PROGRAMMING FOR PROBLEM SOLVING USING C
UNIT-III

Arrays: Concepts, Using Array in C, Array Application, Two Dimensional Arrays, Multidimensional Arrays, Programming Example – Calculate Averages, Tips and Common Programming Errors, Key Terms, Summary, Practice Set.
Strings: String Concepts, C String, String Input / Output Functions, Arrays of Strings, String Manipulation Functions String/ Data Conversion, A Programming Example – Morse Code, Tips and Common Programming Errors, Key Terms, Summary, Practice Set.
Enumerated, Structure, and Union: The Type Definition (Type def), Enumerated Types, Structure, Unions, Programming Application, Tips and Common Programming Errors, Key Terms, Summary, Practice Set.

➢ Arrays:-

An array is a group of related data items that share a common name.
Ex:-Students

- The complete set of students are represented using an array name students.
- A particular value is indicated by writing a number called index number or subscript in brackets after array name.
- The complete set of value is referred to as an array, the individual values are called elements.

➢ Using Array in C:-

- to store list of Employee or Student names,
- to store marks of students,
- or to store list of numbers or characters etc.

Since arrays provide an easy way to represent data, it is classified amongst the data structures in C. Other data structures in c are structure, lists, queues, trees etc. Array can be used to represent not only simple list of data but also table of data in two or three dimensions.

➢ Array Application:-

In c programming language, arrays are used in wide range of applications. Few of them are as follows.

❖ Arrays are used to Store List of values

In c programming language, single dimensional arrays are used to store list of values of same datatype. In other words, single dimensional arrays are used to store a row of values. In single dimensional array data is stored in linear form

❖ Arrays are used to Perform Matrix Operations

We use two dimensional arrays to create matrix. We can perform various operations on matrices using two dimensional arrays.

- **Arrays are used to Perform Matrix Operations**
  We use two dimensional arrays to create matrix. We can perform various operations on matrices using two dimensional arrays.
- **Arrays are used to implement Search Algorithms**
  We use single dimensional arrays to implement search algorihtms like ...

  1. Linear search
  2. Binary search

- **Arrays are used to implement Sorting Algorithms**
  We use single dimensional arrays to implement sorting algorithms like ...
  1. Insertion sort
  2. Bubble sort
  3. Selection sort
  4. Quick sort
  5. Merge sort

➢ **ONE – DIMENSIONAL ARRAYS :-**

A list of items can be given one variable index is called single subscripted variable or a one-dimensional array.

The subscript value starts from 0. If we want 5 elements the declaration will be

```
int number[5];
```

The elements will be number[0], number[1], number[2], number[3], number[4]
There will not be number[5]

➢ **Declaration of One - Dimensional Arrays :**

Type variable – name [sizes];

Type – data type of all elements Ex: int, float etc.,
Variable – name – is an identifier
Size – is the maximum no of elements that can be stored

Ex:- float avg[50]

This array is of type float. Its name is avg. and it can contains 50 elements only. The range starting from 0 – 49 elements.

- **Initialization of Arrays :-**
  Initialization of elements of arrays can be done in same way as ordinary variables are done when they are declared.

  Type array name[size] = {List of Value};

  Ex:- int number[3]={0,0,0};
  If the number of values in the list is less than number of elements then only that elements will be initialized. The remaining elements will be set to zero automatically.

- **TWO – DIMENSIONAL ARRAYS:-**
  To store tables we need two dimensional arrays. Each table consists of rows and columns. Two dimensional arrays are declare as

  type array name [row-size][col-size];

  /* Write a program Showing 2-DIMENSIONAL ARRAY */
  /* SHOWING MULTIPLICATION TABLE */

```
#include<stdio.h>
#include<math.h>
#define ROWS 5
#define COLS 5
main()
{
int row,cols,prod[ROWS][COLS];
int i,j;
printf("Multiplication table");
for(j=1;j< =COLS;j++)
printf("%d",j);
for(i=0;i<ROWS;i++)
{
row = i+1;
printf("%2d|",row);
for(j=1;j < = COLS;j++)
{
```

```
COLS=j;
prod[i][j]= row * cols;
printf("%4d",prod*i+*j+);
}
}
}
```

- **INITIALIZING TWO DIMENSIONAL ARRAYS:-**
  They can be initialized by following their declaration with a list of initial values enclosed in braces.

