Let Us C by Yashavant Kanetkar

> "Four important aspects of any language are the way it stores data, the way it operates upon this data, how it accomplishes input and output, and how it lets you control the sequence of execution of instructions in a program."

32 Keywords available in C:

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

---

> "main() is a function. A function is nothing but a container for a set of statements."

Priority	Operators	Descriptions
1st	* / %	Multiplication, Division, Modular division
2nd	+ -	Addition, Subtraction
3rd	=	Assignment

"When an expression contains two operators of equal priority the tie between them is settled using the associativity of the operators. All operators in C have either *Left* to *Right* associativity or *Right* to *Left* associativity."

```
/* Consider the expression: */
a = 3 / 2 * 5
```

"Here there is a tie between operators of same priority. This tie is settled using the associativity of / and . ***Both enjoy*** **Left** ***and*** **Right** ***associativity. Therefore firstly / operation is done followed by*** .

"Note that in C a non-zero value is considered to be true, whereas a O is considered to be false."

"Remember that in C, 'truth' is always non-zero"

```
/* first question */
int a = 10;
!(a < 10) /* value 1 */

/* second question */
/* Why parenthesis */
i >= 5 ? j = 10 : j = 15;
/* find what is confusing for the compiler */
```

"Almost always, if something is worth doing, it's worth doing more than once"

"Upon reaching the closing brace of for, control is sent back to the for statement, where the value of **count** gets incremented by 1."

"We often come across situations where we want to jump out of a loop instantly, without waiting to get back to the condition. The keyword *break* allows us to do this. When *break* is encountered inside any loop, control automatically passes to the first statement after the loop."

"In some programming situations, we want to take the control to the beginning of the loop, bypassing the statements inside the loop, which have not yet been executed. The keyword *continue* allows us to do this. When continue is encountered inside any loop, control automatically passes to the beginning of the loop"

"The usage of the *goto* keyword should be avoided as it usually disturbs the normal flow of execution."

"On executing the return statement, it immediately transfers then control back to the calling function"

"A function can return only one value at a time. There is a way to get around this limitation, pointers…"

```
int main() {
    int a, b, c;
    sum = calsum(a, b,c);

    return 0;
}

int calsum(int x, int y, int z) {
    int d;
    d = x + y + z;
    return d;
}
```

In the above code, the varaibles a, b, and c are called *Actual Arguments*, the variable x, y, and z are called *Formal arguments*

This means that when values are passed to a called function, the values present in actual arguments are not physically moved to the formal arguments; just a photocopy of values in actual argument is made into formal arguments.

> "The value of each of the actual arguments in the calling function is copied into the corresponding formal arguments of the called function."
>
> "In C order of passing arguments to a function is from right to left"
>
> "If you observe carefully, whenever we called a function and passed something to it we have always passed the 'values' of variables to the called function. Such function calls are called 'calls by value'."
>
> "We have also learnt that variables are stored somewhere in memory. So instead of passing the value of a variable, can we not pass the location number of the variable to a function? If we were able to do so, it would become a 'call by reference' "
>
> "A stack is a last in first out (LIFO) data structure. This means that the last item to get stored on the stack (often called push operation) is the first one to get out of it (often called pop oper-

ation).

Recursive step

- Decompose, extract(condition) and reduce.
- Touch the base
- Operation part: previous extractation(condition) and the targeted operation. Were computation take place.

  "… the primary data types themselves can be of several types. For example, a char can be an **unsigned char** or a **signed char**. Or an **int** can be a **short int** or a **long int**." "To fully define a variable, one needs to mention not only its type but also its storage class."

- 16-bit compiler Turbo C the range is -32768 to 32776 (two/bytes used to store)

- 32-bit compiler like Visual Studio or gcc range id -2147483648 to +2147483648 (four/bytes)

  "Remember that out of the two/four bytes used to store an integer, the highest bit (16th/32nd bit) is used to store the sign of the integer. This bit is 1 if the number is negative and 0 if the number is positive."

  "Sometimes, we come across situations where the constant is small enough to be an **int**, but still we want to give it as much storage as a **long**. In such cases, we add the suffix 'L' or 'I' at the end of the number, as in 23L."

  "A **signed char** is same as an ordinary **char** and has a range from -128 to +127; whereas, an **unsigned char** has a range from 0 to 255."

  "A **float** occupies four bytes in memory and can range from -3.4e38 to +3.4e38. If this is insufficient, then C offers a **double** data type that occupies 8 bytes in memory and has a range from -1.7e308 to +1.7e308."

  "We have already said all that needs to be said about constants, but we are not finished with variables. To fully define a variable,

one needs to mention not only its 'type' but also its 'storage class'. In other words, not only do all variables have a data type, they also have a 'storage class'."

