**DATA STRUCTURES**

**LAB MANUAL**

**Subject /Course Code: 16IS38**

**Class: III Semester**

**Prepared By**

**Prof.GOURI PATIL**

**Miss.GANGA N DHARAK**

The Vision and Mission of PDA College of Engineering are as mentioned below.

**VISION**

To be an institute of excellence in technical education and research to serve the needs of the industry and society at local and global levels.

**MISSION**

* To provide a high-quality educational experience for students with values and ethics that enables them to become leaders in their chosen profession.
* To create, explore, and develop innovations in engineering and science through research and development activities.
* To provide beneficial service to the local, state, national and international industries and communities via educational, technical, and professional activities.

The Vision and Mission of Information Science & Engineering Department are as follows:

**VISION**

To impart quality education and research in Information Technology to produce a competent, committed and goal oriented workforce to fulfil the needs of the local and global requirements.

**MISSION**

The Department’s Mission is to advance knowledge in the Information and Computing Science by providing our students with the highest quality educational experience by

M1: Producing quality workforce with cutting edge technology.

M2: Engaging in innovative teaching learning, research and community service.

M3: Strengthening continuous interactions with alumni and industry and enable graduates to attain entrepreneurial qualities with life-long learning skills.

The program educational objectives of the B.E. in Information Science and Engineering program at PDA College of Engineering, Kalaburagi are to produce graduates who are able to:

PEO 1: Apply the principles of information and allied science, mathematics and scientific investigation to solve real world problems appropriate to the discipline.

PEO 2: Apply current industry accepted computing practices and emerging technologies to analyze, design, implement, and verify high quality IT-based solutions to real world problems.

PEO 3: Exhibit teamwork and effective communication skills**.**

PEO 4: Understand the ethical obligations, social impacts and apply their technical knowledge positively and appropriately in the course of career and professional journey**.**

PEO 5: Be successfully employed or accepted into a postgraduate program, and demonstrate a pursuit of lifelong learning**.**

**Program Specific Outcomes (PSOs)**

The graduates are able to:

**PSO1**: Demonstrate the working principles of computing systems and application Software

**PSO2**: Provide the suitable strategies and design solutions to solve IT related problems.

**PSO3**: Apply the professional practices to implement and deploy the real life applications

1. **Program Outcomes**

**Engineering Graduates will be able to:**

1. **Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability**: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one’s own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-long learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

**DATA STRUCTURES LAB MANUAL**

Subject /Course Code: 16IS38 Class: III Semester Credits: 01

CIE: 50Marks SEE: 50Marks Total Marks: 100

Hours/Week: 2Hrs (Practical)

**SYLLABUS OF OPERATING SYSTEMS LABORATORY**

|  |  |  |  |
| --- | --- | --- | --- |
| **Sl.No** | **Experiments** | **Program Outcomes Attained** | **Program Specific Outcomes Attained** |
| 01 | Write a C program to create a sequential file with at least five records.Each record having the structure show below:   |  |  |  |  |  | | --- | --- | --- | --- | --- | |  | Name | Marks1 | Marks2 | Marks3 | | Non-Zero  Positive | 25-Character | Positive Integer | Positive Integer | Positive Integer |  1. To display all the records in the file 2. To search for a specific record based on the USN. In case the record is not found. Suitable message should be displayed. Both the options in this case must be demonstrated. |  |  |
| 02 | Write a C program, which accepts the Internet protocol(IP)address in decimal dot format(ex 153.18.8.105) and convert it into 32-bit long inter(ex.2568095849)using strtok library function and unions. |  |  |
| 03 | Write a program using Recursion:  a)Solving the Tower of Hanoi problem  b) Searching an element on a given list of inters using the binary search method. |  |  |
| 04 | Write a program to perform Push, Pop and Display operations on a stack using  a)Linear array  b)Pointers. |  |  |
| 05 | Write a C program to convert and print a given valid parenthesized infix arithmetic expression to a postfix expression. The expression consists of single character operands and the binary operators +(plus), -(minus), \*(multiply) and /(divide). |  |  |
| 06 | Write a C program to evaluate a valid suffix/postfix expression using stack. Assume that the suffix/postfix expression is read as a single line consisting of non-negative single digit operands and binary arithmetic operators. The arithmetic operators are +(plus), -(minus), \*(multiply) and /(divide). |  |  |
| 07 | Write a program to perform Insert, Delete and Display operations on a queue using  a)Linear array  b)Pointers |  |  |
| 08 | Write a C program to simulate the working of a circular queue of integers using an array. Provide the following operations  a)Insert b)Delete c)Display |  |  |
| 09 | Write a C program using dynamic variables and pointers, to construct a singly linked list consisting of the following information in each node: student id (interger), student name(character string) and semester(integer). The operations to be supposed are:   1. The insertion operation   i)At the front of a list  ii) At the back of the list  iii) At any position in the list   1. Deleting a node based on student id. If the specified node is not present in the list an error message should be displayed. Both the operation should be demonstrated. 2. Searching a node based on student id and update the information content. If the specified node is not present in the list an error message should be displayed. Both situations should be displayed. 3. Displaying all the nodes in the list. |  |  |
| 10 | Write a C program to support the following operations on a doubly linked list where each node consists of integers:   1. Create a doubly linked list by adding each node at the front 2. Insert a new node to the left of the node whose key value is read as an input. 3. Delete the node of a given data, if it is found, otherwise display appropriate message 4. Display the contents of the list. |  |  |
| 11 | Write a C program using dynamic variables and pointers to construct a stack of integers using singly linked list and to perform the following operations.   1. PUSH 2. POP 3. Display   The program should print appropriate messages for stack overflow, and stack empty. |  |  |
| 12 | Write a C program using dynamic variables and pointers to construct a queue of integers using singly linked list and to perform the following operations.  a)Insert  b)Delete  c)Display  The program should print appropriate messages for queue full and queue empty. |  |  |
| 13 | Write a C program   1. To construct a binary search three of integers. 2. To traverse the tree using all the methods i.e, inorder, preorder and postorder. |  |  |
| 14 | Write a C program to evaluate an expression three. |  |  |

**Data Structures:**

**Definition :** The logical inter-relation between elementary data items is called as **data structure**. The basic data items include integers, bits and characters. **Basically it deals with manipulation and organization of data, solving problems with computer involves data manipulation.** But the data available will usually be in amorphous form. When different types of such data are related to each other, then we call it to be a data structure.

**Types of Data Structures** : 1)**Linear Data Structure :** Stacks, Queues, Linked Lists,etc. 2)**Non-linear Data Structure :** Trees, Graphs, etc.

**Advantages**: The major advantages of data structures are:   
• It gives different level of organizing data.  
• It tells how data can be stored and accessed in its elementary level

Different kind of data structures are suited for different kinds of applications and some are highly specialized to certain tasks. For ex: B-Trees are particularly well suited for implementation of database.

1. **Preliminary programs using pointers:**

* **String operations**
* **To swap two integer values**
* **To reverse an integer array**
* **To perform binary search**
* **Arrange names in alphabetical order**

**String Handling**:

 Strings are defined as an array of characters or a pointer to a portion of memory containing ASCII characters. A string in C is a sequence of zero or more characters followed by a NULL (\0)character:

http://www.cs.cf.ac.uk/Dave/C/string.gif

It is important to preserve the NULL terminating character as it is how C defines and manages variable length strings. All the C standard library functions require this for successful operation.