  Ex:- int table[2][3] = {0,0,0,1,1,1};

  Initializes the elements of first row to zero and second row to one. The initialization is done by row by row. The above statement can be written as
  int table[2][3] = {{0,0,0},{1,1,1}};

When all elements are to be initialized to zero, following short-cut method may be used.

int m[3][5] = {{0},{0},{0}};

- **Multidimensional Arrays:-**
  C allows for arrays of two or more dimensions. A two-dimensional (2D) array is an array of arrays. A three-dimensional (3D) array is an array of arrays of arrays.
  In C programming an array can have two, three, or even ten or more dimensions. The maximum dimensions a C program can have depends on which compiler is being used.

- **Declare a Multidimensional Array in C:-**
  A multidimensional array is declared using the following syntax:

  type array_name[d1][d2][d3][d4]………[dn];

  Where each **d** is a dimension, and **dn** is the size of final dimension.

Examples:

1. int table[5][5][20];
2. float arr[5][6][5][6][5];

In Example 1:

- **int** designates the array type integer.
- **table** is the name of our 3D array.
- Our array can hold 500 integer-type elements. This number is reached by multiplying the value of each dimension. In this case: **5x5x20=500**.

In Example 2:

- Array **arr** is a five-dimensional array.
- It can hold 4500 floating-point elements (**5x6x5x6x5=4500**).

➢ Initializing a 3D Array in C:-

Like any other variable or array, a 3D array can be initialized at the time of compilation. By default, in C, an uninitialized 3D array contains “garbage” values, not valid for the intended use.

Let’s see a complete example on how to initialize a 3D array:-

```
#include<stdio.h>
#include<conio.h>
void main()
{
int i, j, k;
int arr[3][3][3]=
       {
          {
          {11, 12, 13},
          {14, 15, 16},
          {17, 18, 19}
          },
          {
          {21, 22, 23},
          {24, 25, 26},
          {27, 28, 29}
          },
          {
          {31, 32, 33},
          {34, 35, 36},
          {37, 38, 39}
          },
       };
clrscr();
printf(":::3D Array Elements:::\n\n");
for(i=0;i<3;i++)
{
   for(j=0;j<3;j++)
   {
      for(k=0;k<3;k++)
      {
      printf("%d\t",arr[i][j][k]);
      }
      printf("\n");
   }
   printf("\n");
}
getch();
}
```

➢ Programming Example – Calculate Averages:-

This program takes n number of element from user (where, n is specified by user), stores data in an array and calculates the average of those numbers.

Source Code to Calculate Average Using Arrays:-

```
#include <stdio.h>

int main()
{
   int n, i;
   float num[100], sum = 0.0, average;

   printf("Enter the numbers of elements: ");
   scanf("%d", &n);

   while (n > 100 || n <= 0)
   {
      printf("Error! number should in range of (1 to 100).\n");
      printf("Enter the number again: ");
      scanf("%d", &n);
   }

   for(i = 0; i < n; ++i)
   {
      printf("%d. Enter number: ", i+1);
      scanf("%f", &num[i]);
      sum += num[i];
   }

   average = sum / n;
   printf("Average = %.2f", average);

   return 0;
```

}

Output

```
Enter the numbers of elements: 6
1. Enter number: 45.3
2. Enter number: 67.5
3. Enter number: -45.6
4. Enter number: 20.34
5. Enter number: 33
6. Enter number: 45.6
Average = 27.69
```

➢ Strings:-

A String is an array of characters. Any group of characters (except double quote sign)defined between double quotes is a constant string.

Ex: “C is a great programming language”.

If we want to include double quotes.

Ex: “\”C is great \” is norm of programmers “.

➢ Declaring and initializing strings :-

A string variable is any valid C variable name and is always declared as an array.

char string name [size];

size determines number of characters in the string name. When the compiler assigns a character string to a character array, it automatically supplies a null character (‘\0’) at end of String.

