C Programming - Lecture Notes *

T S Pradeep Kumar
VIT University – Chennai Campus
tspembedded@gmail.com
http://www.facebook.com/tspradeep
http://www.pradeepkumar.org

(*This is only a lecture notes and not an elaborate reference to C Programming)

CONTENTS

I. Introduction to C Programming

Features of C Programming

- C is a small language having lesser number of keywords than Java or pascal.
- C is a native language of Unix, Linux. Not only that many of the windows packages, database programs, graphics libraries are written in C program
- C is portable – it provides standard set of libraries that work on the same way with all machines
- C is modular, as it supports functions to divide the program in to sub program
- C is efficient on most machines, because certain constructs are machine dependant.

Structure of a C program

- Comments
  - the comments are usually ignored by the compiler
  - so this part informs about the author of the program
  - what is the usefulness of this program
  - updation or creation of the program
  - Example: // – Single line comment
  - /* —– multi line comments ….*/
- Preprocessor Directives
  - before the compiler runs, the preprocessing takes place.
  - the preprocessors are prefix with a symbol #
  - Examples: #include, #define, #ifdef, #pragma, etc.
- Function prototypes
  - This is to inform the compiler that the following functions are defined in this program.
  - it specifies the name of the function, arguments or parameters and return type and it ends with a semicolon
  - Ex: **int sum(int,int);**
- Global Variables
  - Global variables which are declared above all the functions
  - As the name says, they are available to all the functions of that program
  - They are usually stored in the RAM
- Main Function
  - main is the function which is main to the entire program
  - Every C program should have a main function
  - syntax: int main() { }
  - Earlier, the main program will return nothing, ie void, but now the compilers are instructing us to provide a return type for the main() function
- User defined functions
  - After the main() function, the user defined functions are defined, as per the function prototype declared in the program
  - The function can be defined above the main function also, in that case there is not need for function prototype
  - Syntax:

```
int sum(int a, int b)

{

int f;

f=a+b;

return f;

}
```

II. Structure of C Program

```
/* This program is written by T S Pradeep Kumar on 28th Sep 2010

This is to display Hello World to the display unit

*/

#include <stdio.h> //including the standard IO functions like printf(), scanf()

int main()

{

printf("Hello World \n");

return 0;

}
```

1. stdio.h is the library which contains the printf(), scanf() and other standard IO functions, so before we use any function we need to include in our C program. Most of the compilers will take stdio.h automatically, even if we dont include in our program
2. int main() {} is the main function
3. printf("Hello World\n"); printf() is the function which is printing to the display unit and it is displaying Hello World. Whatever is there inside the double quotes will displayed as it is in the monitor except the format specifiers or format codes (%d, %f, %c)
4. return 0; is the final statement which returns the integer value 0 (this is just to make the compiler happy)

Here in the above program, you can see there are semi colons ;

Every C statement must end with a semicolon. If a statement is not completed, then there need not be a semicolon

III. printf() and scanf() Function

- Printf() is a function to print strings to the display unit and

- scanf() is to scan the input through the keyboard the f in the printf and scanf are called as format. That is, printf() and scanf() will print and scan in some particular format, it is the duty of the developer to provide that formats.
- For example
- **printf("a=%d, b=%d", a,b);** //This function will print a=10, b=20 if the values of a and b are 10 and 20
- **scanf("%f", &a);** //this will accept a float number as the specifier is %f which is for float, See the Ampersand symbol which to tell the compiler to store the variable into the address.
- Here is the list of format codes or format specifiers
- **scanf() format codes**

Code	**Meaning**
%c	read a single character
%d	read a decimal integer
%f	read a floating point value
%e	read a floating point value (even in exponential format)
%g	read a floating point value
%h	read a decimal, heaxadecimal or octal integer
%o	read an octal integer
%s	read a string
%u	read an unsigned integer
%x	read a hexadecimal integer

- h for short integers like %hd
- l for long integers like "%ld
- L for long double like %Lf
- **printf() format codes**