**Basic String Handling Functions**

All the string handling functions are prototyped in:

#include <string.h>

The common functions are described below:

char \*stpcpy (const char \*dest,const char \*src) 🡪 Copy one string into another.   
int strcmp(const char \*string1,const char \*string2) 🡪 Compare string1 and string2 to determine alphabetic order.   
char \*strcpy(const char \*string1,const char \*string2) 🡪 Copy string2 to string1.   
char \*strerror(int errnum) 🡪 Get error message corresponding to specified error number.   
int strlen(const char \*string) 🡪 Determine the length of a string.   
char \*strncat(const char \*string1, char \*string2, size\_t n) 🡪 Append n characters from string2 to stringl.   
int strncmp(const char \*string1, char \*string2, size\_t n) 🡪 Compare first n characters of two strings.   
char \*strncpy(const char \*string1,const char \*string2, size\_t n) 🡪 Copy first n characters of string2 to stringl .   
int strcasecmp(const char \*s1, const char \*s2) 🡪 case insensitive version of strcmp().   
int strncasecmp(const char \*s1, const char \*s2, int n) 🡪 case insensitive version of strncmp().

**Pointer :**

A pointer is a variable which contains the address in memory of another variable. We can have a pointer to any variable type.

The *unary* or *monadic* operator & gives the ``address of a variable''. The *indirection* or dereference operator \* gives the ``contents of an object *pointed to* by a pointer''.

To declare a pointer to a variable do:

   int \*pointer;

**Pointer and Functions :**

Let us examine the close relationship between pointers and functions.

When C passes arguments to functions it passes them by value.

There are many cases when we may want to alter a passed argument in the function and receive the new value back once to function has finished. C uses pointers explicitly to do this.

The best way to study this is to look at an example where we must be able to receive changed parameters.

Let us try and write a function to swap variables around.

Pointers provide the solution it Passes the address of the variables to the functions and access address of function*.*

Thus our function call in our program would look like this:

    swap(&a, &b)

The Code to swap is :

    void swap(int \*px, int \*py)

   { int temp;

temp = \*px;

/\* contents of pointer \*/

\*px = \*py;

\*py = temp;

}We can return pointer from functions.

**Pointers and Arrays :**

Pointers and arrays are very closely linked in C. Consider the following:

 int a[10], x;

int \*pa;

  pa = &a[0]; /\* pa pointer to address of a[0] \*/

  x = \*pa;

/\* x = contents of pa (a[0] in this case) \*/

 http://www.cs.cf.ac.uk/Dave/C/arrays.gif

Fig.  Arrays and Pointers

To get somewhere in the array using a pointer we could do:

   pa + i $\equiv$a[i]

C however is much more subtle in its link between arrays and pointers.

For example we can just type

   pa = a;

instead of

    pa = &a[0]

and a[i] can be written as \*(a + i).   
i.

We also express pointer addressing like this:

   pa[i] =\*(pa + i).

When an array is passed to a function what is actually passed is its initial elements location in memory.

 strlen(s) =strlen(&s[0])

This is why we declare the function:

      int strlen(char s[ ]);

An equivalent declaration is : int strlen(char \*s);   
since char s[ ] =char \*s.

strlen( ) is a *standard library* function that returns the length of a string. Let's look at how we may write a function:

    int strlen(char \*s)

   { char \*p = s;

  while (\*p != ‘\0’);

   p++;

return p-s;

   }

Now lets write a function to copy a string to another string. strcpy() is a standard library function that does this.

    void strcpy(char \*s, char \*t)

   {  while ( (\*s++ = \*t++) != ‘\0’);}

This uses pointers and assignment by value.

**i) Algorithm for String operations:**

Step 1: Read string 1 s1 & string 2 s2.

Step 2: Declare the three pointer variables p,q,r. And make p=s1, q=s2, r=s3.

Step 3: Initialise i=0, scan string s1until \*(p+i)!=’\0’ , increment i by 1 and calculate the length of string s1 as len1.

Initialise j=0, scan string s2 until \*(q+j)!=’\0’ , increment j by 1 and calculate the length of string s2 as len2.

Make \*(r+i+j)=’\n’;

Step 4: print ‘ the concatenated string is’

Initialize i=0, print each character of concatenated string \*(r+i) , increment i by 1, until \*(r+i)!=’\n’

Step 5: stop.

**ii) Algorithm for to swap two integer values:**

Step 1: Read two integer numbers num1 & num2.

Step 2: print value of num1 and value of num2 before swapping.

Step 3:Use temporary variable t to swap the values of num1,num2

t=\*num1;

\*num1=\*num2;

\*num2=t;

Step 4: print value of num 1 and value of num 2 after swapping.

Step 5: stop.

**iii) Algorithm for to To reverse an integer array:**

Step 1: Read integer array a of size n .

Step 2:Declare pointer variable p. And make p=a.

Step 3:Initialise the value of i=0

Step 4: print the array elements (\*p+i) in given order,

increment i by 1

Step 5: Repeat step 4 untill(i<n)

Step 6:Initialise the value of i=n-1

print the array elements(\*p+i) in reverse order

decrement i by 1

Step 7: Repeat step 6 until(i>=0)

step 5: stop.

**iv) Algorithm to perform binary search:**

Step 1: Read integer array a of size n .

Step 2: Initialize low=1, high=n, read the key to be searched and call the binary search sub procedure. .

Step 3: Check if low is less than high, then find mid value as mid=(low+high)/2;

Step 4: Compare key with mid value, if it is equal then return that value and display as key found

Step 5: if key is less than mid then call binary search sub routine with its values of low and mid-1.

Step 6: Failing step 4, call sub procedure with the values of mid+1 and high.

Step 7: Repeat step 4 to 6 until key found.

Step 8: Failing step 7, display the message as key not found.

Step 9: stop.

**Structures and Unions**:  
Arrays are used to store large set of data and manipulate them but the disadvantage is that all the elements stored in an array are to be of the same data type. When we require using a collection of different data items of different data types we can use a structure. A structure is a convenient method of handling a group of related data items of different data types.

structure definition:   
general format:   
struct tag\_name   
{   
data type member1;   
data type member2;   
…   
…   
}   
  
**Example:**struct lib\_books   
{   
char title[20];   
char author[15];   
int pages;   
float price;   
};   
  
the keyword struct declares a structure to holds the details of four fields namely title, author pages and price. These are members of the structures. Each member may belong to different or same data type. The tag name can be used to define objects that have the tag names structure. The structure we just declared is not a variable by itself but a template for the structure.   
  
We can declare structure variables using the tag name any where in the program. For example the statement,   
  
struct lib\_books book1,book2,book3;   
  
declares book1,book2,book3 as variables of type struct lib\_books each declaration has four elements of the structure lib\_books. The complete structure declaration might look like this   
  
struct lib\_books   
{   
char title[20];   
char author[15];   
int pages;   
float price;   
};   
  
struct lib\_books, book1, book2, book3;   
  
structures do not occupy any memory until it is associated with the structure variable such as book1, the template is terminated with a semicolon. While the entire declaration is considered as a statement, each member is declared independently for its name and type in a separate statement inside the template. The tag name such as lib\_books can be used to declare structure variables of its data type later in the program.   
  
We can also combine both template declaration and variables declaration in one statement, the declaration   
  
struct lib\_books   
{   
char title[20];   
char author[15];   
int pages;   
float price;   
} book1,book2,book3;   
is valid. The use of tag name is optional for example   
struct   
{   
…   
…   
…   
}   
  
book1, book2, book3 declares book1,book2,book3 as structure variables representing 3 books but does not include a tag name for use in the declaration.   
  
A structure is usually defines before main along with macro definitions. In such cases the structure assumes global status and all the functions can access the structure.

**Functions and structures:**

We can pass structures as arguments to functions. Unlike array names however, which always point to the start of the array, structure names are not pointers. As a result, when we change structure parameter inside a function, we don’t effect its corresponding argument.   
  
**Passing structure to elements to functions:**A structure may be passed into a function as individual member or a separate variable.   
A program example to display the contents of a structure passing the individual elements to a function is shown below.   
  
