float real;

float img;

} CNo; // CNo replaces the whole “struct complex”.

Chapter 10: File I/O

To save things permanently.

File pointer

File is a structure which needs to be created for opening the file.

A file pointer is a pointer to this structure of the file.

Can also read a .c file

File\*ptr; //FILE is a structure

ptr=open(“filename.extension”, “mode”);

| r | Open for reading | If the file does not exist, fopen returns Null |
| --- | --- | --- |
| rb | Open for reading in binary |
| w | Open for writing | If the file exists the contents will be overwritten |
| wb | Open For writing in binary |
| a | Open for append | If the file does not exist, it will be created |

Types of Files

There are 2 types of files:

1. Text files(.txt, .c)
2. Binary files(.jpg, .dat)

Reading a file

Instead of scanf we use fscanf(ptr, “%d”, &num); //this will read an integer from file in num variable

fclose(ptr); This tells that we are done working with this file and the associated resources culd be freed.

ptr=fopen("Tabish.txt", "r");

if(ptr!=NULL){

Can be used to read the file if and only if it exists.

Writing a file

fprintf(fptr, “%d”, num);

fgetc() and fputc() are used to read and write a character form/to a file respectively.

Remember the pointer continues to shift from left to right with reading.

If continue to read : End of file error

fgets() returns EOF when all the characters from a file have been read So we can write a check like below to detect end of file

Chapter 11: Dynamic Memory Allocation

ctype.h

Using toupper(char) will turn the lower to upper and leave the upper characters as they are.

Similarly for tolower() and so on.