C PROGRAMMING NOTES BY NEHA MARUYA

What is programming.?

Computer programming is a medium for us to communicate with computer. Just like we use 'Hindi', 'or', 'English' to communicate with each other. Programming is a way for us to deliver our instruction to the computer.

What is C.?

C is a programming language. C is one of the oldest and finest programming language. C was developed by "Dennis Ritchie" at Bell Labs, USA in 1972. C is a middle level language. C supports function programming.

Scanf() :- used for input.

Printf() :- used for output.

File extension in C is .c

C is subset of C++.

Use of C

C language is used to program a wide variety of systems. Some of the use of c are as follows:

- Major part of windows, Linux and other operating system are written in C.
- C is used to write driver programs for device like Tablets, printers etc.
- C Language is used to program embedded where programs need to run faster in limited memory. (Microwave, Comers etc.)
- C is used to develop game an area where latency is very important ie .Computer has to react quickly on user input.

Variables

A variable is a container which stores a 'value'. In Kitchen , we have containers storing Rice, Sugar etc. Similar to that variables in C stores value of a constant.

Example:

```
a = 3;        //a is assigned "3"
b = 4.7;      //b is assigned "4.7"
c = 'A';      //c is assigned 'A'
```

A variable is an entity whose value can be changed.

Rule for naming variables in C

There are many rules:

- First character must be an alphabet or underscore (_)
- No commas, blanks allowed.
- No special symbol other than (_) allowed.
- Variable names are case sensitive.

We must create meaningful variable names is our programs. This enhances readably of our programs.

Constants

An entity whose value doesn't change is called as a constant.

Type of constants

1. Integer Constant //-1, 2, 4, 7, 9, etc.
2. Real Constant // -3.5, 5.5,2.0
3. Character Constant//'a', '$', '@' (must be enclosed within single inverted commas)

Keywords

These are reserved words, whose meaning is already known to the compiler there are 32 keyword 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

Our first program

```
#include<stdio.h>
Int main()
{
Printf("Hello  I am learning C ");
Return 0'
}
```

Basic Structure of a C program

All C programs have to follow a basic structure.

A C program starts with a main function and executes instructions present inside it.

Each instruction is terminated with a semicolon (;).

There are some rules which are applicable to all the C programs:

1. Every program's execution starts from main() function.
2. All the statements are terminated with a semicolon.
3. Instructions are case-sensitive.

4. Instructions are executed in the same order in which they are written.

Comments

Comments are used to clarify something about the program in plain language. It is a way for us to add notes to our program. There are two types of comments in C.

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

Comments in a C program are not executed and are ignored.

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.

A C program is written in plain text.

This plain text is combination of instruction in a particular sequence. The compiler performs some basic checks and finally compiler performs program into an executable.

first.c => C compiler =>first.exe (There are first.c input ant first.exe output)

Library Function

C language has a lot of valuable library functions which is used to carry out certain tasks. For instance printf function is used to print values on the screen .

Printf("this is %d", i);

%d for integers

%f for real values

%c for characters

Type of variables

1. Integer variable //int a=3;
2. Real variables // float a=7.7;
3. Character variables //char a='B';

Receiving input from the user

In order to tale input from the user and assign it to a variable, we use scanf function .

Syntax for using scanf :

Scanf("%d", &i); // This & is important !

& is the "address of" operator and it means that the supplied value should be copied to the address which is indicated by variable i.

Instructions and Operators

A C program is a set of instructions. Just like a recipe-which contains instructions to prepare a particular Dish.

Types of instructions

1. Type declaration instruction
2. Arithmetic instruction
3. Control instruction

Type declaration instruction

Int a;

Float b;

Other variation :

int I=10; int j= i; int a=2

int j= a + j – I ;

float b=a+3; float a=1.1 (Error! as we are typing to use a before defining it.)

int a, b, c, d;

a=b=c=d=30; (Value of a, b, c & d will be 30 each.)

Arithmetic Instructions

int i =(3*2) + 1 (there are * and + are operators and 3, 2 and 1 are operands)

operands can be int/float etc.