# include <stdio.h>   
void main()   
{   
int emp\_id;   
char name[25];   
char department[10];   
float salary;   
};   
  
static struct emp1={125,”sampath”,”operator”,7500.00};   
/\* pass only emp\_id and name to display function\*/   
display(emp1.emp\_id,emp1.name);   
}   
/\* function to display structure variables\*/   
display(e\_no,e\_name)   
int e\_no,e\_name;   
{   
printf(“%d%s”,e\_no,e\_name);   
}  
in the declaration of structure type, emp\_id and name have been declared as integer and character array. When we call the function display() using display(emp1.emp\_id,emp1.name);   
we are sending the emp\_id and name to function display(0);   
it can be immediately realized that to pass individual elements would become more tedious as the number of structure elements go on increasing a better way would be to pass the entire structure variable at a time.

**Passing entire structure to functions:**

In case of structures having to having numerous structure elements passing these individual elements would be a tedious task. In such cases we may pass whole structure to a function as shown below:   
  
# include <stdio.h>   
{   
int emp\_id;   
char name[25];   
char department[10];   
float salary;   
};   
  
void main()   
{   
static struct employee emp1=   
{   
12,   
“sadanand”,   
“computer”,   
7500.00   
};   
  
/\*sending entire employee structure\*/   
display(emp1);   
}   
  
/\*function to pass entire structure variable\*/   
display(empf)   
struct employee empf   
{   
printf(“%d%s,%s,%f”, empf.empid,empf.name,empf.department,empf.salary);   
}

**Arrays of structure:**

It is possible to define a array of structures for example if we are maintaining information of all the students in the college and if 100 students are studying in the college. We need to use an array than single variables. We can define an array of structures as shown in the following example:   
  
structure information   
{   
int id\_no;   
char name[20];   
char address[20];   
char combination[3];   
int age;   
}   
student[100];   
  
An array of structures can be assigned initial values just as any other array can. Remember that each element is a structure that must be assigned corresponding initial values.

**2.. Algorithm to calculate average of students marks for a student database:**

Step 1: Read student name, roll no., & marks of three subjects m1,m2,m3 in students structure s[i].

Step 2: initialize i=0;

Step 3: calculate the average of students marks until(i!=n)

{

sum[i]+=s[i].m1+s[i].m2+s[i].m3;

temp[i]= sum[i]/3;

increment i by 1.

}

temp[i]= (s[i].m1+s[i].m2+s[i].m3)/3;

Step 4: initialize i=0.

Step 5: print the standard information and average

Print name, roll number, average.

Increment i by 1.

Step 6: if (i!=n) repeat step 5.

Step 5: stop.

**Algorithm for Binary Search using recursion:**

Step 1: Initialize low=1, high=n, read the key to be searched and call the binary search sub procedure.

Step 2: Check if low is less than high, then find the mid value as mid = (low + high) / 2.

Step 3: Compare key with mid value, if it is equal then return that value and display as key found.

Step 4: If key is less than mid then call binary search routine recursively with its values of low and mid-1.

Step 5: Failing step 4, call binary search routine recursively with its values of mid+1 and high.

Step 6: Repeat steps 3 to 5 until key found.

Step 7: Failing step 6, display the message as key not found.

Step 8: stop.

**LAB PROGRAMS:**

1. Write a C program to create a sequential file with at least five records. Each record having the structure show below:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Name | Marks1 | Marks2 | Marks3 |
| Non-Zero  Positive | 25-Character | Positive Integer | Positive Integer | Positive Integer |

1. To display all the records in the file
2. To search for a specific record based on the USN. In case the record is not found. Suitable message should be displayed. Both the options in this case must be demonstrated.

Program:

#include<stdio.h>

#include<conio.h>

#include<string.h>

struct student

{

char name[10];

int usn;

int marks1,marks2,marks3;

};

main()

{

int n,avg[10],i,j;

struct student a[10];

clrscr();

printf("enter the number of student:\n");

scanf("%d",&n);

printf("enter the details of student\n");

for(i=0;i<n;i++)

{

printf("name=");

scanf("%s",a[i].name);

printf("usn=");

scanf("%d",&a[i].usn);

printf("marks1=");

scanf("%d",&a[i].marks1);

printf("marks2=");

scanf("%d",&a[i].marks2);

printf("marks3=");

scanf("%d",&a[i].marks3);

avg[i]=(a[i].marks1+a[i].marks2+a[i].marks3)/3;

}

printf("the details of students is:\n");

for(i=0;i<n;i++)

{

printf("name\t usn\t marks1\t marks2\t marks3\t average\n");

printf("%s\t %d\t %d\t %d\t %d\t %d\n",a[i].name,a[i].usn,a[i].marks1,a[i].marks2,a[i].marks3,avg[i]);

getch();

}

return;

}

OUTPUT:

Enter the numbers of students

Enter details of student

Name Krutika

USN 3PD17IS017

Marks1 90

Marks2 80

Marks3 90

Name Krishna

USN 3PD17IS019

Marks1 70

Marks2 80

Marks3 80

Name Krishma

USN 3PD17IS020

Marks1 60

Marks2 80

Marks3 100

Name usn marks1 marks2 marks3 Average

Krutika 3PD17IS017 90 80 90 88.3

Krishna 3PD17IS019 70 80 80 76.6

Krishma 3PD17IS020 60 80 100 80.0

1. Write a C program, which accepts the Internet protocol(IP)address in decimal dot format(ex 153.18.8.105) and convert it into 32-bit long inter(ex.2568095849)using strtok library function and unions.

Program:

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

typedef union

{

unsigned char chaddr[4];

unsigned long numaddr;

}IP\_Addr;

int main()

{

IP\_Addr addr;

char \*parser;

char straddr[16]="153.18.8.105";

int i;

clrscr();

parser=strtok(straddr,".");

addr.chaddr[3]=strtol(parser,(char \*\*)NULL,10);

for(i=2;i>=0;i--)

{

parser=strtok(NULL,".");

addr.chaddr[i]=strtol(parser,(char \*\*)NULL,10);

}

printf("IP decimal dot:%d%d%d%d\n",addr.chaddr[3],addr.chaddr[2],addr.chaddr[1],

addr.chaddr[0]);

printf("IP binary:%lu\n",addr.numaddr);

getch();

return 0;

}

Output:

IP Decimal : 153188105

IP Binary : 2568095849

1. Write a C program using Recursion: a) Solving the Tower of Hanoi problem.

The Tower of Hanoi is a mathematical game or puzzle. It consists of three rods, and a number of disks of different sizes which can slide onto any rod. The puzzle starts with the disks in a neat stack in ascending order of size on one rod, the smallest at the top, thus making a conical shape. The objective of the puzzle is to move the entire stack to another rod, obeying the following simple rules: Only one disk can be moved at a time.

• Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack i.e. a disk can only be moved if it is the uppermost disk on a stack.

* No disk may be placed on top of a smaller disk.

• With three disks, the puzzle can be solved in seven moves. The minimum number of moves required to solve a Tower of Hanoi puzzle is 2 n - 1, where n is the number of disks.

Program:

#include<stdio.h>

#include<conio.h>

main()

{

int n;

clrscr();

printf("Enter the number of discs\n");

scanf("%d",&n);

towers(n,'A','C','B');

getch();

return;

}

towers(int n,char scr,char dst,char temp)

{

if(n==1)

printf("Move disc %d from %c to %c\n",n,scr,dst);

else

{

towers(n-1,scr,temp,dst);

printf("Move disc %d from %c to %c\n",n,scr,dst);

towers(n-1,temp,dst,scr);

}

return;

}

Output:

Enter number of Disc

3

Move disc 1 from A to C

Move disc2 from A to B

Move disc1 from C to A

Move disc3 from A to C

Move disc1 from B to A

Move disc2 from B to C

Move disc1 from A to C

Total Number of moves are: 7

3 b) Searching an element on a given list of inters using the binary search method.