Code	**Meaning**
%c	prints a single character
%d	print a decimal integer
%e	print a floating point value in exponential form
%f	print a float point value
%g	print a float point value either in f type or e type

	depending on the value
%i	print a signed decimal integer
%o	prints an octal integer
%s	prints a string
%u	print an unsigned integer
%x	print a hexadecimal number

-
- **Backslash Character constants or Escape Characters**
- There are some characters inside the double quotes for printf() and scanf() which are escaped by the compiler and hence it will the next character after the escape characters

Character	**What it will do**
\n	go to the new line
\t	horizontal tab
\a	alert bell
\f	form feed
\r	carriage return
\”	will print “
\’	will print ‘
\?	will print ?
\\	will print \
\v	vertical tab
\b	backspace

IV. Tokens in C

Tokens are classified as

- Keywords
- Identifiers
- Operators
- Constants
- Special Symbols

Keywords

Keywords are reserved by the compilers and keywords will not be redefined (means the keywords are not been declared as variable names, etc). There are standard 32 keywords in C compiler, and some compilers uses more keywords based on their specification. Here is the list of 32 keywords

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

Identifiers

1. Identifiers are variable names declared inside the program.
2. They use alphabets, number, symbols like _ and $
3. They must begin with alphabet or _
4. They are case sensitive i.e **Var_name** is different from **var_name**

Some valid identifiers are

john	Value	r_sie	x1
x2	Chennai	Sum1	abcd

invalid identifiers

123	%	(super)	25th
1st	ab-d	sum of total	

Operators

operators are further classified into different categories like

- Arithmetic operators

+ – for addition 1+3 = 4

- for subtraction 3 – 2 = 1

* for multiplication 4* 2 = 8

/ for Division 4 / 2 = 2

% for modulo 4 % 3 = 1 (when 4 is divided by 3, the remainder is 1)

- Increment/ Decrement operators

++ increment Ex: a++ or ++a

– decrement Ex: a—or –a

- relational Operators

< – less than

> – greater than

<= less than or equal to

>= greater than or equal to

== is equal to

- logical operators

&& – Logical AND

|| – Logical OR

! – Logical NOT

- bitwise operators

& Bitwise AND

| bitwise OR

^ bitwise XOR

~ one's complement

<< shift left

>> shift right

- Conditional Operators

? : Conditional operator

f =(a>b) ?a : b;

if the condition a> b is true then

a is assigned to f and if false then b is assigned to f

the statement can be rewritten as

```
if(a>b>
{
f=a;
}
else
{
f=b;
}
```

- Other Operators

= assignment operator (this is not equality operator, it just assigns the value to some other variable from right to left assignment)

Example: a=b means b is assigned to a

, comma operator (This is to separate the array variables, etc)

sizeof() operator (to find out the various sizes of the entities like int, char, array size, etc)

- Special symbols

{} This to begin and end any statement, loop in C program

[] This is for specifying the array index (Ex: int a[10];)

() This is used for functions in C and helpful for grouping arithmetical operations

Constants

- Constants is an entity will never changes it value during the execution of the program
- They are different from variables

- Integer constant

Examples are 2, –1, 0, 345, 324, etc

- float Constant

Examples are 3.0, –2.345, 32.56, 4.65

- String Constant

“abc”, “Pradeep Kumar”

- Character constant

‘A’, ‘b’, ‘1’

There are two ways to declare a constant

1st way is to define at the preprocessing level

Example is

#define PI 31.4

2nd way is declare inside the program

const int a=3.14;

const is the keyword to define a constant.

V. Data types in C

Declaration of variable names does two things

- it tells the compiler what the variable name is
- specifies what type of data the variable will hold

So it is mandatory that each variable should belong to a particular data type. C supports different data types like integer, character, float, double, etc.