+ - */are arithmetic operators

Int b =2, c=3;

Int z; z=b*c; legal

Int z; b*c=z; Illegal (not allowed)

% => Modular division operator

% => Returns the remainder

% => Cannot be applied on float

% =>Sign in same as of numerator(-5%2=-1)

5%2=1 -5%2=-1

Note :--

1. **No operator is assumed to be present**

Int i =ab //invalid

Int i =a*b // valid

2. There is no operator to preform exponentiation in C
   however we can use pow(x, y) form <math.h>(more latss)

Type Conversion

An Arithmetic operation between

int and int =>int

Int and float =>float

float and float =>float

important!!

5/2 =>2 5.0/2 =>2.5

2/5=>0 2.0 =>0.4

Note :--

Int a=3.5; //In this case 3.5(float) will be demoted to 3(int) because a is not able to store floats.

Float a=8; // a will store 8.0

8 =>8.0 (promotion to float)

Operator precedence In C

3 * x -8y is (3x)-(8y) or 3(x – 8y) ?

In C language simple mathematical rules like BODMAS, no longer Applies.

The answer to the above question is provided by operator precedence & associativity.

Operator Precedence:

The following table lists the operator priority in c.

Priority	Operators
1st	* / %
2nd	+ -
3rd	=

Operators of higher priority are evaluated first in the absence of parenthesis.

Operator Associativity:

When Operators of equal priority are present in an expression, the fie is taken care of by associativity.

X * y / Z => (X *y) / Z

X / Y * Z => (X/Y) * Z

*, / follows left to right associativity

Control instruction :

Determines the flow of control in a program four types of control Instruction in are:

1. Sequence Control Instruction
2. Decision Control Instruction
3. Loop control Instruction
4. Case Control Instruction

Conditional Instructions

Sometimes we want to watch comedy videos on YouTube if the day is Sunday.

Sometimes we order junk if it is our friend's birthday in the hostel.

You might want to busy an Umbrella if its raining and you have the money.

You order the meal if dal or your favorite bhindi is listed on the menu.

All these are decisions which depends on a condition being met.

In C language too, we must be able to execute instructions on a condition (s) being met.

Decision making instruction in C

1. If-else statement
2. Switch statement

If -else statement

The syntax of an if-else statement in C looks like:

```
If (condition to be checked)
{
Statements – if – condition - false;
}
else {
statements – if – condition – false;
}
```

Code example :

```
Int a =23;
If (a>18)
{
printf(“you can drive \n”);
}
```

Note: that else block is not necessary but optional.

Relational operators in C

Relational operators are used to evaluate conditions (true or false) inside the if statements.

Some examples of relational operators are:-

= =, >=, >, <, <=, !=

Important note :- ‘=’ is used for assignment where as ‘==’ is used for equality check.

The condition can be any validexpression . In C a non – zero value is considered to be true.

Logical operators

&&, ||, and ! are three logical operators in C .

There are read as “AND”, “OR” and “NOT”. They are used to provide logic to our C programs.

Usage of Logical Operators:

(i) && -> AND -> is true where both the conditions are true
“1 and 0” is evaluated as false.
“0 and 0” is evaluated as false.
“1 and 1” is evaluated as false.

(ii) || -> OR -> is true when at least one of the conditions is true.
(1 or 0 ->1)(1 or 1->1)

(iii) ! -> return true if given false and false if given true

! (3==3) -> evaluates to false

! (3 >30) -> evaluates to true.

As the number of condition increases, the level of indentation increases. This reduces readability. Logical operators come to rescue in such cases.

Else if clause

Instead of using multiple if statements , we can also use else if along with if thus forming an if -else -if -else ladder.

```
If {
//statements;
}
Else if  {
……
}
Else {
…..
}
```

Using if -else if -else reduces indents the last “else” is optional.

Also there can be any number of “else if”

Last else is executed only if all conditions fail.

Operator Precedence

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

Conditional Operators

A short hand “if -else” can be written using the conditional or ternary operators.