Program:

#include<stdio.h>

#include<conio.h>

void main()

{

int i,item,a[10],n,pos;

clrscr();

printf("enter the number of element");

scanf("%d",&n);

printf("enter the elements");

for(i=0;i<n;i++)

scanf("%d",&a[i]);

printf("enter the element to be searched\n");

scanf("%d",&item);

search(item,a,n,&pos);

if(pos==-1)

printf("element not found");

else

printf("element found at posotion %d",pos+1);

getch();

}

search(int item,int\*a,int n,int\*pos)

{

int low,high,mid;

low=0;

high=n-1;

while(low<=high)

{

mid=(low+high)/2;

if(item==\*(a+mid))

{

\*pos=mid;

return;

}

if(item<\*(a+mid))

high=mid-1;

else

low=mid+1;

}

\*pos=-1;

return;

}

Output:

Enter the number of elements

5

Enter the elements

11 ,12,13,14,15

Enter the elements to be searched

14

Element found at position 4

Enter element to be searched

6

Element not found

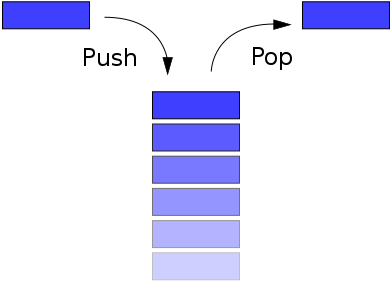
4) Write a program to perform push, pop, and display operations on stack using i) linear array ii)Pointers

**Definition of stack**

Stack is a non primitive data structure which works based on principle of  last in first out (LIFO). It is an ordered list in which addition of a new item or deletion of an already existing data item is done at one end. And this end is known as TOP of the stack. In computing world, stack data structure can be applied in many applications such as parsing syntax of expressions, run-time memory management (used in Java virutal machine) and solving search problem.

**Stack operations**

Push and pop are the operations that are provided for insertion of an element into the stack and the removal of an element from the stack.



**Representation of stack**

Stack can be represented in the memory by using either the static representation i.e. array representation or the dynamic representation i.e. linked list representation.

**Algorithm to Insert/Push operation:**

Step 1: As we insert an item we must take care of the overflow situation i.e., when top reaches STACK\_SIZE-1, stack results in overflow condition and appropriate error message has to be returned.

if ( top ==STACK\_SIZE-1 )

{

printf( “Stack Overflow\n” );

return;

}

Here, STACK\_SIZE should be #defined (preprocessor directive) and is called symbolic constant.

Step 2: If Step 1 fails, the item has to be inserted. First increment top by 1 and then insert an item.

top = top + 1;

s[\*top] = item;

OR

S[++(\*top)] = item;

**Algorithm to Delete/Pop Operation:**

Step 1: During deleting an item from the stack we are checking for an extreme condition for underflow.

if ( top == -1 )

{

return 0;

}

Step 2: Deleting an item from the top of the stack can be achieved by first accessing the top element s[\*top] and then decrementing top by one.

item\_deleted = s[(\*top)--];

**Algorithm for displaying the contents of the stack:**

Step 1: If there are no items present in the stack, the appropriate error message is displayed.

if ( top == -1 )

{

printf( “Stack is empty\n” );

return;

}

Step 2: Otherwise, Display contents of the stack

for( i=0; i<=top; i++ )

printf(“%d\n”,s[i]);

4) Write a program to perform push, pop, and display operations on stack using i) linear array ii)Pointers

I) linear array program:

#include<stdio.h>#include<conio.h>int i,top,ch,s[3],item;void push(),pop(),dis();

main(){clrscr();top=-1;ch=0;while(ch!=4){ printf("\nEnter the menu for stack operation\n"); printf("\n1:insert\n2:delete\n3:display\n4:exit\n"); printf("Input ur choice\n"); scanf("%d",&ch); switch(ch) { case 1: push(); break; case 2: pop(); break; case 3: dis(); break; case 4: exit(0); break; default: printf("Invalid choice\n"); } }getch();return 0;}

void push() { int item; if(top==3-1) { printf("Stack Overflow\n"); return; } printf("Enter an item to be pushed:"); scanf("%d",&item); top+=1; s[top]=item; }

void pop() { if(top==-1) { printf("Stack Undeflow\n"); return; } printf("Item poped is %d\n",s[top--]); }

void dis() { if(top==-1) { printf("Stack is empty\n"); return; } printf("\nStack contains......\n"); for(i=0; i<=top; i++) printf("%d\t",s[i]); }

1. Write a program to perform push, pop, and display operations on a stack using

ii)Pointers

#include<stdio.h>

int s[10],top=-1,STACK\_SIZE=3;

int main(void)

{

int ch=1,a,itdel;

clrscr();

while(ch)

{

printf("Enter ur choice\n");

printf("1:push\t 2:pop\t 3:display\t 4:exit\n");

printf("choice?");

scanf("%d",&ch);

switch(ch)

{

case 1:printf("enter the number to be pushed\n");

scanf("%d",&a);

push(a,&top,s);

break;

case 2:itdel=pop(&top,s);

if(itdel==0)

printf("stack is empty");

else

printf("item deleted=%d",itdel);

break;

case 3:display(top,s);

break;

case 4:exit();

break;

default:printf("wrong choice");

break;

}

}

return;

}

int push(int item,int \*top,int s[])

{

if(\*top==STACK\_SIZE-1)

{

printf("stack overflow");

return;

}

s[++(\*top)]=item;

return;

}

int pop(int \*top,int s[])

{

int item\_deleted;

if(\*top==-1)

{

return 0;

}

item\_deleted=s[(\*top)--];

return item\_deleted;

}

int display(int top,int s[])

{

int i;

if(top==-1)

{

printf("stack is empty\n");

return;

}

printf("contents of stack");

for(i=0;i<=top;i++)

{

printf("%d",s[i]);

}

return;

}

Output:

1. Write a C program to convert and print a given valid parenthesized infix arithmetic expression to a postfix expression. The expression consists of single character operands and the binary operators +(plus), -(minus), \*(multiply) and /(divide).

Objectives:

**Applications of Stack**

**Infix, Postfix expressions**

An expression in a programming language is a meaningful combination of operands and operations. The operands may be of type int, float or double. The operators may be arithmetic, logical, relational or bitwise. Here we use arithmetic expressions addition, subtraction, multiplication and division.

Infix is the form of an arithmetic expression in which we fix (place) the arithmetic operator in between the two operands. This is the usual notation of writing mathematical expressions.

Example: (A+B)-C

In postfix from of an arithmetic expression in which we fix the arithmetic operator after (Post) its two operands. The postfix notation is called suffix notation.

Example: Infix Postfix

A+B\*C ABC\*+

**Representation of Expression**

|  |  |  |
| --- | --- | --- |
| **Operation** | **Operators** | **Precedence** |
| Exponential | ^ | Highest |
| Multiplication/division | \*, / | Next |
| Addition/Subtraction | +,- | Last |

a) An arithmetic expression is evaluated from LEFT to RIGHT.

b) Exponentiation operation is evaluated from RIGHT to LIFT.

c) In ^ is an Exponentiation Operators.

d) Parenthesized expression is evaluated FIRST.

**Infix to Postfix Conversion :**   
In normal algebra we use the infix notation like a+b\*c. The corresponding postfix notation is abc\*+. The algorithm for the conversion is as follows :

**Algorithm for Infix to Postfix Conversion:**

Step 1: Scan the Infix string from left to right.

Step 2: Initialize an empty stack.

Step 3: If the scanned character is an operand, add it to the Postfix string. If the scanned character is an operator and if the stack is empty Push the character to stack.