Data type	**Size on a 16 bit machine**	**Range**
char	1 byte or 8 bits	-128 to 127
signed char	1 byte	-128 to 127
unsigned char	1 byte	0 to 255
short int or short	1 byte	-128 to 127
int	2 bytes	-32768 to 32767
unsigned int	2 bytes	0 to 65535
long int or long	4 bytes	-2,147,483,648 to 2,147,483,647
unsigned long int	4 bytes	0 to 4,294,967,295
float	4 bytes	3.4E-38 to 3.4 E+38

double	8 bytes	1.7E-308 to 1.7E+308
long double	10 bytes	3.4E-4932 to 1.1E+4932

Example of declaring variables

int a=20,b=20,c;

int a=10;

int b=9;

int c=20;

float c=3.5;

char a ='A';

long int al or long a;

short int b or short b;

double c;

long double c;

VI. Decision Making and Looping

C supports Decision Making and Looping (Interations)

In decision Making,

if, if-else, if- else if – else, switch – case is used.

if(condition)

{

//These statements are executed only if the condition is true or 1

}

if – else

if(condition)

{

```
//These statements are executed only if the condition is true or 1

}

else

{

//these statement are executed only if the condition is false or 0

}
```

(in the above example, there is no else part means that for some situation, there may not be a need for else part)

if – else if – else

```
if(condition 1)

{

//These statements are executed if the condition 1 is true or 1

}

else if (condition 2)

{

//These statements are executed if the condition 2 is true or 1

}

else

{

//These statements are executed neither condition 1 nor condition 2 are true

// if all the conditions fails, then this statement will execute

}
```

Switch Case

```
switch (expression)
```

```
{

case 1: statement 1;

break;

case 2: statement 2;

break;

case 3: statement 3;

break;

default: statement d;

break;

}

//Statement n;
```

The switch case works similar like a multiple if else if statement.

Assume if the expression takes 3 as the value and the statement 3 will be executed and next there is a break statement, so the control comes to statement n and it ill be printed.

so when case 3: is success, then

statement 3;

statement n;

will be printed

Looping

- while()
- do .. while();
- for()

while ()

```
while(condition)

{
```

//statement 1

}

statement 1 will keep on execution till the condition becomes false

so this loop is called as entry controlled loop

do while()

do

{

//Statement 1

}while(condition);

in this, whether the condition is true or false, first time the loop will execute and only then it will check for validity of the condition.

so this loop is called as exit controlled loop

for loop

for (initialization: condition; increment/decrement)

{

//Statement

}

More details about the looping will be in subsequent posts.

Assume the number 1234, after reversal the number becomes 4321.

Input: One number, Ex 1234, declared to **num**

Output : One number, Ex 4321, **rev** is the variable

Logic:

1. separate each digit using modulo operator from the right and bring that number to front
2. divide the given number by 10, so the right most digit will be discarded.
3. run till the number becomes <=0

output the result

C Program 1 – To find the reversal of a given number

```
#include <stdio.h>
#include <conio.h>

int main()
{
   int num,rev=0;
   printf("enter the number to be reversed");
   scanf("%d", &num);

   while(num>0) //till the number is positive, perform the process
   {
   rev=rev*10 + (num%10); //separate each digit and bring it to the first
   num=num/10;
   }
   printf("The reversal is %d ",rev);
   getch();
   return 0;
}
```

C Program 2 – Electricity Bill calculation

This program is to display the electricity bill calculation based on the number of units consumed every month

Input : the number of units – variable name – unit

Output – Amount of rupee – variable name – amount

Logic:

Units	**Rupees**
1-50 units	0.75/unit
51-100	0.85/unit
101-200	1.50/unit
201-300	2.20/unit
>300	3.00/unit

In all the cases, there will be service charge of Rs.20 will be collected.

```
#include <stdio.h>
#include <conio.h>
int main()
```

```
{
   float amount=0,units;;
   printf("Enter the number of units");
   scanf("%f", &units);
   if(units <=50)
   {
    amount = units * 0.75;
   }
   else if(units >50 && units <=100)
   {
      amount=0.75 * 50 + 0.85*(units-50);
   }
   else if(units >100 && units <200)
   {
      amount=(0.75*50) + (0.85*50 )+ (1.5 *(units-100));
   }
   else if(units >200 && units <300)
   {
      amount=(0.75*50) + (0.85*50 )+ (1.5 *100) + (2.20 *(units-200));
   }
   else
   {
      amount=(0.75*50) + (0.85*50 )+ (1.5 *100) + (2.20 * 100) +(3.0*(units-300));
   }
   amount=amount+(0.2*amount);
   printf("The total electricity bill is %f", amount);
   getch();
   return 0;
}
```

C Program 3 – To check whether a number is Armstrong Number

The armstrong number is of the form 153= $1^3 + 5^3 + 3^3$

The input is : 153 or any other number

output: The number is armstrong or not.