Therefore, size should be equal to maximum number of character in String plus one.

There are two ways to declare a string in c language.

1. By char array
2. By string literal

1. char city*10+= ,‘N’,’E’,’W’,’ ‘,’Y’,’O’,’R’,’K’,’\0’-;

2. char city*10+= “NEW YORK’;

C also permits us to initializing a String without specifying size.

Ex:- char Strings*+= ,‘G’,’O’,’O’,’D’,’\0’-;

➢ String Input / Output Functions:-

- C provides two basic ways to read and write strings
- First we can read and write strings with the formatted input/output functions,scanf/fscanf and prinf/fprinf.
- Second we can use a special set of strin only functions ,get string(gets/fgets)and put string(puts/fputs).

Formatted string Input/Output:

- Formatted String Input:scanf/fscanf:
- **int fscanf(FILE ****stream***, const char ****format***, ...);**
- **int scanf(const char ****format***, ...);**
- The ..scanf functions provide a means to input formatted information from a stream.
- **fscanf** reads formatted input from a stream
- **scanf** reads formatted input stdin

These functions take input in a manner that is specified by the format argument and store each input field into the following arguments in a left to right fashion.

String Input/Output

In addition to the Formatted string functions,C has two sets of string functions that read and write strings without reformatting any data.These functions convert text file lines to strings and strings to text file lines

gets():

Declaration:

```
char *gets(char *str);
```

Reads a line from `stdin` and stores it into the string pointed to by *str*. It stops when either the newline character is read or when the end-of-file is reached, whichever comes first. The newline character is not copied to the string. A null character is appended to the end of the string.

puts:

Declaration:

```
int puts(const char *str);
```

Writes a string to `stdout` up to but not including the null character. A newline character is appended to the output.

On success a nonnegative value is returned. On error `EOF` is returned.

➢ STRING HANDLING/MANIPULATION FUNCTIONS:-

strcat( )	Concatenates two Strings
strcmp( )	Compares two Strings
strcpy( )	Copies one String Over another
strlen( )	Finds length of String

✓ strcat() function:

This function adds two strings together.

Syntax: char *strcat(const char *string1, char *string2);

strcat(string1,string2);
string1 = VERY
string2 = FOOLISH
strcat(string1,string2);
string1=VERY FOOLISH
string2 = FOOLISH

Strncat: Append n characters from string2 to stringl.
char *strncat(const char *string1, char *string2, size_t n);

✓ strcmp() function :

This function compares two strings identified by arguments and has a value 0 if they are equal. If they are not, it has the numeric difference between the first non-matching characters in the Strings.

Syntax: int strcmp (const char *string1,const char *string2);

strcmp(string1,string2);

Ex:- strcmp(name1,name2);
strcmp(name1,"John");
strcmp("ROM","Ram");

Strncmp: Compare first n characters of two strings.

int strncmp(const char *string1, char *string2, size_t n);

- ✓ **strcpy() function :**

It works almost as a string assignment operators. It takes the form

Syntax: char *strcpy(const char *string1,const char *string2);

strcpy(string1,string2);
string2 can be array or a constant.

Strncpy: Copy first n characters of string2 to stringl .

char *strncpy(const char *string1,const char *string2, size_t n);

- ✓ **strlen() function :**

Counts and returns the number of characters in a string.

Syntax:int strlen(const char *string);

n= strlen(string);

n→ integer variable which receives the value of length of string.

- ➢ **String/ Data Conversion:-**

  - ✓ **atof :-** convert a string to a double precision number.

Usage

```
Double_Type atof (String_Type s)
```

  - ✓ **atoi:-** convert a string to an integer

Usage

```
Int_Type atoi (String_Type str)
```

- ✓ **atol**:- convert a string to an long integer.

Usage

```
Long_Type atol (String_Type str)
```

- ✓ **char**:- convert a character code to a string.

