C programming is a general-purpose, procedural, programming language developed in 1972 by Dennis M. Ritchie. Advantages of learning C Programming are:

- Easy to learn
- Structured language
- It produces efficient programs
- It can handle low-level activities

A C program basically consists of the following parts −

- Preprocessor Commands
- Functions
- Variables
- Statements & Expressions
- Comments

A variable is nothing but a name given to a storage area that our programs can manipulate. Each variable in C has a specific type, which determines the size and layout of the variable's memory; the range of values that can be stored within that memory; and the set of operations that can be applied to the variable. The name of a variable can be composed of letters, digits, and the underscore character. It must begin with either a letter or an underscore. A variable declaration provides assurance to the compiler that there exists a variable with the given type and name so that the compiler can proceed for further compilation.

Few Basic programs in C:

1. Program to add two numbers

```
#include <stdio.h>

void main() {

   int number1, number2, sum;

   printf("Enter two integers: ");

   scanf("%d %d", &number1, &number2);

   // calculating sum
```

```
    sum = number1 + number2;
    printf("%d + %d = %d", number1, number2, sum);
}
```

2. Program to swap two numbers

```
#include<stdio.h>
void main() {
      int first, second, temp;
      printf("Enter first number: ");
      scanf("%d", &first);
      printf("Enter second number: ");
      scanf("%d", &second);
      // Value of first is assigned to temp
      temp = first;
      // Value of second is assigned to first
      first = second;
      // Value of temp (initial value of first) is assigned to second
      second = temp;
      printf("\nAfter swapping, firstNumber = %d\n", first);
      printf("After swapping, secondNumber = %d", second);
      }
```

3. Program to find sum of n natural numbers

```
#include <stdio.h>
int main() {
    int n, i, sum = 0;
```

```
    printf("Enter a positive integer: ");
    scanf("%d", &n);
    for (i = 1; i <= n; ++i) {
        sum += i;
    }
    printf("Sum = %d", sum);
    return 0;
}
```

4. Linear search

```
#include <stdio.h>
int main()
{
  int array[100], search, c, n;
  printf("Enter number of elements in array\n");
  scanf("%d", &n);
  printf("Enter %d integer(s)\n", n);
  for (c = 0; c < n; c++)
    scanf("%d", &array[c]);
  printf("Enter a number to search\n");
  scanf("%d", &search);
for (c = 0; c < n; c++)
  {
    if (array[c] == search)    /* If required element is found */
    {
      printf("%d is present at location %d.\n", search, c+1);
```

```
      break;
    }
  }
  if (c == n)
  printf("%d isn't present in the array.\n", search);
  return 0;
}
```

5. Binary search

```
int main()
{
  int c, first, last, middle, n, search, array[100];

  printf("Enter number of elements\n");
  scanf("%d", &n);
  printf("Enter %d integers\n", n);
  for (c = 0; c < n; c++)
    scanf("%d", &array[c]);
  printf("Enter value to find\n");
  scanf("%d", &search);
  first = 0;
  last = n - 1;
  middle = (first+last)/2;
  while (first <= last) {
    if (array[middle] < search)
```

```
      first = middle + 1;
    else if (array[middle] == search) {
      printf("%d found at location %d.\n", search, middle+1);
      break;
    }
    else
      last = middle - 1;
    middle = (first + last)/2;
  }
  if (first > last)
    printf("Not found! %d isn't present in the list.\n", search);
  return 0;
}
```

6. Bubble sort

```
int main()
{
  int array[100], n, c, d, swap;

  printf("Enter number of elements\n");
  scanf("%d", &n);

  printf("Enter %d integers\n", n);

  for (c = 0; c < n; c++)
```

```
    scanf("%d", &array[c]);


  for (c = 0 ; c < n - 1; c++)
  {
    for (d = 0 ; d < n - c - 1; d++)
    {
      if (array[d] > array[d+1]) /* For decreasing order use < */
      {
        swap       = array[d];
        array[d]   = array[d+1];
        array[d+1] = swap;
      }
    }
  }


  printf("Sorted list in ascending order:\n");


  for (c = 0; c < n; c++)
     printf("%d\n", array[c]);


  return 0;
}
```

Structures

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 there is a need to keep track of books in a library. The user might want to track the following attributes about each book −

Title

Author

Subject

Book ID

For such a purpose a structure can be used.