Processing: take 153 as an example, remove 3, 5 and 1 in the reverse order (using % operator) and take the power of 3 and add to the sum variable.

if the total sum and the original number, both are same, then that is the arm strong number.

if else, the number is not an armstrong number

```
#include <stdio.h>
#include <conio.h>
```

```
int main()
{
   int original_num, check, temp, sum=0;
   printf("Enter the number to check for armstrong number");
   scanf("%d", &original_num); // Get the original number
   temp=original_num;
   while(original_num>0) //run the loop till the number becomes 0
   {
    check=original_num%10;  //remove the last digit using modulo operator
    sum=sum+check*check*check; //the last digit is taken power to 3 and added to sum
    original_num=original_num/10; //truncate the last digit and run the loop again
    }
    if(sum==temp)
    printf("This is an armstrong number\n");
    else
    printf("This is not an armstrong number \n");
    getch();
    return 0;
}
```

C Program 4–To check whether a given number is prime or not

A prime number can be divided by 1 and itself, there are no other divisors,

Examples are : 2 3, 5, 7, 11, …..

To find out whether a given number is prime or not, here is the logic

1. Get the number

2. divide the given number from 2 to n-1 (Example if 6 is the number divided by 2,3,4,5 will get the remainder respectively 0,0,2,3)

3. increment a counter to 1 if the remainder is 0

4. if there counter variable is 0, then the given number is prime (because we didn't get any remainder) else non prime

Here is the program

```
#include <stdio.h>
#include <conio.h>

int main()
{
   int a,i,count=0;
   printf("enter a"); //Let the given number is a
```

```
  scanf("%d",&a); //get the number
  for(i=2;i<a;i++) //divide the number a from 2 to a-1
  {
  if(a%i==0)
  count++; //increment a counter if the divisibility is 0
  }
  if(count !=0) //if the counter is not zero, then prime
  printf("a is not a prime number");
  else
  printf("a is a prime number");
getch();
return 0;
}
```

VII. Functions in C

Almost all the programming languages uses functions. Functions are the entities which are grouping a set of statements which do a specific job or set of jobs.

Example: sum of integers, sum of float number, complex number addition.

All the above three can be implemented as a single function or three separate functions.

When someone wants to use that, a function can be simply called.

So function implementation happens as

- Function prototype or Function declaration
- Function definition or function implementation
- Function call

Function prototype

- It is necessary to specify the name of the function, the parameters and the return type to the compiler that the function is being defined in this program.
- The prototype ends with a semicolon

syntax:

int sum(int, int); //function prototype

Function Definition

This is the actual function definition which shows the function implementation.

for the above syntax here is the function

```
int sum(int a, int b)

{

int c;

c=a+b;

return c;

}
```

The above function is returning an integer, hence int is specified. If a function is not returning any thing, a void can be used.

Function call

The last is the function call, which when being needed a simple call will make the function to work. Here is the example

```
int main()

{

int x,y,z;

scanf("%d %d",&x,&y);

z=sum(x,y); //function call

printf("The sum is %d", z);

return 0;

}
```

Once the function is called, the control goes to the actual implementation and execute the statements inside the function and the value is returned to the main function.

Components of a function

1. Name of the function
2. Return type
3. Parameters or arguments

In the above example

name of the function is : **sum()**

return type is **: int**

parameters are: **int, int**

VIII. Arrays in C

Arrays in C programming starts with memory addresses. Yes the compiler handles arrays as memory addresses.

- Similar elements group together with a single name is called array
- the indexing always starts with 0 and ends at n-1 (if the size of the array is n)
- Array elements stores in consecutive memory location.