"There are two kinds of locations in a computer where such value may be kept - Memory and CPU registers. It is the variable's storage class that determines in which of these two types of locations, the value is stored."

Four storage classes in C:

- Automatic storage class
- Register storage class
- Static storage class
- External storage class

"All variables that are defined inside a function are normally created on the stack each time the function is called. These variables die as soon as control goes back from the function. However, if the variables inside the function are defined as static then they do not get created on the stack. Instead they are created in a place in memory called 'Data Segment'. Such variables die only when program execution comes to an end."

"If a variable is defined outside all functions, then not only is it available to all other functions in the file in which it is defined, but also available to functions defined in other files. In the other files the variable should be declared as extern."

"We can make use of proper storage classes like **auto**, **register**, **static**, and **extern** to control four properties of a variable -storage, default initial value, scope, and life."

"The C preprocessor is exactly what its name implies. It is a program that processes our source program before it is passed to the compiler. Preprocessor commands (often known as directives) form what can almost be considered a language within C language."

"For debuging macro, view the expanded source code with cpp source.c output.l"

> "#include feature is used when creating our own library of functions which we wish to distribute to others. In this situation the functions are defined in a".c" file and their corresponding prototype declarations and macros are declared in a ".h" file".

- #include "mylib.h"
  - This command would look for the file **mylib.h** in the current directory as well as the specified list of directories as mentioned in the include search path that might have been set up
- #include <mylib.h>
  - This command would look for the file mylib.h in the specified list of direcotires only

Array

> "An array is also known as a subscripted variable"
>
> "... This ability to use variables to represent subscripts is what makes arrays so useful"

```
int arr[8];
```

What happens in memory when we make this declaration ? `32 bytes`get immediately reserved in memory, `4` bytes for the 8 integers. All the array elements would always be present in contiguous memory locations.

> "... when the integer pointer x is incremented, it points to an address four locations after the current location, since an *int* is 4 bytes long."
>
> "When you subtract one pointer from another in C, the operation internally calculates the difference in bytes between the two memory addresses and then divides that difference by the size of the data type the pointers are pointing to. This division adjusts the byte difference to a difference in terms of array elements. [CHATGPT]"

String

A string constant is a one-dimensional array of characters terminated by a null ('\0')

Each character in the array occupies 1 byte of memory and the last characteris always '\0'.

The terminating null ('\0') is important, because it is the only way the functions that work with a string can know where the string ends. In fact, a string not terminated by a '\0' is not really a string, but merely a collection of charaters.

```
char name[] = {'H', 'A', 'E', 'S', 'L', 'E', 'R', '\0'};
/* short cut for initializing strings */
char name[] = "HAESLER";
/* C inserts the null character automatically.

/* Let us understand the difference between the following two statements: */
/* Here str acts as a constant pointer to a string, whereas,
 * p acts as a pointer to a constant string.
 */
char str[] = "Quest";
char *p = "Quest";

str++; /* error, constant pointer cannot change */
*str = 'Z'; /* works, because string is not constant */
p++; /*works, because pointer is not constant */
*p = 'M'; /* error, because string is constant */
```

> "Though in principle a 2-D array can be used to handle several strings, in practice an array of pointers to strings is preferred since it takes less space and is efficient in processing strings."
>
> "malloc() function can be used to allocate space in memory on the fly during execution of the program"

Structures

> "A structure gathers together, different atoms of informations that comprise(make up) a given entity"

- (a) it is important to understand that a structure type declaration does not tell the compiler to reserve any space in memory. All a structure declaration does is, it defines the 'form' of the structure

- (b) Usually structure type declaration appears at the top of the source cpcode
- (c) If a structure variable is initialed to a value {0}, then all its elements are set to value 0. This is a handy way of initializing structure variables.

```
struct book {
    char name[10];
    float price;
    int pages;
};

struct book b1 = {"Basic", 130.00, 550};
struct book b2 = {"Physics", 150.80, 800};
struct book b3 = {0};
```

Input/Output

There are numerous library functions available for I/O. These can be classified into two broad categories:

- (a) Console I/O functions:
  - functions to receive input from keyboard and write output to VDU(screen).
- (b) File I/O functions:
  - functions to perform I/O operations on a floppy disk or a hard disk.

"The screen and keyboard together are called a console. Console I/O functions can be further classified into two categories - formatted and unformatted console I/O functions.

Unformatted console I/O functions

So far, for input we have consistently used the **scanf()** function.However, for some situations, the **scanf()** function has one glaring weakness… you need to hit the Enter key before the function can digest what you have typed. **getch()**_ and **getche()**

are two functions which serve this purpose. These functions returns the character that has been most recently typed. The 'e' in ___getche() function means it echoes(displays) the character that you typed to the screen.