Condition ? expression -if true : expression -if -false

Switch case control instruction

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

```
Switch (integer-expression)
{
Case C1:
        Code;
Case C2:
        Code;
Case C3:
        Code;
default:
        code;
}
```

The value of integer-expression is matched against C1, C2, C3,…… If it matches any of these cases, that case along with all subsequent “case” and “default” statements are executed.

Important Notes

1. We can use switch-case statements even by writing cases in any order of our choice (not necessarily ascending)
2. Char values are allowed as they can be easily evaluated to an integer
3. A switch can occur within another but in practice this is rarely done.

Loop Control Instruction

Why Loops

Sometime we want our programs to execute few set of instruction over and over again. For ex:

Prnting 1 to 100, first 100 ever numbrs etc.

Hence loops make it easy for a programmer to tell computer that a given set of instructions must be executed repeatedly.

Type of loops

Primarily, there are three types of loops in C language:

1. Entry Control loop:--
   a. For loop
   b. While loop
2. Exit control loop:--
   a. Do-while loop

For loop

for loop is entry control loop. It is used to execute a statement multiple time.

Each loop condition three parts:

Initialization: starting point of loop

Condition: ending point of loop

Updation: steps /increment or decrement

Syntax:

```
For(initialization : condition : updation):
{
//statement;
}
```

Example:

```
For(i=0 ; i<3 ; i++)
{
Printf("%d", i);
Printf("\n");
}
```

Output:

```
0
1
2
```

While loop

While loop is a entry control loop where firstly given condition with loop is checked and if given condition is true then only block of loop will execute.

Syntax:

```
Initialization;
While (condition)
{
//updation;
//statement;
}
```

⇨ the block keeps executing as long as the Condition is true.

An Example

```
Int I = 0                  // initialization
While(i<10)          // condition
{
Printf (“The value of I is %d”, i );
i++;                       // updation
}
```

Note :

If the condition never becomes false, the while loop keeps getting executed. Such a loop is known as infinite loop.

Quick Quiz:

Write a program to print natural numbers from 10 to 20 when initial loop counter i is initialized to 0.

The loop counter need not be int, it can be float as well.

Increment and decrement operators:

i++ → I is increased by 1

i-- => I is decreased by 1

printf("- -i=%d", --i);

this first decrement I and then prints it

printf(" i-- =%d", i- -);

this first I and then decrements it

- +++ Operator does not exist => Important
- += is compound assignment operator
  Just like - =, * =, / = & % = =>Also Important

do – while – loop

do -while loop is a exit control loop. That means in do – while given condition is checked after execution of block.

That means if given condition is false at starting then also the loop will execute for one time for sure.

Syntax :--

```
Initialization
do
{
// statement
//updation
}
While(condition);
```

Note:

While => checks the condition & then executes the code

do – while => execute the code & then checks the condition

Nested loop :--

When you use one loop within another loop then this structure if programming is know as nested loop.

Syntax :

```
Outer loop:
    For ( initialization ; condition ; updation )
    {
Inner loop :
        For (initialization ; condition ; updation )
        {
        //inner loop statement
        }
    //outer loop statement
    }
```

Example 1:

```
#include<stdio.h>
void main()
{
        int i, k;
        for(i=1; i<=4; i++)//i=1
        {
                for(k=1; k<=i; k++)//k=1
        {
                        printf("*");
        }
        printf("\n");
        }
}
```

Output :

```
*
**
***
****
```

Example 2:

```
#include<stdio.h>
void main()
{
        int i, k;
        for(k=5;k>=1;k--)
        {
        for(i=1;i<=k;i++)
        {
                printf("%d",k);
                }
                printf("\n");
        }
}
```

Output:

```
55555
4444
333
22
1
```

Conditional statement

Conditional statement are those statements of programs where given instructions in executed based on the condition result.