Structure for the above requirement can be defined as:

```
struct Books {
   char  title[50];
   char  author[50];
   char  subject[100];
   int   book_id;
} book;
```

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.

A Sample structure program

Live Demo

```
#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;
}
```

C program to store student information using a structure

```
#include <stdio.h>
struct student {
    char firstName[50];
    int roll;
    float marks;
}

int main() {
    int i;
    struct student s[10];
    printf("Enter information of students:\n");

    // storing information
```

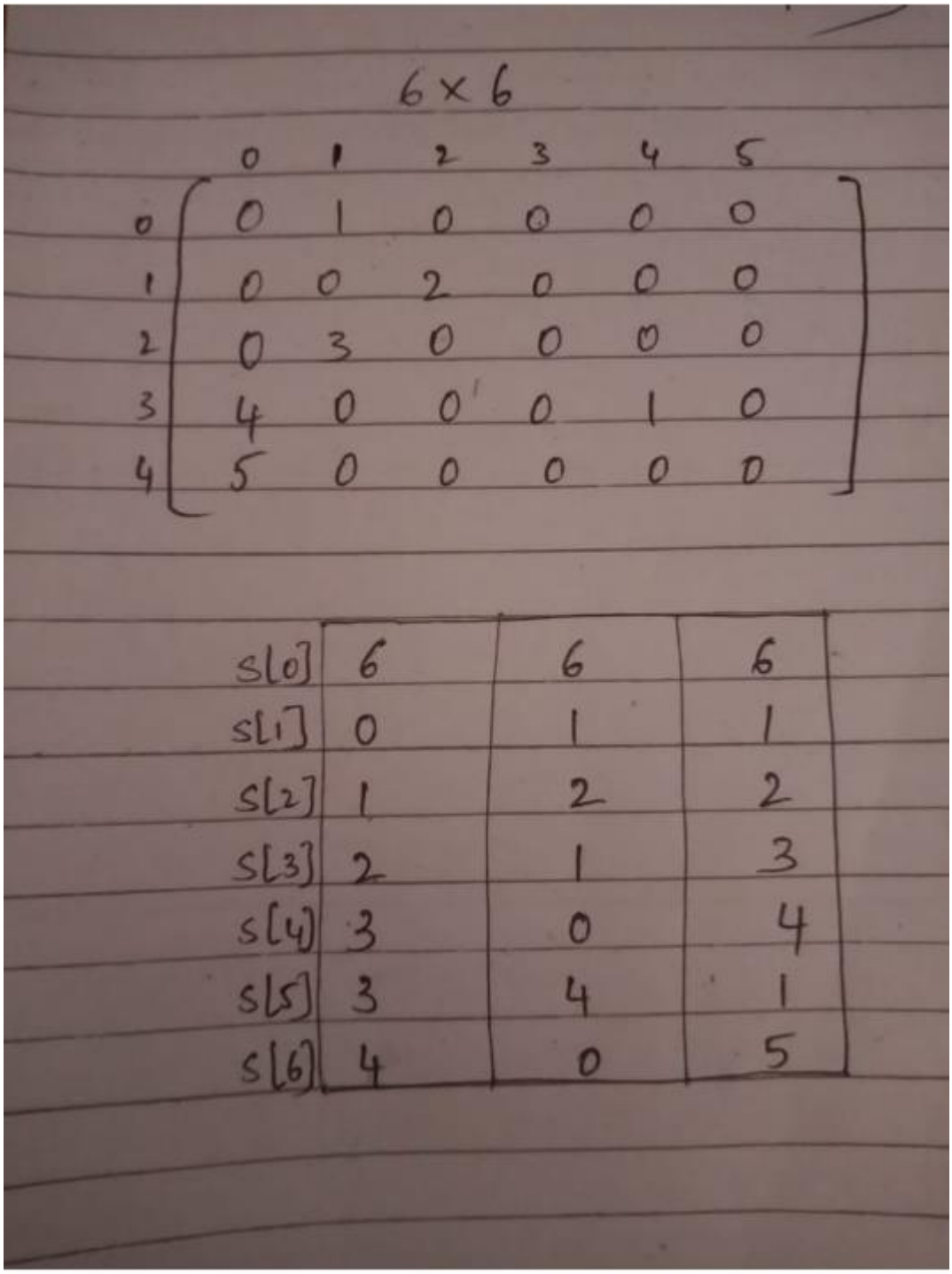
```
    for (i = 0; i < 5; i++) {
        s[i].roll = i + 1;
        printf("\nFor roll number%d,\n", s[i].roll);
        printf("Enter first name: ");
        scanf("%s", s[i].firstName);
        printf("Enter marks: ");
        scanf("%f", &s[i].marks);
    }
    printf("Displaying Information:\n\n");

    // displaying information
    for (i = 0; i < 5; ++i) {
        printf("\nRoll number: %d\n", i + 1);
        printf("First name: ");
        puts(s[i].firstName);
        printf("Marks: %.1f", s[i].marks);
        printf("\n");
    }
    return 0;
}
```