Usage

```
String_Type char (Integer_Type c)
```

- ✓ **integer :-** Convert a string to an integer

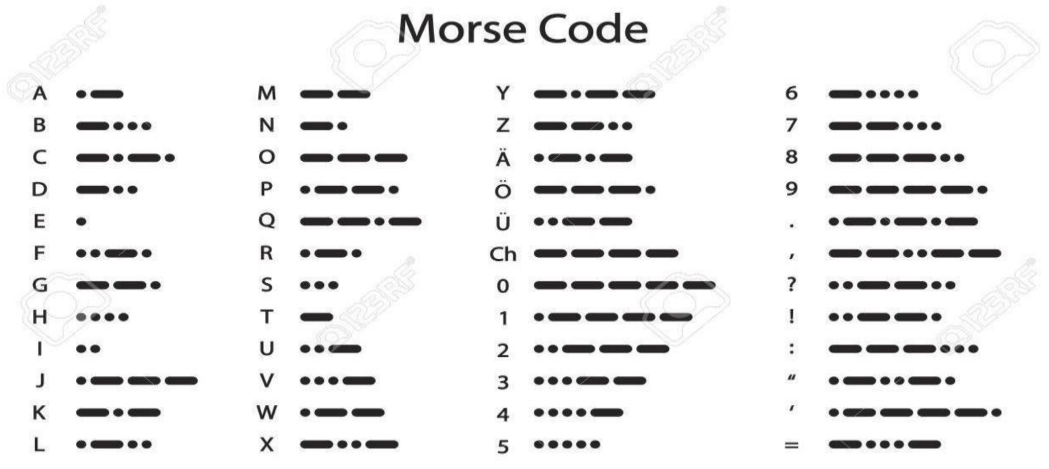
Usage

```
Integer_Type integer (String_Type s)
```

➢ A Programming Example – Morse Code:-

Morse code is a method of transmitting text information as a series of on-off tones, lights, or clicks that can be directly understood by a skilled listener or observer without special equipment.

It is named for Samuel F. B. Morse, an inventor of the telegraph.



Morse Code

A	•—	M	——	Y	—•——	6	—••••
B	—•••	N	—•	Z	——••	7	——•••
C	—•—•	O	———	Ä	•—•—	8	———••
D	—••	P	•——•	Ö	———•	9	————•
E	•	Q	——•—	Ü	••——	.	•—•—•—
F	••—•	R	•—•	Ch	————	,	——••——
G	——•	S	•••	0	—————	?	••——••
H	••••	T	—	1	•————	!	••——•
I	••	U	••—	2	••———	:	———•••
J	•———	V	•••—	3	•••——	"	•—••—•
K	—•—	W	•——	4	••••—	'	•————•
L	•—••	X	—••—	5	•••••	=	—•••—

We're gonna make an array with these values. Actually , just for the alphabet (a through z). The rest don't really matter that much.

So this is our array of strings:

char *u[36] = {".-","-...","-.-.","-..",".","..-.","--.","....","..",".---","-.-",".-..","--","-.","---",".--.","--.-",".-.","...","-","..-","...-",".--","-..-","-.--","--..","-----",".----","..---","...--","....-",".....","-....","--...","---..","----."};

Program:-

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

int main()
{
	int i;
	char input[255], morse['o ---', '--- o o o', '--- o --- o', '--- o o', 'o',
			'o o --- o', '--- --- o', 'o o o o', 'o o', 'o ----------',
						'--- o ---', 'o --- o o', '--- ---', '--- o', ' ---------- ',
						'o --- --- o', '--- --- o ---', 'o --- o', 'o o o', '---',
						'o o ---', 'o o o ---', 'o --- ---', '--- o o ---', '--- o ---
---',
						' ------ o o']

	printf("Enter your string: ");

	gets(string);

	for (i = 0; input[i] != '\0'; i++)
	{
		if ()