1. If statement
    a. Simple if statement
    b. If else statement
    c. Ladder – if statement
    d. Nested if statement
2. Switch statement

Exampal :--

```
#include<stdio.h>
Void main()
{
        int  a=15;
        if (a>0) //true
        {
        Printf(“ohh ! this is a positive number”);
        }
        if (a>10) //false
        {
        Printf(“and it is greater then 10”);
        }
}
```

Note :--If condition is true , first statement will execute, if condition is false the second statement is execute.

Simple if statement :--

The program where only one if statement is given and the block of if statement will execute only if the given. Condition with if is true.

Syntax :--

```
If (condition)
{
//statement;
}
```

If – else statement :--

The program where a single condition is given and two different statements is given, one with if block and second with else block.

If given condition is true then block will execute otherwise else block will execute.

Syntax :--

```
If (condition)
{
//statement
}
else
{
//statement
}
```

Switch Statement

Switch statement is used to execute different statement based on the sign single value of variable.

It is the optional of else – if lader.

Syntax :--

```
Switch (val/exp)
{
Case value ;
//statement
Break;
}
```

Example of switch statement :--

```
Void main()
{
Int n1, n2, val, res;
Printf("enter two number:");
Scanf("%d%d", &n1, &n2);
Printf("enter(1-add, 2-subtract, 3-multi, 4-division, 5-reminder :)");
Scanf("%d", &val);
Switch (val)
{
Case1:
```

```
res=n1+n2;
Printf(“addition result=%d”, res);
Break;
Case2:
res =n1+n2;
printf(“subtract result =%d”, res);
break;
case3:
res =n1*n2;
printf(“mul result =%d”, res);
break;
case4:
res =n1/n2;
printf(“result is =%d” , res);
break;
case5:
res =n1%n2;
printf(“remender result =%d”, res);
break;
default;
printf(“invalid number”);
}
}
```

Branching Statement:--

Branching statement are those statements which is used to transfer control from one place to another.

There are three type of branching statement :---

1. Break;
2. Continue;
3. Go to;

Break ; -- ☹ It is only used with looping statement.

Continue; --😊 Continue statement is used within looping statement. Executed within a loop then terminates the current iteration and continue with next iteration.

Go to;--☹ Go to statement is used to transfer control from one place to another by using label.

Function and Recursion

Something our program get bigger in size and its not possible for a programmer to track which price of code is doing what.

Function is a way to break our into (chunks) so that it is possible for a programmer to reuse them.

What is a function ?

A function is a block of code which performs a particular task. A function can be reused by the programmer in a given program any number of times.

Reusability of code is called function.

Example and Syntax of a function :

```
#include<stdio.h>
void display();      =>Function prototype
int main (){
        Int a;
display();           =>function call
return 0;
}
void display(){
printf("hi I am display");
}
```

Function prototype

Function prototype is a way to tell the compiler about the function we are going to define in the program. Here void indicates that the function returns nothing.

Function call

Function call is a way to tell the compiler to execute the function body at the time the call is made.

Note that the program execution starts from the main function is the sequence the instructions our written.

Function definition

This part contains the exact set of instructions which are executed during the function call. When a function is called from main(), the main function false asleep and gets temporarily suspended. During this time the control goes to the function being called. When the function body id done executing main() resumes.

Type of functions

- Library function→ Commonly required functions ground together in a library file on disk
- User defined functions→ these are the functions declared and defined by the user.

Why use function ?

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

Passing values to functions

We can pass values to a function and can get a value in return from a function.

Int sum (int a, int b)

The above prototype means that sum in a function which takes values a (of type int) and returns a value of type int

Function definition of sum can be :

```
Int  sum(int a, int b)
{
Int c;
C=a+b;
Return c;
}
```

Now we can call sum (2, 3); from main to get 5 in return.

Int d =sum (2,3);

Note:

- Parameters are the value or variable placeholder in the function definition .
  Ex. a & b.
- Arguments are the actual values passed to the function to make a call.
  Ex . 2 & 3
- A function can return only one value at a time .
- If the passed variable is changed inside the function, the function call doesn't change the value in the colling function

```
int change (int a)
{
A=77;
```

```
Return 0;
}
```

Change is a function which changes a to 77. No if we call it from main like this .

```
int b =22
change(b);
printf("b is %d", b);
```

This happens because a copy of b is passed to the change function

Recursion→

A function defined in C can call itself.