The array elements usually stores in consecutive memory location. For example,

int a[10]; **//Array with size 10**

the above example is an integer array with size 10 or this array can hold 10 variables.

The elements starts from a[0], a[1], a[2]…..a[9], so totally there are 10 elements.

Array elements	values	memory address location
a[0]	32	600AH **(This address is called as the base address)**
a[1]	34	600EH
a[2]	23	6012H
a[3]	35	6016H
a[4]	45	601AH
a[5]	67	601EH
a[6]	56	6022H
a[7]	1	6026H
a[8]	34	602AH
a[9]	25	602EH

The memory addresses shown above are in hexadecimal, that's why each address suffixed with a letter **H. The memory addresses in each machine will be different.**

The base address is the address where the first element of the array is stored**.** Usually it is represented as **&a[0]** or simply **a.**

The entire array is represented as an address by specifying the base address alone to various other elements of C programming like passing array to functions involves sending the base address only.

For the above table, here is the declaration syntax

```
int a[10]={32,34,23,35,45,67,56,1,34,25};
```

or

```
int a[]={32,34,23,35,45,67,56,1,34,25};
```

A simple program to get some numbers and print them on the screen

```
int main()
{
int a[10], j; //declaration of array “a” and its index j
printf(“Enter all the 10 numbers”);
for(j=0;j<10;j++) //get the numbers using a for loop which executes for 10 times (0 to 9)
{
scanf(“%d”, &a[j]);
}
for(j=0;j<10;j++) //print the numbers using a for loop which executes for 10 times (0 to 9)
{
printf(“%d”,a[j]);
}
return 0;
}
```

In the above example, j varies from 0 to 9 which means the loop will execute 10 times for the array a like this

a[j] when j=0 then a[0]

a[j] when j=1 then a[1]

a[j] when j=2 then a[2]…. and so on

like this the input is entered through the keyboard and same for printing the output.

C Program–Arrays with Functions (passing Array elements to function)

```
/*

Program to demonstrate arrays with function in which the array elements are passed as function
parameters

*/

#include <stdio.h>
#include <conio.h>
void display(int); //function prototype
int main()
{
   int a[10],i;
   for(i=0;i<10;i++) //getting the array elements
   {
   scanf("%d",&a[i]);
   }
   for(i=0;i<10;i++)
      display(a[i]); //call the function inside the loop, this function called for 10 times
   getch();
   return 0;
}

void display(int a) //function implementation to print the elements, here the parameter is an
integer only(means we are passing the array elements)
{
    printf("%d",a);
}
```

C Program–Passing an entire array to a function

```
/* This program is to pass the entire array to a function */

#include <stdio.h>
#include <conio.h>
int maximum(int[],int); //function prototype, 1st parameter is array and the second is integer

int main()
{
   int a[100],i,n,max=0;
   printf("Enter the numbers");
   scanf("%d",&n); //get the index number
   for(i=0;i<n;i++) //get the array elements
   {
   scanf("%d",&a[i]);
   }
   max=maximum(a,n); //function call
   printf("Maximum number is %d",max);
   getch();
   return 0;
}

int maximum(int b[],int a) //function implementation
{
   int i,max=b[0];
   for(i=0;i<a;i++)
   if(b[i]>max)
   max=b[i];
   return max;
}
```

In the function call ie

max=maximum(a,n) where a is the array and n is the number of array elements

the above line can be written as

max = maximum(&a[0],n)

since the array is just a memory address, it is enough to mention the base address (address of the first element) of the array

so the array name is just equivalent to the base address

in the above example

a means &a[0]

C Program–To sort a Given set of numbers in ascending order (Bubble Sort)

```
/* Program to sort the given set of numbers in
ascending order, this sorting is called as bubble sort algorithm
*/
```

```
#include <stdio.h>
#include <conio.h>
int main()
{
   int a[10],i,j,temp=0;
   printf("Enter all the 10 numbers");
   for(i=0;i<10;i++)
   scanf("%d",&a[i]);
   for(i=0;i<10;i++)  //This loop is for total array elements (n)
   {
   for(j=0;j<9;j++) //this loop is for total combinations (n-1)
   {
               if(a[j]>a[j+1]) //if the first number is bigger then swap the two numbers
               {
               temp=a[j];
               a[j]=a[j+1];
               a[j+1]=temp;
               }
   }
   }
printf("The ordered array is");
for(j=0;j<10;j++) //Finally print the ordered array
printf("%d \t",a[j]);
getch();
return 0;
}
```