- There is no keyword available in C for doing input/output
- All I/O in C is done using standard library functions

```
#include <stdio.h>
void xgets(char *str) {
    scanf("%[^\n]s", str);
}

void xputs(char *str) {
    printf("%s\n", str);
}
```

- `[...]` is a scanset, telling scanf to read a sequence of characters that match the characters specified between the brackets
- `^` the caret the reverse (negated)
- `^\n` any character except newline character

File Input/Output

"If there are multiple exit points in the program, then the value passed to exit() can be used to find out from where the execution of the program got terminated"

"The prototype of exit() function is declared in the header file stdlib.h"

"On closing the file, the buffer associated with the file is removed from memory."

"What scanf() does is it assigns name, age and salary to appropriate variables and keeps the Enter key unread in the keyboard buffer."

Types of buffering:

- Fully Buffered: Data is stored in the buffer and only written out when thebuffer is full. This is common for file operations.
- Line Buffered: Data is buffered until a newline character is encountered, which is typical for terminal output. This means that if you print a line without a newline character, it might not appear immediately on the terminal.
- Unbuffered: Data is written out as soon as it's available. `stderr` is often unbuffered by default, ensuring error messages are displayed immediately.

Two main areas where text and binary mode files are different: - (a) Handling of newlines - (b) Storage of numbers

The only function that is available for storing numbers in a disk file is the fprintf() function. It is important to understand how numerical data is stored on the disk by fprintf(). Text and characters are stored one character per byte, as we would expect. Are numbers stored as they are in memory, 4 bytes for an integer, 4 bytes for a float, and so on? No.

Numbers are stored as strings of characters. Thus, 12579, even though it occupies 4 bytes in memory, when transferred to the disk using fprintf(), would occupy 5 bytes, 1 byte per character. Similarly, the foating-point number 1234.56 would occupy 7 bytes on disk. Thus, numbers with more digits would require more disk space.

> "The `sizeof()` operator gives the size of the variable in bytes."
>
> "An operandis a term used to describe any object that is manipulated by an operator. In simpler terms, operands are the values or variables that operators act upon"
>
> "Unary operators have one operand (`!condition`, `-x`)"

```c
int condition = 0; /* false in C */

/* the operator works by taking the value of `condition`,
 * inverting its truthiness, and producing a result
 */
if (!condition) {
    printf("we passed in, condition is false, !condition is true.\n");
```

```
else {
    printf("condition is true, that the reverse so here we go passed through the r
}
```

Operations on bits

> "The smallest element in memory on which we are able to operate as yet is a byte; and we operated on it by making the use of the data type char."
>
> "Being able to operate on a bit-level, can be very important in programming, especially when a program must interact directly with the hardware. This is because, the programming languages are byte-oriented, whereas hardware tens to be bit-oriented"

Operator	Meaning
~	One's complement
>>	Right shift
<<	Left shift
&	Bitwise AND
|	Bitwise OR
^	Bitwise XOR

bitwise AND operator

&	0	1
0	0	0
1	0	1

- AND operator is used in two situations:
  - (a) To check whether a particular bit of an operand is ON or OFF
  - (b) To turn OFF a particular bit

(b) putting 3rd bit off

```
10101101 Original bit pattern
11110111 AND mask
```

```
--------
10100101
```

bitwise OR operator

0	0 1
1	1 1

```
11010000 Original bit pattern
00000111 OR mask
--------
11010111
```

Bitwise OR operator is usually used to put ON a particular bit in a number.

bitwise XOR operator

^	0 1
0	0 1
1	1 0

XOR operator is used to toggle (change) a bit ON or OFF.

A number XORed with another number twice gives the original number.

Summary for bit operation:

- The bitwise operators include operators like one's complement, right-shift, left-shift, bitwise AND, OR, and XOR.
  - The one's complement converts all 0s in its operand to 1s and all 1s to 0s
  - The right-shift and left-shift operator are useful in eliminating bits from a number -either from the left or from the right.
  - The bitwise AND operators is useful in testing whether a bit is on/off add in putting off a particular bit
  - The bitwise OR operator is used to turn on a particular bit.
  - The XOR operator works almost sams as the OR execpt one minor variation, `1 ^ 1 = 0`, great for toggling bits.

Miscellaneous Features

- Functions with variable number of arguments:

  "In C, `va_start`, `va_arg` and `va_list` are part of the `stdarg.h` header and are used for handling variable function arguments"

  "These macros allow functions to accept an indefinite number of arguments, providing the flexibility to work with functions where the number and types of arguments are not know in advance"

- Unions:

  "Both structures and unions are used to group a number of different variables together. But while a structure enables us to treat a number of different variables stored at different places in memory, a union enables us to treat the same space in memory as a number of different variables"

- typedef:

```
typedef struct book [
    char name[10];
    float price;
    int pages;
} BK;

BK b1, b2, b3;
```

"Communication is the essence of all progress"