This is called recursion.

A function calling itself in also called 'recursive' function.

Example of Recursion

A very good example of recursion is factorial .

Factorial(x) =1*2*3…….*x

Factorial(x) =1*2*3 …..x-1 *x

Factorial(x) = factorial (x-1) *x

Since we can write factorial of a number in terms of itself, we can program it using recursion.

```
int factorial (int x)
{
Int f ;
If (x==0 || x==1)
```

```
 Return 1;
else
  f = x * factorial(x-1);
return f;
}
```

How does it work?

Factorial (5)

5	x	factorial(4)						
5	x	4	x	factorial(3)				
5	x	4	x	3	x	factorial(2)		
5	x	4	x	3	x	2	x	factorial(1)
5	x	4	x	3	x	2	x	1

Important Notes:

1. Recursion is sometimes the most direct way to code an algorithm
2. The condition which doesn't call the function any further in a recursive function is called as the base condition.
3. Sometimes, due to a mistake made by the programmer, a recursive function can keep running without returning resulting in a memory error.

Pointers

A pointer is a variable which stores the address of another variable

By using pointer, you can directly work on the memory address of variable.

& and * is two important symbols using in. where & denotes the memory address of a variable and * can denotes the stored in a variable.

Int *p;

Here p is a pointer type variable, which can store memory address a integer type variable.

f is a pointer

f pointe to i

The “address of “ (&) operator

The address of operator is used to option the address of a given variable.

If you refer to the diagrams above

& I => 87994

& f => 87998

Format Specifier for printing pointer address is ‘%u’

The ‘value at address’ operator (*)

The value at address or * operator is used to option the value present at a given memory address. It is denoted by *

*(&i) =72

*(&f) =87994

How to declare a pointer?

A pointer is declared using the following Syntax—

Int * f; →declare a variable f of type int-pointer

f = &I → Store address of I in f.

just like pointer of type integer, we also have pointers to char, float etc.

int *ch_ptr; => pointer to integer

char *ch_ptr; =>pointer to character

float * ch_ptr; => pointer to float

Although it's a good practice to use meaningful variable names, we should be very careful while reading & working on programs for fellow programmers.

A Program to demonstrate pointers

```
#include<stdio.h>
int main(){
    int i=8;
    int *f;
    printf ("Add I =%u\n",&i);
    printf("Add f =%u\n",&f);
    printf("Add f = %u\n",&f);
    printf ("Value i =%d\n", i);
    printf("Value I = %d\n", i);
    printf("Value I =%d\n",*(&i));
    printf("Value I =%d\n",*f);
    return 0;
    }
```

Output:

Add i = 87994

Add i = 87994

Add f = 87998

Value i = 8

Value i = 8

Value i = 8

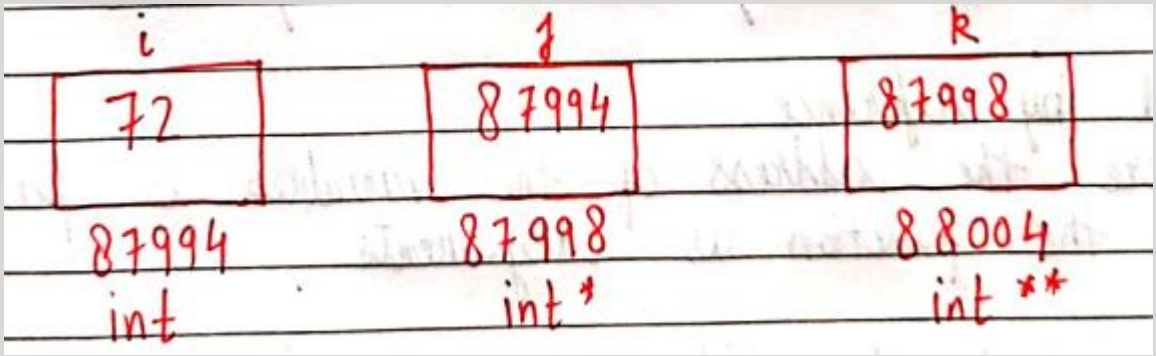
The program sums it all. If you understand it, you have got the idea of pointers.