Step 4: If the scanned character is an Operand and the stack is not empty, compare the precedence of the character with the element on top of the stack (topStack). If topStack has higher precedence over the scanned character Pop the stack else Push the scanned character to stack. Repeat this step as long as stack is not empty and topStack has precedence over the character.

Repeat this step till all the characters are scanned.

Step 5: (After all characters are scanned, we have to add any character that the stack may have to the Postfix string.) If stack is not empty add topStack to Postfix string and Pop the stack. Repeat this step as long as stack is not empty.

Step 6: Return the Postfix string.

Step 7: stop.

Program:

#include<stdio.h>

#include<conio.h>

#include<string.h>

int stack[50],top=-1;

char postfix[25];

int main(void)

{

char infix[25];

clrscr();

printf("enter a valid infix expression:");

gets(infix);

infix\_to\_postfix(infix);

getch();

return 0;

}

infix\_to\_postfix(char infix[])

{

int length;

static int index=0,pos=0;

char symbol,temp,postfix[50];

length=strlen(infix);

push('#');

while(index<length)

{

symbol=infix[index];

switch(symbol)

{

case'(':push(symbol);

break;

case')':temp=pop();

while(temp!='(')

{

postfix[pos]=temp;

pos++;

temp=pop();

}

break;

case'+':

case'-':

case'\*':

case'/':while(preced(stack[top])>=preced(symbol))

{

temp=pop();

postfix[pos]=temp;

pos++;

}

push(symbol);

break;

default:postfix[pos++]=symbol;

break;

}

index++;

}

while(top>0)

{

temp=pop();

postfix[pos++]=temp;

}

postfix[pos++]='\0';

printf("the postfix of %s is=%s",infix,postfix);

return;

}

int preced(char ch)

{

switch(ch)

{

case'\*':

case'/':return 2;

case'+':

case'-':return 1;

case'(':

case')':return 0;

case'#':return -1;

}

return 0;

}

push(char ch)

{

top++;

stack[top]=ch;

return;

}

pop()

{

char ch;

ch=stack[top];

top--;

return ch;

}

Output:

* 1. Enter valid infix expression

a+b

Postfix expression

ab+

* 1. Enter valid infix expression

a +b\*cd

Postfix expression

abcd \*+

1. Write a C program to evaluate a valid suffix/postfix expression using stack. Assume that the suffix/postfix expression is read as a single line consisting of non-negative single digit operands and binary arithmetic operators. The arithmetic operators are +(plus), -(minus), \*(multiply) and /(divide).

Program:

#include<stdio.h>

#include<math.h>

#include<string.h>

#include<ctype.h>

double compute(char symbol,double op1,double op2)

{

switch(symbol)

{

case'+':return op1+op2;

case'-':return op1-op2;

case'\*':return op1\*op2;

case'/':return op1/op2;

case'$':

case'^':return pow(op1,op2);

}

return 0;

}

void main()

{

double s[20];

double res;

double op1;

double op2;

int top;

int i;

char postfix[20];

char symbol;

printf("Enter the postfix expression\n");

scanf("%s",postfix);

top=-1;

for(i=0;i<strlen(postfix);i++)

{

symbol=postfix[i];

if(isdigit(symbol))

s[++top]=symbol-'0';

else

{

op2=s[top--];

op1=s[top--];

res=compute(symbol,op1,op2);

s[++top]=res;

}

}

res=s[top--];

printf("The result is %f\n",res);

}

Output:

Enter your postfix expression

82+

Res: 10

7) Write a program to perform Insert, Delete and Display operations on a queue using

a)Linear array 7b)Pointers

**Definition of Queue:**

A queue is a non-primitive linear data structure. It is an ordered homogeneous group of elements in which element are added at one end, called the rear, and they are deleted at the other end, called the front.

In queue, elements can only be removed from the queue in the order in which they are inserted into the queue. That is the first element added to the queue is the first element to be removed. Therefore, a queue is often called a First-in, First-out (FIFO) data structure.

**Representation of Queue**

Queue can be represented in the memory by using either the Static representation or the Dynamic representation. An array-based representation of a queue, involves declaring a one-dimensional array of some specified size. The array representation of a queue needs two indices; FRONT and REAR, which indicate the position of front and rear ends. The array declaration in queue is

int queue [MAXSIZE];

Where, queue is the array name and it has MAXSIZE elements. The FRONT and REAR helps in positioning the items whenever insertion or deletion operation is performed. To insert an element to the queue, increment the REAR index by 1 and place the element in the rear position. Similarly, taking element from the FRONT position does deletion and then the Front is incremented by one.

**OPERAtiONS ON A QUEUE**

The basic operations performed on a Queue are:

|  |  |  |
| --- | --- | --- |
| **Operation** | **Description** | **Restriction** |
| Create Queue | It creates a new empty queue. This operation must be carried out in order to make the queue logically accessible. |  |
| Qinsert | It inserts a new data item at the rear of the queue. | Queue must not be full |
| Qdelete | It deletes and then returns the data item at the front of the queue. | Queue must not be empty |
| Queue Full | It returns true if the queue is full. Otherwise, it returns false. |  |
| Queue Empty | It returns true if the queue is empty. Otherwise, it returns false. |  |

**Algorithms for operation on queue**

**Inserting an element**

Step 1: If rear is equal to Max (the Max elements to be in queue) then display as queue full and end the insert procedure.

Step 2: Failing step 1, increment the rear pointer by 1

Step 3: insert an element into the queue into the position indicated by rear,

Queue [rear] = element

Step 4:Repeat step 2 and step 3 until queue full.

**Deleting an element**

Step 1: If front pointer is equal to the rear pointer, then display as queue empty and end delete procedure.

Step 2: Failing step 1 remove an element from its queue from the pointer indicated by front pointer,

Element:=queue[front]

Step 3: Increment the front pointer by 1.

Step 4: repeat step 2 and step 3 until queue empty.

**Displaying an element**

The contents of the queue can be displayed only if queue is not empty. If queue is empty an appropriate message is displayed. The function display is shown below:

void display ( int q[], int f, int r)

{

int i;

if ( qempty(f,r) )

{

printf( “Queue is empty\n” );

return;

}

printf( “Contents of queue is \n” );

for( i=f; i<=r; i++ )

printf( “%d\n”,q[i] ); }

7a) Using linear array

Program:

#include<stdio.h>

#include<conio.h>

int queue[3],maxsize=3;

int front=-1,rear=-1;

main()

{

int ch;

clrscr();

do

{

printf("Enter the choice/n");

printf("1->Add\t2->Delete\t3->Display\t4->Exit\t");

scanf("%d",&ch);

switch(ch)

{

case 1:insert();

break;

case 2:delete();

break;

case 3:display();

break;

case 4:return;

default:printf("Wrong choice\n");

}

}

while(ch!=4);

return;

}

insert()

{

int num;

if(rear==(maxsize-1))

{

printf("Queue is full\n");

return;

}

else

{

printf("Enter the element:");

scanf("%d",&num);

rear++;

queue[rear]=num;

if(front==-1)

front=rear;

}

return;

}

delete()

{

if(front==-1)

{

printf("Queue is empty\n");

return;

}

else

{

printf("Deleted element is %d\n",queue[front]);

if(front==rear)

front=-1;

else

front++;

}

return;

}

display()

{

int i;

if(front==-1)

{

printf("Queue is empty\n");

return;

}

else

{

printf("The element are:");

for(i=front;i<=rear;i++)

printf("%d",queue[i]);

}

printf("\n");

return;

}

Output:

7b) Write a program to perform Insert, Delete and Display operations on a queue using

b)Using Pointers:

Program:

#include<stdio.h>

#include<process.h>

#define max 3

void main()