C Program–To search a number in a given array

```
//This program is to search a given number in an array
#include <stdio.h>
#include <conio.h>

int main()
{
   int a[10],i,num;
   printf("enter the array elements");
   for(i=0;i<10;i++) //get all the numbers
   scanf("%d",&a[i]);
   printf("Enter the number to search");
   scanf("%d",&num);
   for(i=0;i<10;i++)
   {
               if(a[i]==num) //given num is matched in the array
               {
               printf("The number is found in the %d position",i+1);
               getch();
```

```
                exit(0); //to go the end of the program
                }
  }
printf("The number is not found"); //if num not found, this will be displayed
getch();
return 0;
}
```

C Program–Adding two matrices

```
/* Program to add two matrices */

#include <stdio.h>
#include <conio.h>
int main()
{
  int a[10][10], b[10][10],c[10][10],i,j;
  printf("Enter a");
  for(i=0;i<2;i++)  //get the matrix A
    for(j=0;j<2;j++)
  scanf("%d",&a[i][j]);
  printf("Enter b");
  for(i=0;i<2;i++) //get the matrix B
    for(j=0;j<2;j++)
  scanf("%d",&b[i][j]);

  for(i=0;i<2;i++)
  {
    for(j=0;j<2;j++)
    {
    c[i][j] = a[i][j] +b[i][j];    //adding two matrices
    }
  }
  printf("Added Matrix is \n");
  for(i=0;i<2;i++)
    for(j=0;j<2;j++)
  printf("%d ",c[i][j]);
    getch();
    return 0;
}
```

C Program–Multiplying two matrices

```
/* Program to multiply two matrices */

#include <stdio.h>
#include <conio.h>
```

```
int main()
{
   int a[2][3],b[3][2],c[2][2],k,j,i;
   printf("enter a");
   for(i=0;i<2;i++)   //Get array A
   {
   for(j=0;j<3;j++)
   {
   scanf("%d",&a[i][j]);
   }
   }
   printf("enter b");
   for(i=0;i<3;i++) //Get array B
   {
   for(j=0;j<2;j++)
   {
   scanf("%d",&b[i][j]);
   }
   }

   for(i=0;i<2;i++)
   {
   for(j=0;j<2;j++)
   {
   c[i][j]=0;  //to Hold a temporary multiplication
   for(k=0;k<3;k++)
   {
   c[i][j]=c[i][j]+a[i][k]*b[k][j];   //Multiplication algorithm
   }
   }
   }
   printf("C is ");
   for(i=0;i<2;i++)
   for(j=0;j<2;j++)
   printf(" c[%d][%d] – %d \n",i,j,c[i][j]);

getch();
   return 0;
}
```

IX. Structures in C Programming

1. Structure in C groups different data type under a common name
2. struct is a keyword to declare a structure
3. The size of the structure depends on the size of the different data types
4. Structure declaration always ends with a semicolon

Example:

```
struct employee

{

char name[50];

int empid;

float salary;

};
```

The total size occupied by the above structure is 50 + 2 + 4 = 56 bytes (on TC compiler) and 50 + 4 + 4= 58 (on other compilers like GCC, etc). Again the size depends on the compiler.

GCC compiler always increase the size of structure which is divisible by 4 for faster code generation, so the above structure takes 60 bytes of memory

1. Declaring a structure does not consume any memory, only when the structure object is created memory occupies
2. Variables initialization will not be done during the structure declaration

The above syntax declaration does not consume memory, but the following do

```
int main()

{

struct employee e1;

}
```

Now e1 consumes memory of 60 bytes.