Pointer to a pointer

Just like if is pointing to I or storing the address of I, we can have another variable k which can further store the address of j. what will be the type of k.

Int **k;

K=&j;



We can even go further one level and create a variable l of type int *** to store the address of k. we mostly use int* and int** sothings in real word programs.

Type of function calls

Based on the way we pass arguments to the function, function calls are of two types.

1. **Call by value** →Sending the value of arguments
2. **Call by reference** →Sending the address of arguments

Call by value

Here the value of the arguments are passed to the function, consider this example:

Int C= sum (3,4); →assume x=3 and y=4

If sum is defined as Sum (int a, int b), the values 3 and 4 are copied to a and b Now ever if we change a and b, nothing happened to the variables x and y.

This is call by value.

In C we usually make a call by value.

Call by reference

Here the address of the variables is passed to the function as arguments.

Now since the address are passed to the function, the function can now modify the value of a variable in calling function using of a variable in calling function using * and & operators Example:

```
Void swap (int *x, int *y)
{
Int temp;
Temp= *x;
Temp= *y;
*x= *y;
*y=temp;
}
```

This function is capable of swapping the values passing to it. If a=3 and b=4 before a call to swap (a, b), a=4 and b=3 after calling swap.

```
Int main()
{
int a=3
int a=4   →a is 3 and b is 4
swap (a, b)
return 0;    →Now a is 4 and b is 3
}
```

Arrays

Array is a collection of homogeneous (with same data type) data elements. It stores multiple values at different index of array. Index of array always starts from 0 and last index of array is size 1. Array stores data in memory sequentially.

Syntax:

Data type Array name[size];

Example:

Int marks [90]; →integer array

Char name [20]; →character array or string

Float percentile [90]; →float array

Type of Array

1. **One dimensional array**
2. **Two-dimensional array**

Initialization of on array

There are many other ways in which an array can be initialized.

int cgpa [3] ={9, 8, 8} =>Array can be initialized while declaration

float marks []={33,40}

Arrays in memory

Consider this array:

Int arr[3]={1, 2, 3} =>1 integer =4bytes

This will reserve 4x3=12 byte in memory 4 byte for each integer.

==> arr in memory

Pointer Arithmetic

A pointer can be incremented to pointer to the next memory location of that type.

Consider this example

int i =32;

int *a =&i; → a=8794

a++; → Now a =8795

char a='A';

char *b=&a; →b=8794

b++; →now b=8795

float I =1.7;

float *a =&I; →Address of I or a =8794

a++; →Now a =8795

Following operation can be performed on a pointer :

1. Addition of a number to a pointer.
2. Subtraction of a number from a pointer.
3. Subtraction of one pointer from another.
4. Comparison of two pointer.

Note:

If ptr points to index 0, ptr ++ will point to index 1 & so on …..

This way we can have an integer pointer pointing to first element of the array like this :

Int *ptr =&arr[0];

Ptr ++;

*ptr → will have 1 as its value

Passing arrays to functions

Arrays can be passed to the functions like this …

PrintArray(arr, n); → function call

Void printArray(int*i, int n); →function prototype

Void printArray(int i[], int n);

Multidimensional Arrays

A array can be of 2 dimension /3 dimension /n dimensions..

A 2 dimension array can be defined as :

Int arr[3] [2] ={ {1,4}

{7,9}

{11, 22} };

We can access the element of this array as arr [0] [0], [0] [1] & so on…

2-D Arrays in Memory

A 2d array like a 1-d array is stored in contiguous memory blocks like this :

arr[0][0] arr[0][1]…….

Strings

String is a collection of multiple characters. In C directly you can not declare a variable with string data type.

Whenever you need to store a string value in memory you have to declare a array of initialize character.