Sparse Matrix using structures

A sparse matrix is a matrix that has more zeros than the actual values. An m*n representation for such matrices will be a waste of space. Such matrices can be represented in a 3 tuple format.

Sparse Matrix

A matrix which has more number of zeroes than other values is called a sparse matrix. Representing it as a m*n array would result in wastage of large amount of space. Hence it is represented in 3 tuple format.



In the above example a sparse matrix is represented in a 3 tuple manner. Each row has 3 fields. The first row contains the information about the matrix. It has the number of rows, columns and number of non zero values in column one two and three respectively. The further rows have the row number in the first column, followed by column number in the second column and the actual value in the third column.

Such a representation can be programmatically created in C using structures.

Program to store a sparse matrix and search an element in it.

```
#include <stdio.h>


struct sparse
{
   int r, c, v;
};


int main()
{
   int m,n,i,j,x,k=1,key,flag=0;
         struct sparse s[10];


   printf("Enter the number of rows and columns\n");
   scanf("%d%d", &m,&n);


   printf ("Enter elements of matrix\n");


   for(i=0;i<m;i++)
      {
         for(j=0;j<n;j++)
         {
            scanf("%d",&x);
            if(x!=0)
```

```
            {
                s[k].r = i;
                s[k].c = j;
                s[k].v = x;
                k++;
            }

        s[0].r = m;
        s[0].c = n;
        s[0].v = k-1;
        }
    }

printf("Sparse matrix\n");
for(i=0;i<=s[0].v;i++)
{
    printf("%d\t%d\t%d\n", s[i].r,s[i].c,s[i].v);
}

printf("Enter key to be searched\n");
scanf("%d",&key);

for(i=0;i<=s[0].v;i++)
{
```

```
        if(s[i].v == key)
        {
            flag = 1;
            printf("key found at %d %d\n",s[i].r,s[i].c);
        }
    }
    if(flag==0)
        printf("key not found");
 return 0;
}
```

Write a program to generate n Fibonacci numbers

```
#include <stdio.h>
int fib(int n)
{
    if(n==0)
    return 0;
    if(n==1)
    return 1;
    return fib(n-1)+fib(n-2);
}
int main()
{
    int n,i;
    printf("Enter n");
    scanf("%d",&n);
    for(i=0;i<n;i++)
   {
```

```
        printf("%d\t",fib(i));
    }
    return 0;
}
```

Write a program to generate n Fibonacci numbers without recursion

```
#include<stdio.h>
int main()
{
 int n1=0,n2=1,n3,i,number;
 printf("Enter the number of elements:");
 scanf("%d",&number);
 printf("\n%d %d",n1,n2);          //printing 0 and 1
 for(i=2;i<number;++i)             //loop starts from 2 because 0 and 1 are already printed
 {
  n3=n1+n2;
  printf(" %d",n3);
  n1=n2;
  n2=n3;
 }
  return 0;
 }
```

Write a program to check if a number is prime or not.

```
#include<stdio.h>
int main(){
int n,i,m=0,flag=0;
printf("Enter the number to check prime:");
scanf("%d",&n);
m=n/2;
for(i=2;i<=m;i++)
{
```

```
if(n%i==0)
{
printf("Number is not prime");
flag=1;
break;
}
}
if(flag==0)
printf("Number is prime");
return 0;
 }
```