{

int ch=1,f,r,q[10];

f=0;

r=-1;

clrscr();

while(ch)

{

printf("\n1:insertrear\t 2:deletefront\t 3:diplay\t 4:exit\n");

printf("\n enter ur chioce\n");

scanf("%d",&ch);

switch(ch)

{

case 1:insert\_rear(&r,q);

break;

case 2:delete\_front(q,&f,&r);

break;

case 3:display(q,f,r);

break;

case 4:exit(0);

break;

default:printf("please enter correct chioce\n");

break;

}

}

}

insert\_rear(int \*r,int q[])

{

int item;

if(\*r==max-1)

{

printf("\n q is overflow\n");

return;

}

printf("\n enter the elements to be inserted\n");

scanf("%d",&item);

\*r=\*r+1;

q[\*r]=item;

return;

}

delete\_front(int q[],int \*f,int \*r)

{

if(\*f>\*r)

{

printf("q is underflow\n");

return;

}

printf("\n the elements deleted is %d\n",q[(\*f)++]);

if(\*f>\*r)\*f=0,\*r=-1;

return;

}

display(int q[],int f,int r)

{

int i;

if(f>r)

{

printf("\n q is empty\n");

return;

}

printf("\n contents of q is \n");

for(i=f;i<=r;i++)

printf("%d\n",q[i]);

return;

}

Output:

1. Write a C program to simulate the working of a circular queue of integers using an array. Provide the following operations
2. Insert b) Delete c) Display

ABOUT THE EXPERIMENT: Circular queue is a linear data structure. It follows FIFO principle. In circular queue the last node is connected back to the first node to make a circle. Circular linked list fallow the First In First Out principle. Elements are added at the rear end and the elements are deleted at front end of the queue. The queue is considered as a circular queue when the positions 0 and MAX-1 are adjacent. Any position before front is also after rear.

ALGORITHM:

Step 1: Start.

Step 2: Initialize queue size to MAX

Step 3: Insert the elements into circular queue. If queue is full give a message as ‘queue is overflow”

Step 4: Delete an element from the circular queue. If queue is empty give a message as ‘queue is underflow’.

Step 5: Display the contents of the queue.

Step 6: Stop.

Program:

#include<stdio.h>

#include<process.h>

#idefine queue\_size 5

void insert\_rear(int item,int q[],int \*r,int \*count)

{

if(\*count==queue\_size)

{

printf("overflow of queue\n");

return;

}

\*r=(\*r+1)%queue\_size;

q[\*r]=item;

\*count+=1;

}

void delete\_front(int q[],int \*f,int \*count)

{

if(count==0)

{

printf("underflow of queue\n");

return;

}

printf("the deleted element is %d/n",q[\*f]);

\*f=(\*f+1)%queue\_size;

\*count=1;

}

void display(int q[],int f,int count)

{

int i;

if(count==0)

{

printf("q is empty\n");

return;

}

printf("contents of queue is\n");

for(i=1;i<=count;i++)

{

printf("%d\n",q[f]);

f=(f+1)%queue\_size;

}

}

void main()

{

int choice,item,f,r,count,q[20];

f=0;

r=-1;

count=0;

clrscr();

for(;;)

{

printf("1:insert 2:delete\n");

printf("3:display 4:exit\n");

printf("enter the choice\n");

scanf("%d",&choice);

switch(choice);

{

case 1:printf("enter the item to be inserted\n");

scanf("%d",&item);

insert\_rear(item,q,&r,&count);

break;

case 2:delete\_front(q,&f,&count);

break;

case 3:display(q,f,count);

break;

default:exit(0);

}

}

}

Output:

9) Write a C program using dynamic variables and pointers, to construct a singly linked list consisting of the following information in each node: student id (interger), student name(character string) and semester(integer). The operations to be supposed are:

1. The insertion operation

i)At the front of a list

ii) At the back of the list

iii) At any position in the list

1. Deleting a node based on student id. If the specified node is not present in the list an error message should be displayed. Both the operation should be demonstrated.
2. Searching a node based on student id and up date the information content. If the specified node is not present in the list an error message should be displayed. Both situations should be displayed.
3. Displaying all the nodes in the list.

**Stack using single linked list**

A stack is a data structure in which all insertion and deletion are performed at one end called top. Insertion operation is called pushing. Deletion operation is called popping.

We can insert a new element at the front of a linked list. This operation accomplishes the newly added element as the only immediately accessible element and similarly, the deletion from the front always accomplishes the new front element for the immediate removal. In both cases insertion and deletion are performed at the front of a linked list.

Comparing these operations with the push and pop operations of stack, reveals that they are quite identical. That is, inserting a new element at the front is equivalent to pushing a new element on to the stack at the top position. And similarly, deleting an element from the beginning of a linked list is just similar to popping an element from the top of the stack.

**Algorithm for stack using single linked list to push items into the linked list**

Step 1: Allocate memory for the new node.

Step 2: Assign the value to the data field of the new node

Step 3: Make the link field of the new node to point to the starting node of the linked list.

Step 4: Then, set the external pointer, i.e, TOP (which is pointing to the starting node) to point to the new node.

**Algorithm for stack using single linked list to Pop an item from stack**

Step 1: If the list is not empty then check whether the element to be deleted belongs to the first node of the list i.e., TOP

Step 2: Move the head pointer to the second node.

Step 3: Free the first node i.e. TOP.

Program:

#include<stdio.h>

#include<alloc.h>

#include<process.h>

#include<string.h>

struct student

{

char name[20];

int id;

int sem;

struct student \*link;

};

typedef struct student \*STUDENT;

STUDENT getnode(void)

{

STUDENT X;

X=(STUDENT)malloc(sizeof(struct student));

if(X==NULL)

{

printf("out of memory\n");

exit(0);

}

return X;

}

void freenode(STUDENT X)

{

free(X);

}

STUDENT insert\_front(char name[],int id,int sem,STUDENT first)

{

STUDENT temp;

temp=getnode();

strcpy(temp->name,name);

temp->sem=sem;

temp->id=id;

temp->link=first;

first=temp;

return first;

}

STUDENT insert\_rear(char name[],int id,int sem,STUDENT first)

{

STUDENT temp;

STUDENT cur;

temp=getnode();

strcpy(temp->name,name);

temp->id=id;

temp->sem=sem;

temp->link=NULL;

if(first==NULL)return temp;

cur=first;

while(cur->link!=NULL)

{

cur=cur->link;

}

cur->link=temp;

return first;

}

STUDENT delete\_student(int id,STUDENT first)

{

STUDENT pre,cur;

if(first==NULL)

{

printf("no student in the organization\n");

return NULL;

}

pre=NULL;

cur=first;

while(cur!=NULL && id!=cur->id)

{

pre=cur;

cur=cur->link;

}

if(cur==NULL)

{

printf("student id not found\n");

return first;

}

if(pre==NULL)

first=first->link;

else

pre->link=cur->link;

free(cur);

return first;

}

void display(STUDENT first)

{

STUDENT temp;

if(first==NULL)

{

printf("no student in the organization\n");

return;

}

printf("studentname studentid studentsem\n");

printf("................\n");

for(temp=first;temp!=NULL;temp=temp->link)

printf("%10s %4d %4d\n",temp->name,temp->id,temp->sem);

printf("\n");

}

void main()

{

STUDENT first=NULL;

int choice,id,sem,pos;

char name[10];

clrscr();

for(;;)

{

printf("1:insert front 2:insert rear\n");

printf("3:delete 4:display\n");

printf("5:exit\n");

printf("enter the ur choice\n");

scanf("%d",&choice);

if(choice==1 || choice==2)

{

printf("name:");

scanf("%s",name);

printf("id:");

scanf("%d",&id);

printf("sem:");

scanf("%d",&sem);

}

switch(choice)

{

case 1:

first=insert\_front(name,id,sem,first);

break;

case 2:

first=insert\_rear(name,id,sem,first);

break;

case 3:

printf("delete student details for id:");

scanf("%d",&id);

first=delete\_student(id,first);

break;

case 4:

display(first);

break;

default:

exit(0);

}

}

}

Output:

11) Write a C program using dynamic variables and pointers to construct a stack of integers using singly linked list and to perform the following operations.