A string is a 1-D character array terminated by a null ('\0')

('\0')→ this is null character

Null character is used to denote string termination characters are stored in contiguous memory locations

Initializing Strings

Since string is an array of characters, it can be initialized as follows:

char s[]= {'H', 'A', 'R', 'R', 'Y', '\0'};

There is another shortout for initializing strings in C language:

char 5[] = "Neha"; →In this case C adds a null character automatically.

Printing strings

A string can be printed character by character using printf and %c. But there is another convenient way to print strings in C.

char st[]="Neha";

printf ("%s", st); →prints the entire string.

Taking string input from the user

We can use %s with scanf to take string input from the user:

Char st[50];

Scanf("%s", &st);

Scanf automatically adds the null character whwn the entire key is pressed.

Note:

1. The string should be short enough to fit into the array.
2. Scanf cannot be used to input multi-word strings with spaces.

gets() and puts()

get() is a function which can be used to recive a multi-word string.

char St[30];

get (St); →The entered string is stored in St !

Multiple gets() call will be needed for multiple strings.

Likewise, puts can be used to output a string.

puts(St); → prints the string places the cursor on the next line

Declaring a string using pointers

We can declare string using pointer

Char*ptr =”neha”;

This tells the compiler to store the string in memory and assigned address is stored in a char pointer.

Note:

1. Once a string is defined using char st[] =”Neha”, it cannot be reinitialized to something else.
2. A string defined using pointers can be reinitialized.
   ptr =”Rohan”;

Standard library function for strings.

C provides a set of standard library functions for string manipulation.

Some of the most commonly used string functions are:

Strlen()

This function is used to count the number of character in the string excluding the null(‘\0’) character.

Int length=Strlen(St);

This function are declared under <string.h> heder file.

Strcpy()

This function is used to copy the content of second string into first string passed to it.

Char source[ ]=”Neha”;

Char target [30];

Strcpy(target, source); →target now contains “Neha”

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

Strcat()

This function is used to concatenate two strings.

Char S1[11] =”Hello”;

Char S2[ ] =”Neha”;

Strcmp()

This function is used to compare two strings. It returns :0 if strings are equal, Negative value is not greater than second string’s corresponding mismatching character, It returns positive values otherwise.

Stecnp(“For”, “Joke”); →Positive value

Strcmp(“Joke”, “for”); →Negative value

Structures

Array and strings => Similar data (int, float, char)

Structures can hold => dissimilar data

Syntax for creating Structures :

A C Structure can be created as follows:

```
Struct employee {
int code;
float salary;
char name[10];
};
```

We can use this user defined data type as follows:

```
Struct employee e1;
Strcpy(e1.name, “Neha”)
E1.code =100;
E1.salary =71.22;
```

So a structure in C is a collection of variables of different types under a dingle name.

Why Use Structures?

We can create the data type in the employee structure separately but when the number of properties in a structure increases, it becomes difficult for us to create data variables without structures. In a nut shell:

a. Structures keep the data organized.
b. Structures make data management easy for the programmer.

Array of Structures

Just like an array of integers, an array of floats and an array of characters, we can create an array of structures.

Struct employee facebook[100];

We can access the data using:

Facebook[0] .code =100;

Facebook[1] .code =101;

000 & 50 on

Initializing Structures

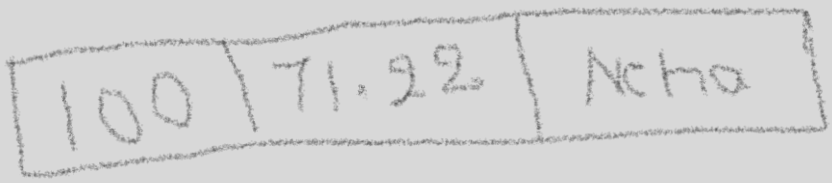
Structures can also be initialized as fallows:

Struct employee neha ={100, 71.22, “Neha”};

Struct employee harry={0}; → All elements set to 0

Structures in memory

Structures are stored in contiguous memory locations for the structure e1 of type struct employee, memory layout looks like this:



Address → 78810 78814 78818

In an array of structures, these employee instances are stored to each other.