Write a program to read two matrix and perform multiplication of two matrices.

```
#include<stdio.h>
#include<stdlib.h>
int main(){
int a[10][10],b[10][10],mul[10][10],r,c,i,j,k;
system("cls");
printf("enter the number of row=");
scanf("%d",&r);
printf("enter the number of column=");
scanf("%d",&c);
printf("enter the first matrix element=\n");
for(i=0;i<r;i++)
{
for(j=0;j<c;j++)
{
scanf("%d",&a[i][j]);
}
}
printf("enter the second matrix element=\n");
for(i=0;i<r;i++)
{
```

```
for(j=0;j<c;j++)
{
scanf("%d",&b[i][j]);
}
}

printf("multiply of the matrix=\n");
for(i=0;i<r;i++)
{
for(j=0;j<c;j++)
{
mul[i][j]=0;
for(k=0;k<c;k++)
{
mul[i][j]+=a[i][k]*b[k][j];
}
}
}
//for printing result
for(i=0;i<r;i++)
{
for(j=0;j<c;j++)
{
printf("%d\t",mul[i][j]);
}
printf("\n");
}
return 0;
}
```

Introduction to Data Structures

Data structures are used to structure and organize the data such that accessing the data members in it is possible in a linear time. Different kinds of data structures are meant for different kinds of applications, and some are highly specialized to specific tasks. Data structures are used in almost every program or software system. Data structures are key building blocks of important algorithms. Specific data structures are essential ingredients of many efficient algorithms, and helps in easier management of huge amounts of data. Data structures are also reusable.

Stacks, queues, and lists are data collections with items ordered according to how they are added or removed. Once an item is added, it stays in the same position relative to its neighbours. Because of this characteristic, we call these collections as linear data structures.

Linear structures can be thought of as having two ends, referred to variously as “left” and “right”, “top” and “bottom”, or “front” and “rear”. What distinguishes one linear structure from another is where additions and removals may occur. For example, one data structure might only allow new items to be added at one end and removed at another; another may allow addition or removal from either end.

Introduction to stacks

A *stack* is an ordered collection of items where the addition of new items and the removal of existing items always takes place at the same end. This end is commonly referred to as the “top”. The most recently added item is always on the top of the stack and thus will be removed first. There are many examples of stacks in everyday situations. Consider a stack of plates on a table. It is only possible to add or remove plates to or from the top. Also consider a stack of books on a desk. The only book whose cover is visible is the one on top. To access the others, one must first remove the ones sitting on top of them.

One of the most useful features of stacks is that the insertion order is the reverse of the removal order. For example: Every web browser has a “Back” button. As the user navigates from page to page, the URLs of those pages are placed on a stack. The page currently being viewed is on the top, and the first page that user looked will be at the bottom of a stack. Clicking on the Back button helps the user move in the reverse order , through the stack of pages.

The basic operations performed on the stack include:

1) Push - Adds an item in the stack.

2) Pop- Removes an item from the stack. The items are popped in the reversed order in which they are pushed.
3) Peek or Top**:** Returns top element of stack.

Applications of stack:

1. Parsing
2. Expression Conversion
3. Redo-undo features at many places like editors, photoshop.
4. Forward and backward feature in web browsers
5. Used in many algorithms like Tower of Hanoi, tree traversals etc

Algorithm for PUSH operation

1. Check if the stack is full or not.
2. If the stack is full, then print error of overflow and exit the program.
3. If the stack is not full, then increment the top and add the element.

Algorithm for POP operation

1. Check if the stack is empty or not.
2. If the stack is empty, then print error of underflow and exit the program.
3. If the stack is not empty, then print the element at the top and decrement the top.

C Program to implement a stack using push(), pop() and display() functions

```
#include <stdio.h>
#include<conio.h>
#define size 50
int top = -1;
int stack[size];
void push(int item)
{
   if (top == size-1)
   printf("Stack overflow");
   else
   stack[++top]=item;
}
```

```
int pop()
{
   if (top == -1){
   return 0;
  }
   else
   return stack[top--];
}

void display()
{
   int i;
   if (top==-1)
   printf("Stack Empty");
   else
   {
      printf("The contents of stack\n");
      for(i=0;i<=top;i++)
      printf("%d\n", stack[i]);
   }
}
int main()
{
   int rpt=1, item, ch, element;
   while(rpt){
      printf("1:Insert 2: Delete 3:Display 4:Exit");
      scanf("%d",&ch);
      switch(ch)
      {
         case 1: printf("Enter element to be inserted\n");
                 scanf("%d",&item);
                 push(item);
```

```
            break;
        case 2: element = pop();
            if(element == 0)
            printf("Stack Underflow\n");
            else
            printf("%d is popped\n",element);
            break;
        case 3: display();
            break;
        default : exit(0);
      }
      printf("Do you want to continue 1:yes 0: No\n");
      scanf("%d", &rpt);
   }
  return 0;
}
```