1. PUSH
2. POP
3. Display

The program should print appropriate messages for stack overflow, and stack empty.

Program:

#include<stdio.h>

#include<alloc.h>

#include<process.h>

struct node

{

int info;

struct node \*link;

};

typedef struct node \*NODE;

NODE getnode()

{

NODE X;

X=(NODE)malloc(sizeof(struct node));

if(X==NULL)

{

printf("out of memory\n");

exit(0);

}

return X;

}

void freenode(NODE X)

{

free(X);

}

NODE insert\_front(int item,NODE first)

{

NODE temp;

temp=getnode();

temp->info=item;

temp->link=first;

return temp;

}

void display(NODE first)

{

NODE temp;

if(first==NULL)

{

printf("list is empty\n");

return;

}

printf("the contents of singly linked list\n");

temp=first;

while(temp!=NULL)

{

printf("%d",temp->info);

temp=temp->link;

}

printf("\n");

}

NODE delete\_front(NODE first)

{

NODE temp;

if(first==NULL)

{

printf("list is empty can not delete\n");

return first;

}

temp=first;

first=first->link;

printf("item deleted=%d\n",temp->info);

freenode(temp);

return first;

}

void main()

{

NODE first=NULL;

int choice,item;

for(;;)

{

printf("1:insert front 2:delete front\n");

printf("3:display 4:exit\n");

printf("enter the choice\n");

scanf("%d",&choice);

switch(choice)

{

case 1:

printf("enter the item to be inserted\n");

scanf("%d",&item);

first=insert\_front(item,first);

break;

case 2:

first=delete\_front(first);

break;

case 3:

display(first);

break;

default:

exit(0);

}

}

}

Output:

12) Write a C program using dynamic variables and pointers to construct a queue of integers using singly linked list and to perform the following operations.

a)Insert

b)Delete

c)Display

The program should print appropriate messages for queue full and queue empty.

**Queue using Linked List:**

A queue is a data structure in which all insertions are performed at one end called the rear and all deletions are performed at the other end called the front.

When the queue is implemented as an array, it suffers from limitation of its size, i.e size of a queue cannot be increased or decreased, once it is declared. This problem can be overcome by implementing the queue using linked list. And such a queue is called linked queue.

In a linked queue, the insertion of a new node is done at the end of a linked list. And, the deletion operation is performed from the beginning of a linked list Front is an external pointer pointing to the very first node in the linked queue. It indicates the position of the deletion. Similarly, rear is a pointer to the last node in the linked queue and the new node is added to the linked queue after this position.

**Algorithm for queue using single linked list to insert item to linked list:**

Step 1: If the list is empty, then create the new node.

Step 2: If the list is not empty then go to the last node and then insert the new node after the last node.

Step 3: Make the link field of the rear node to point to the new node and call it as rear node.

Step 4: Make the link field of the rear node as null.

**Algorithm for queue using single linked list to Delete item from linked list:**

Step 1: If the list is not empty, (front is not null) then check whether the element to be deleted belongs to the first node of the list.

Step 2: Move the head pointer to the second node.

Step 3: Free the first node( ie front).

Program:

#include<stdio.h>

#include<stdlib.h>

#include<process.h>

struct node

{

int info;

struct node \*link;

};

typedef struct node \*NODE;

NODE getnode()

{

NODE X;

X=(NODE)malloc(sizeof(struct node));

if(X==NULL)

{

printf("out of memory\n");

exit(0);

}

return X;

}

void freenode(NODE X)

{

free(X);

}

void display(NODE first)

{

NODE temp;

if(first==NULL)

{

printf("list is empty\n");

return;

}

printf("the contents of singly linked list\n");

temp=first;

while(temp!=NULL)

{

printf("%d",temp->info);

temp=temp->link;

}

printf("\n");

}

NODE delete\_front(NODE first)

{

NODE temp;

if(first==NULL)

{

printf("list is empty can not delete\n");

return first;

}

temp=first;

first=first->link;

printf("item delete=%d\n",temp->info);

freenode(temp);

return first;

}

NODE insert\_rear(int item,NODE first)

{

NODE temp;

NODE cur;

temp=getnode();

temp->info=item;

temp->link=NULL;

if(first==NULL)return temp;

cur=first;

while(cur->link!=NULL)

{

cur=cur->link;

}

cur->link=temp;

return first;

}

void main()

{

NODE first=NULL;

int ch,item;

clrscr();

for(;;)

{

printf("1:insert rear\t 2:delete front\t 3:display\t 4:exit\n");

printf("enter ur choice");

scanf("%d",&ch);

switch(ch)

{

case 1:

printf("enter the item to be inserted\n");

scanf("%d",&item);

first=insert\_rear(item,first);

break;

case 2:

first=delete\_front(first);

break;

case 3:

display(first);

break;

default:

exit(0);

}

}

}

Output:

13) Write a C program

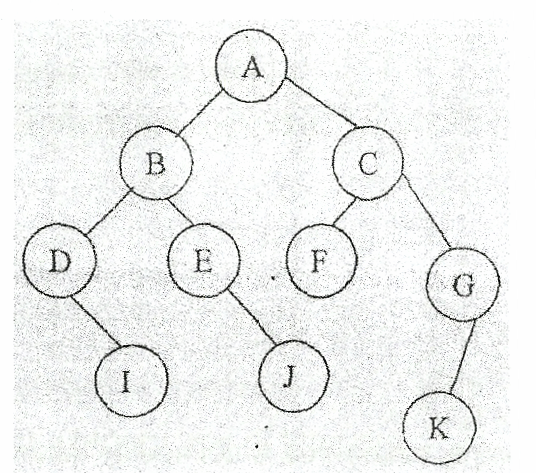
1. To construct a binary search three of integers.
2. To traverse the tree using all the methods i.e, inorder, preorder and postorder.

**Binary Trees**

**Definition Binary Search Tree:**

A binary tree is a finite set of data items which is either empty or consist of a single item called the root and two disjoint binary trees called the left subtree and the right subtree.

A binary is a very important and most commonly used non-linear data structure. In a binary tree the maximum degree of any node is at most two. That means there may be a zero degree node or a one-degree node and two degree node.



In the above binary tree, A is the root of a tree. The left subtree consists of the tree with root B. And the right subtree consists of the tree with root C. Further B has its left subtree with root D and right subtree with root E. Similarly, C has its left subtree with root F and its right subtree with root G. In the next level D has an empty left subtree and its right subtree with root I. E has also an empty left subtree and contains its right sub-tree with root J. F has neither left subtree nor right subtree. G has its left subtree K as its root and has a right sub tree.