Pointer to Structures

A pointer to structure can creates as fallows:

Struct employee*ptr;

Ptr = & ei;

Now we can print structure elements using:

Print(“%d”, *(ptr).code);

Arrow Operator

Instead of writing *(ptr). Code, we use arrow operator to access structure properties as follows

*(ptr). code or ptr → code

Here → is known as the arrow operator.

Passing structure to a function

A structure can be passed to a function just like any other data type.

Void show (struct employee e); => Function prototype

Typedef keyword

We can use the typedef keyword to create an alias name for data types in C. typedef is more commonly used with structures.

```
Struct complex {
    Float real;
    Float img;
};
Typedef struct complex {
    Float real;
    Float img;
    }complexNo;
```

File Input/Output

The Random Access Memory is volatile and its content is lost once the program terminates. In order to persist the sata forever we use files.

A file is data stored in a storage device.

A C program can talk to the file by reading content from it and writing content to it.

File pointer

The “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.

FILE pointer is needed for communication between the file and the program.

A FILE pointer can be created as follows:

FILE *ptr ;

Ptr= fopen(“Filename.ext”, “mode”);

File opening mode in C

C offes the programmers to select a mode for opening a file.

Following mode are primarily used in C file I/O.

“r” → open for reading

“rb” → open for reading in binary

“w” → open for writing

“wb” →open for writing in binary

“a” →open for append

Types of files

There are two types of files:

1. Text file(.txt, .C)
2. Binary file(.jpg, .dat)

Reading a file

A file can be opened for reading as follows:

FILE *ptr;

Ptr = fopen(“Neha.txt”, “r”);

Int num;

Let us assume that “Neha.txt” contain an integer we can read that integer using :

Fscanf(ptr,”%d”, &num);

This will read an integer from file in num variable.

CLOSING the file

It is very important to close the file after read or write. This is achieved using fclose follows :

Fclose(ptr);

This will tell the compiler that we are done with this fileand the associated resources could be freed.

Write to a file

We can write to a file in a very similar member like we read the file

FILE *fptr;

Fptr =fopen (“Neha.txt”, “w”);

Int num =432;

Fprintf(fptr, “%d”,num);

Fgetc() and fpute()

Fgetc and fputc are used to read and write a character from /to file

```
Fgetc(ptr)
Fputc(‘c’,ptr);
```

Eof :End of File

Fgetc return Eof when all the characters form file have read. So we can write a check we like below to detect end of file.

```
While (1) {
    Ch= fgetc(ptr);
    If (ch == Eof) {
        Break;
        }
        //code
        }
```

Dynamic Memory Allocation

C is a language with some fixed rules of programming for example: Changing the size of an array is not allowed.

Dynamic Memory Allocation

Dynamic memory allocation is away to allocate memory to a data structure during the runtime. We can use DMA functions available in C to allocate and free memory during runtime.

Function for DMA in C

Following function are available in C to perform Dynamic memory Allocation:

Malloc ()

Calloc ()

Free ()

Realloc ()

Malloc () function

Malloc stand for memory allocation. It takes numbers of bytes to be allocated as an input and returns a pointer of type void

Syntax :

Ptr =(int*) malloc (30* size of (int))

The expression return a null pointer if the memory cannot be allocated.

Calloc () Function

Calloc stand for Continuous allocation. It initializes each memory blok with a default value of 0.

Syntax :

Ptr =(float*) calloc (30, size of (float));

It the space is not sufficient, memory allocation dails and a NULL pointer is returned.

Free () function

We can use free() function to deallocate the memory. The memory allocated using calloc/malloc is not deallocated automatically.

Syntax:

```
Free (ptr);
```

Realloc () function

Sometime the dunamically allocated memory is insufficient or more than required.

Realloc is used to allocate memory of new size using the provious pointer and size

Syntax:

```
Ptr =realloc (ptr, newSize);
Ptr  =realloc (ptr, 3*sizeof (int));
```