	}

	printf(" Your string in morse is: ");

	return 0;
}
```

➢ The Type Definition (Type def):-

- The C programming language provides a keyword called **typedef**, which you can use to give a type a new name.
- Following is an example to define a term **BYTE** for one-byte numbers –

  `typedef unsigned char BYTE;`
- After this type definition, the identifier BYTE can be used as an abbreviation for the type unsigned char, for example..

  `BYTE  b1, b2;`
- By convention, uppercase letters are used for these definitions to remind the user that the type name is really a symbolic abbreviation, but you can use lowercase, as follows –

  `typedef unsigned char byte;`

- You can use **typedef** to give a name to your user defined data types as well. For example, you can use typedef with structure to define a new data type and then use that data type to define structure variables directly as follows –

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

typedef struct Books {
   char title[50];
   char author[50];
   char subject[100];
   int book_id;
} Book;

int main( ) {
Book book;
   strcpy( book.title, "C Programming");
   strcpy( book.author, "Nuha Ali");
   strcpy( book.subject, "C Programming Tutorial");
   book.book_id = 6495407;

   printf( "Book title : %s\n", book.title);
   printf( "Book author : %s\n", book.author);
   printf( "Book subject : %s\n", book.subject);
   printf( "Book book_id : %d\n", book.book_id);

   return 0;
}
```

When the above code is compiled and executed, it produces the following result –

```
Book  title : C Programming
Book  author : Nuha Ali
Book  subject : C Programming Tutorial
Book  book_id : 6495407
```

➢ Enumerated Types:-

- Enumeration (or enum) is a user defined data type in C. It is mainly used to assign names to integral constants, the names make a program easy to read and maintain.

Here is the syntax of enum in C language

```
enum enum_name{const1, const2, ......... };
```

- The enum keyword is also used to define the variables of enum type. There are two ways to define the variables of enum type as follows.

1. **enum week{sunday, monday, tuesday, wednesday, thursday, friday, saturday};**
2. **enum week day;**

Example:-

```
#include<stdio.h>
enum week{Mon=10, Tue, Wed, Thur, Fri=10, Sat=16, Sun};
enum day{Mond, Tues, Wedn, Thurs, Frid=18, Satu=11, Sund};
int main() {
  printf("The value of enum week: %d\t%d\t%d\t%d\t%d\t%d\t%d\n\n",Mon ,
Tue, Wed, Thur, Fri, Sat, Sun);
  printf("The default value of enum day:
%d\t%d\t%d\t%d\t%d\t%d\t%d",Mond , Tues, Wedn, Thurs, Frid, Satu, Sund);
  return 0;
}
```

Output:-

```
The value of enum week: 10 11    12    13    10    16    17
The default value of enum day: 0 1     2     3     18    11    12
```

- In the above program, two enums are declared as week and day outside the main() function. In the main() function, the values of enum elements are printed.

➢ Structure:-

- Arrays allow to define type of variables that can hold several data items of the same kind. Similarly **structure** is another user defined data type available in C that allows to combine data items of different kinds.
- Structures are used to represent a record. Suppose you want to keep track of your books in a library. You might want to track the following attributes about each book –

  Title
  Author
  Subject
  Book ID

➢ Defining a Structure:-

- To define a structure, you must use the **struct** statement. The struct statement defines a new data type, with more than one member. The format of the struct statement is as follows –

```
struct [structure tag] {
member definition;
  member definition;
...
  member definition;
} [one or more structure variables];
```

- The **structure tag** is optional and each member definition is a normal variable definition, such as int i; or float f; or any other valid variable definition.

Accessing Structure Members:-

- To access any member of a structure, we use the **member access operator (.)**. The member access operator is coded as a period between the structure variable name and the structure member that we wish to access. You would use the keyword **struct** to define variables of structure type.