**Traversal of a binary tree:**

There are 3 popular ways of a binary tree traversal. They are

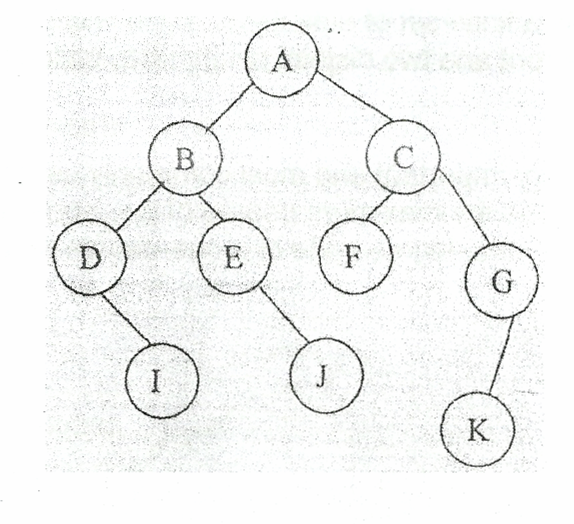
1. Preorder traversal

2. Inorder traversal

3. Postorder traversal

**Preorder traversal**

The preorder traversal of a non-empty binary tree is defined as follows:



1. Visit the root node

2. Tavares the left subtree in the preorder

3. Taverse the right subtree in the preorder

ROOT(N) A

LEFT(L) BDIEJ

RIGHT(R)- CFGK

**Algorithm to traverse the tree using preorder:**

Step 1: Display the data of the root node.

Step 2: Update root node to hold the left link of the root node.

Step 3: Call preorder procedure recursively with new link of the root node.

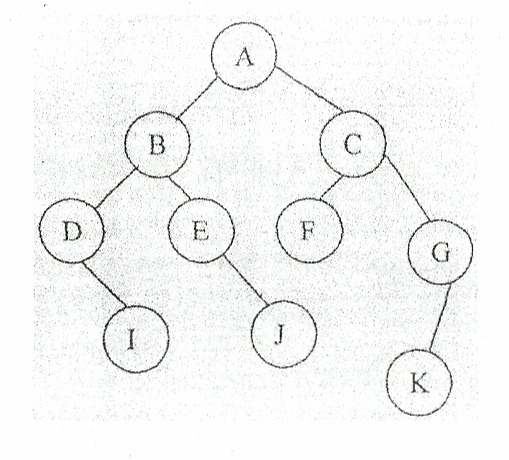
Step 4: When whole left sub tress is traversed, then update root node to hold the RIGHT link of the root node.

Step 5: Call preorder procedure recursively with new link of the root node.

Step 6: Repeat step 1 to 5 until root node points to null.

**Inorder traversal:**

The inorder traversal of non-empty binary tree is defined as follows:



1. Traverse the left subtree in the inorder

2. Visit the root node

3. Traverse the right subtree in the inorder

LEFT--------- D I B E J

ROOT--------- A

RIGHT--------- F C K G

**Algorithm to traverse the tree using inorder:**

Step 1: Update the root node to hold the LEFT link of the root node.

Step2: Call in order procedure recursively until no LEFT link le traverse whole left sub trees.

Step 3: Display the data field of the root node.

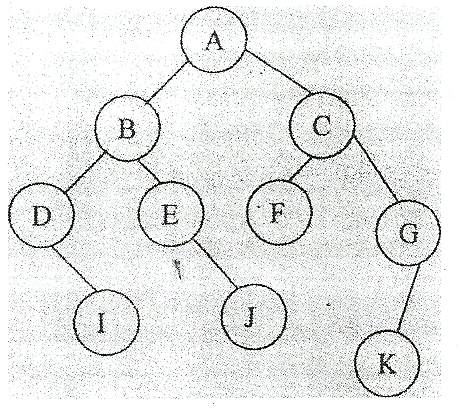
Step 4: Update the root node to hold RIGHT link of the root node.

Step 5: Call in order procedure recursively.

Step 6: Repeat the steps until root node points to null.

**Postorder traversal:**

The postorder traversal of non-empty binary tree is defined as follows:



1. Traverse the left subtree in the postorder

2. Traverse the right subtree in the postorder

3. Visit the Root

LEFT IDJEB

RIGHT FKGC

ROOT—-A

**Algorithm to traverse the tree using postorder:**

Step 1: Update the root node to hold the LEFT link of the root node.

Step 2: Call post order procedure recursively with new link.

Step 3: When no left link, Update the root node to hold RIGHT link of the root Node.

Step 4: Call post order procedure recursively with new link.

Step 5: Display the data of the root node

Step 6: Repeat the steps until root node points to null.

Program:

#include<stdio.h>

#include<conio.h>

#include<alloc.h>

#include<process.h>

struct node1

{

int data;

struct node1 \*lt,\*rt;

};

typedef struct node1 node;

node \*root=NULL;

node \*insert(node \*,int);

void inorder(node \*);

void preorder(node \*);

void postorder(node \*);

int ch,num;

void main()

{

clrscr();

while(ch!=5)

{

printf("\n\*\*\*\*\*MENUOPERATION\*\*\*\*\*");

printf("\1\_INSERT\n");

printf("2\_INORDER\n");

printf("3\_POSTORDER\n");

printf("4\_PREORDER\n");

printf("5\_exit\n");

printf("enter ur choice:\n");

scanf("%d",&ch);

switch(ch)

{

case 1:printf("enter the elements to be inserted\n");

scanf("%d",&num);

root=insert(root,num);

break;

case 2:printf("elements in INORDER\n");

inorder(root);

break;

case 3:printf("elements in POSTORDER\n");

postorder(root);

break;

case 4:printf("elements in PREORDER\n");

preorder(root);

break;

case 5:exit(0);

default:printf("enter the ur choice\n");

}

}

getch();

}

node \*insert(node \*p,int num)

{

if(p==NULL)

{

p=(node\*)malloc(sizeof(node));

p->data=num;

p->rt=NULL;

p->lt=NULL;

}

else

{

if(num<p->data)

p->lt=insert(p->lt,num);

else

p->rt=insert(p->rt,num);

}

return(p);

}

void preorder(node \*p)

{

if(p!=NULL)

{

printf("%d\t",p->data);

preorder(p->lt);

preorder(p->rt);

}

}

void inorder(node \*p)

{

if(p!=NULL)

{

inorder(p->lt);

printf("%d\t",p->data);

inorder(p->rt);

}

}

void postorder(node \*p)

{

if(p!=NULL)

{

postorder(p->lt);

postorder(p->rt);

printf("%d\t",p->data);

}

}

Output:

14) Write a C program to evaluate an expression three.

Program:

#include<stdio.h>

#include<ctype.h>

#include<alloc.h>

#include<math.h>

#include<string.h>

#define STACK\_SIZE 20

struct node

{

char info;

struct node \*llink;

struct node \*rlink;

};

typedef struct node \*NODE;

NODE getnode()

{

NODE x;

x=(NODE)malloc(sizeof(struct node));

if(x==NULL)

{

printf("out of memory\n");

exit(0);

}

return x;

}

NODE creat\_tree(char postfix[1])

{

NODE temp,st[20];

int i,k;

char symbol;

for(i=k=0;(symbol=postfix[i])!='\0';i++)

{

temp=getnode();

temp->info=symbol;

temp->llink=temp->rlink=NULL;

if(isalnum(symbol))

st[k++]=temp;

else

{

temp->rlink=st[--k];

temp->llink=st[--k];

st[k++]=temp;

}

}

return st[--k];

}

float eval(NODE root)

{

float num;

switch(root->info)

{

case'+':return eval(root->llink)+eval(root->rlink);

case'-':return eval(root->llink)-eval(root->rlink);

case'/':return eval(root->llink)/eval(root->rlink);

case'\*':return eval(root->llink)\*eval(root->rlink);

case'$':

case'^':return pow(eval(root->llink),eval(root->rlink));

default:

if(isalpha(root->info))

{

printf("%c=",root->info);

scanf("%f",&num);

return num;

}

else

return root->info-'0';

}

}

void main()

{

char postfix[40];

float res;

NODE root=NULL;

printf("enter the postfix expression\n");

scanf("%s",postfix);

root=creat\_tree(postfix);

res=eval(root);

printf("result=%f\n",res);

getch();

}

Output:

1)Enter the postfix expression 2) Enter the postfix expression

ab+ ab\*

a=2 a=4

b=4 b=6

Result = 6.00 result= 24.00