Simple Structure to get some details and print them to the screen

```
// GCC Compiler under windows 64bit

#include <stdio.h>
#include <conio.h>

/*  structures are declared outside any functions,
so that all the functions in the program can have access to it
*/
struct employee
```

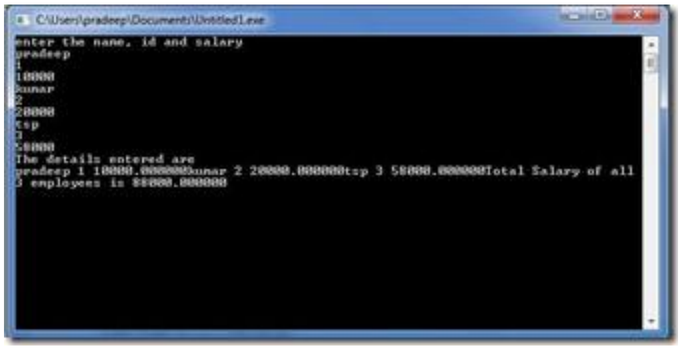
```
{
    char name[50];
    int empid;
    float salary;
};

int main()
{
  struct employee e1;
  printf("enter the name, id and salary");
  scanf("%s %d %f",e1.name, &e1.empid, &e1.salary);

  printf("The details entered are \n");
  printf("%s %d %f",e1.name, e1.empid, e1.salary);
  getch();
  return 0;
}
```

In the above program name, empid and salary are part of employee, so it should be associated with the member of structure, that's why e1.name, e1.empid and e1.salary.

Array of Structures

```
//the following program gets the inputs of 3 employees like name, id and salary and compute the total
salary taken by all the employees

#include <stdio.h>
#include <conio.h>

/*  structures are declared outside any functions,
so that all the functions in the program can have access to it
*/
struct employee
{
    char name[50];
    int empid;
    float salary;
};

int main()
{
  struct employee e[3];
  int i;
  float sum=0;
  printf("enter the name, id and salary");
  for(i=0;i<3;i++)
  {
    scanf("%s %d %f",e[i].name, &e[i].empid, &e[i].salary);
    sum=sum+e[i].salary;
    }


    printf("The details entered are \n");
    for(i=0;i<3;i++)
    printf("%s %d %f",e[i].name, e[i].empid, e[i].salary);
    printf("Total Salary of all 3 employees is %f",sum);
    getch();
    return 0;
}
```

Sample Output



```
C:\Users\pradeep\Documents\Untitled1.exe
enter the name, id and salary
pradeep
1
10000
kunar
2
20000
tsp
3
58000
The details entered are
pradeep 1 10000.000000kunar 2 20000.000000tsp 3 58000.000000Total Salary of all
3 employees is 88000.000000
```

Arrays within Structures

//The following program gets the 2 student inputs like 3 subject marks and compute the total of all three subject marks

```
#include <stdio.h>
#include <conio.h>

/*  structures are declared outside any functions,
so that all the functions in the program can have access to it
*/
struct student
{
    int subjects[3];
    float total;
};

int main()
{
   struct student s[2];
   int i,j;

   printf("enter the 3 subject marks");
   for(i=0;i<2;i++)
   {
   s[i].total=0;
   for(j=0;j<3;j++)
   {
   scanf("%d",&s[i].subjects[j]);
   s[i].total += s[i].subjects[j];
   }
   printf("The total marks scored by student%d is %f",i+1,s[i].total);
   }

   getch();
   return 0;
}
```

X. <u>Passing Entire structure to Function</u>

- A function can accept a structure as a parameter and even it returns a parameter too.
- Either the individual structure elements can be passed or the entire structure can be passed.

Here is the example, for passing the entire structure to a function

```
#include <stdio.h>
#include <conio.h>

//Structure declaration

struct student
{
char name[10];
int no;
};
```

//Function prototypes

```
void read(struct student);
```

struct student display(struct student); **//function which takes the entire structure as a parameter and it returns a structure also**

```
int main()
{
   struct student s1,s;
    printf("Enter the details");
   scanf("%s %d",s1.name,&s1.no);
   s=display(s1);  //function call
printf("Name is %s and number is %d",s.name,s.no);
   getch();
}
```

//Function definition

```
struct student display(struct student s)
{
return s;
}
```

Nested Structures

A structure can be declared within another structure.