- The following example shows how to use a structure in a program −

```
#include  <stdio.h>
#include <string.h>
struct Books {
  char  title[50];
  char author[50];
  char subject[100];
  int  book_id;
};
 int main( ) {
  struct Books Book1;       /* Declare Book1 of type Book */
  struct Books Book2;       /* Declare Book2 of type Book */

  /* book 1 specification */
  strcpy( Book1.title, "C Programming");
  strcpy( Book1.author, "Nuha Ali");
  strcpy( Book1.subject, "C Programming Tutorial");
  Book1.book_id = 6495407;

  /* book 2 specification */
  strcpy( Book2.title, "Telecom Billing");
  strcpy( Book2.author, "Zara Ali");
  strcpy( Book2.subject, "Telecom Billing Tutorial");
  Book2.book_id = 6495700;

  /* print Book1 info */
  printf( "Book 1 title : %s\n", Book1.title);
  printf( "Book 1 author : %s\n", Book1.author);
  printf( "Book 1 subject : %s\n", Book1.subject);
  printf( "Book 1 book_id : %d\n", Book1.book_id);

  /* print Book2 info */
  printf( "Book 2 title : %s\n", Book2.title);
  printf( "Book 2 author : %s\n", Book2.author);
  printf( "Book 2 subject : %s\n", Book2.subject);
  printf( "Book 2 book_id : %d\n", Book2.book_id);
  return 0;  }
```

When the above code is compiled and executed, it produces the following result −

```
Book 1 title : C Programming
Book 1 author : Nuha Ali
Book 1 subject : C Programming Tutorial
Book 1 book_id : 6495407
Book 2 title : Telecom Billing
Book 2 author : Zara Ali
Book 2 subject : Telecom Billing Tutorial
Book 2 book_id : 6495700
```

Structures as Function Arguments:-

You can pass a structure as a function argument in the same way as you pass any other variable or pointer.

➢ Unions:-

- A **union** is a special data type available in C that allows to store different data types in the same memory location.
- You can define a union with many members, but only one member can contain a value at any given time.
- Unions provide an efficient way of using the same memory location for multiple-purpose.

Defining a Union:-

- To define a union, you must use the **union** statement in the same way as you did while defining a structure.
- The union statement defines a new data type with more than one member for your program.
- The format of the union statement is as follows −

```
union [union tag] {
  member definition;
  member definition;
  ...
  member definition;
} [one or more union variables];
```

The **union tag** is optional and each member definition is a normal variable definition, such as int i; or float f; or any other valid variable definition.

```
union Data {
  int i;
  float f;
  char str[20];
} data;
```

- Now, a variable of **Data** type can store an integer, a floating-point number, or a string of characters.
- It means a single variable, i.e., same memory location, can be used to store multiple types of data.
- You can use any built-in or user defined data types inside a union based on your requirement.
- The memory occupied by a union will be large enough to hold the largest member of the union. For example, in the above example, Data type will occupy 20 bytes of memory space because this is the maximum space which can be occupied by a character string.
- The following example displays the total memory size occupied by the above union –

```
#include  <stdio.h>
#include <string.h>
union Data {
  int i;
  float f;
  char str[20];
};
 int main( ) {
 union Data data;
  printf( "Memory size occupied by data : %d\n", sizeof(data));
 return 0;
}
```

When the above code is compiled and executed, it produces the following result −

```
Memory size occupied by data : 20
```

Accessing Union Members:-

- To access any member of a union, we use the **member access operator (.)**. The member access operator is coded as a period between the union variable name and the union member that we wish to access.

- You would use the keyword **union** to define variables of union type. The following example shows how to use unions in a program –

```
#include  <stdio.h>
#include <string.h>
union Data {
  int i;
  float f;
  char str[20];
};
 int main( ) {
 union Data data;
 data.i = 10;
  data.f = 220.5;
  strcpy( data.str, "C Programming");
   printf( "data.i : %d\n", data.i);
  printf( "data.f : %f\n", data.f);
 printf( "data.str : %s\n", data.str);
 return 0;
}
```

When the above code is compiled and executed, it produces the following result –

```
data.i : 1917853763
data.f : 4122360580327794860452759994368.000000
data.str : C Programming
```

➢ Programming Application:-

Mainly C Language is used for Develop Desktop application and system software. Some application of C language are given below.

- C programming language can be used to design the system software like operating system and Compiler.
- To develop application software like database and spread sheets.
- For Develop Graphical related application like computer and mobile games.

- To evaluate any kind of mathematical equation use c language.
- C programming language can be used to design the compilers.
- UNIX Kernal is completely developed in C Language.
- For Creating Compilers of different Languages which can take input from other language and convert it into lower level machine dependent language.
- C programming language can be used to design Operating System.
- C programming language can be used to design Network Devices.