Example include: for a ***student*** structure maintaining the date of birth is tougher within the structure,

to avoid the complexity, dateofbirth can be a separate structure as defined below

```
struct dateofbirth
{
int date;
int month;
int year;
};

struct student
{
char name[50];
int no;
struct dateofbirth dob;
};
```

so the structure can be accessed using the following:

s.dob.date, s.dob.month, s.dob.year, and s.name, s.no

Example of Nested Structures

```
#include <stdio.h>
#include <conio.h>

//declare the structure dateofbirth which holds date,month and year
```

```
struct dateofbirth
{
int date;
int month;
int year;
};

//declare another structure student which also includes dateofbirth as a datatype

struct student
{
char name[50];
int no;
struct dateofbirth dob; //from another structure
};

int main()
{
struct student s;
printf("Enter the name number and date of birth (dd/mm/yyyy)");
scanf("%s %d %d/%d/%d",s.name,&s.no,&s.dob.date,&s.dob.month,&s.dob.year);
printf("%s %d %d/%d/%d",s.name,s.no,s.dob.date,s.dob.month,s.dob.year);
getch();
return 0;
}
```

It is always nice to include nested structures in applications as good readability is provided

for Example: **building.concrete.cement.quality**

XI. String Handling Functions

Like numbered arrays, C handles character arrays (strings).

1. Each string is identified as a character array and ends with a '\0'(null) character (the compiler automatically adds the null character). So the end of string is identified as a null character.
2. Being an array, all the elements of the string array stored in continuous memory locations

following is the declaration of character array

char name[50];

or

char name[]="Pradeep Kumar";

or

```
char name[20]="Hello Pradeep";
```

Strings can be handled or manipulated through loops or library functions. There are some library functions to handle strings are available at string.h

Some of them are

strlen(string)

- This is to find the length of the string and returns an integer
- For example, if "Hello" is the string, the length will be 5 and "Hello ", the length is 6 (there is a blank space after o in hello)

strcat(String1, string2)

- to concatenate two strings
- One string will be appended to another string (the concatenation happens by inserting the new string at the null character '\0' of the first string)
- Example, if String1 is "Hello" and String2 is "Pradeep", then strcat(string1, string2) will be "HelloPradeep"
- Similarly, if String1 is "Hello " and String2 is "Pradeep", then strcat(string1, string2) will be "Hello Pradeep"

strcpy(Destination, source)

- This function is to copy one string to other
- usually the copying string will be the 2nd parameter and copied string will be the first parameter
- Example, if string1 is "Bad" and string2 is "Good" the strcpy(string1,string2) will give Good for string1

strcmp(Destination, source)

- This function is to compare two strings whether they are equal or not
- like numbers, two strings should not be compared directly like if(a==b)
- If both the strings are equal, the function returns 0, else it will return the difference of ASCII value of the first non matching characters
- Example, strcmp("Their", "There") will return ASCII(i) – ASCII(r) (an integer will be returned)

Example Program

```
//program to demonstrate string handling functions
#include <stdio.h>
#include <conio.h>
int main()
{
   char a[50];
   char b[50];
   printf("Please the two strings one by one\n");
   gets(a);
   gets(b);
   printf("Length of String a is %d \n",strlen(a));
   printf("Length of String b is %d \n",strlen(b));
   if(!strcmp(a,b))  //Comparing two string will return 0, ! before strcmp() is needed
   printf("Both the strings are Equal");
   else
   printf("Both the strings are not equal");
   strcat(a,b);  //Concatenation function
   printf("the concatenated String is:");
   puts(a);
   strrev(a);
   printf("The reverse string is\n");
   puts(a);
   getch();
   return 0;
}
```

In the above program , instead of gets() function, scanf() can be used, but the disadvantage of scanf() being it truncates the blank space, new line, form feed, tab.

So for example, if the input give is “Hello pradeep”, then Hello only will be accepted, the string after the blank space will be omitted by the compiler, so gets() can be used.

Similarly, puts() or printf() can be used to display strings.