Introduction to Data Structure(CS215)
Lab Exercises Programs

Siddhant Sanyam (siddhant3s **at** gmail **dot** com)

November 12, 2010

Preface

This document contains the solution to the lab exercises for the course 'Introduction to Data Structure with course code CS215 at the National Institute of Technology, Tiruchirapalli under the faculty Mr. R. Mohan. The problems include implementation of different type of important data structures (and few related algorithms). All these twelve problem have been solved and tested by the me.

Solving these problem was a great learning opportunity to practice the implementation detail of various data structures including arrays, sparse matrix, linked lists, trees etc. They not only improved the quality of code writing but also helped in learning the theory with a deeper understanding.

I started writing the code as the part of this exercise. Later it was realized, that writing the code properly would serve two advantage: primarily, it would help me understand the concept better. Secondarily, a standard and working code for these problems could be made available in the public domain. Since I version control all my code using Git version controlling system, I made an online repository of this code so that it could be improvised further by open source programmers around the world. Soon, few developers joined in to improve the code and this was taken up as a project by me. The aim was to write an ideal piece of code, which could be suited for reference and learning purpose. This aim has been partially accomplished. Although I am submitting this code, it should be pointed out that there are still a lot of improvement to be done to make it a reference-level code.

As a printed form of the code was required, the same has been attached. Although, it is highly recommended that the online version should be referred for any and every need. The online version is maintained and updated thoroughly. This code was written in Linux environment with `gcc` and `g++` as the compiler (version 4.5). Care has been taken to make the code compliant to the C and C++ standards. Hence the code should compile on any modern compiler following the standards.

Online Viewing

The source code is available online at the the following URL http://github.com/siddhant3s/cs215. It is a Git repository. Hence it is version controlled and duly maintained by me.

Current Limitations

The few part of source code might lack proper documentation. This is continuously been added online. As the different part of the code was written at different point in time, there might be a lack in uniformity of style. References made from various books and Internet have not been added currently. To keep the code short and helpful for beginners to grasp at an educational level, error checking codes have been skipped at few trivial places. Thus this code is not suited for any production level use.

Licensing

Bugs, Errors, Suggestions, Patches

Since no code is perfect, any bugs, suggestions or patches might be submitted on the Issue tracker page https://github.com/siddhant3s/cs215/issues.

Contents

1 Program

Suppose that an ordered list of numbers is implemented by means of an array, Write a C program using separate functions to

1. Insert a number
2. Delete an element
3. Reverse the list

Write the main program where an initial list is created and a user is prompted to ask which operation (insert, delete, reverse or quit) the user wants to perform, Depending on the user response, more inputs may be taken and the resultant list is printed after each operation by the user. The main program continues unless the user chooses "quit´´ operation.

```
1  #include <stdio.h>
2
3  void array_print(int *a, size_t size)
4  {
5    while(size--)
6      printf("%i ",*a++);
7  }
8
9  int array_insert(int *a, size_t size, int element, size_t position)
10 /* Inserts the 'element' to the array 'a'.
11 The'size' denotes the number of current elements in the array.
12 The 'element' is inserted at a[position].
13 The 'size' should not be the size of the array allocated.
14 The function will check if the position specified is within the size specified.
15 It returns a positive integer if the insertion was successful else returns a fal
16 with the error printed on stderr*/
17 {
18   if (position >=size)
19     {
20       fprintf(stderr,"\narray_insert: The specified position %i is not less than
21               " size of the array which is %i\n",position,size);
22       return 0;
23     }
24   int *p=a+position;//the pointer to the location where the new element will get
25   a+=size;//a points to one past the last element of original array
26   while(a>p)     *(a)=*(a-1),a--;
27   *a=element;
28   return 1;
29 }
30
31 int array_delete(int *a, size_t size, size_t position)
```

```
32 /*Removes the element a[position] from the array. The 'size' denotes the
   number of
33   elements in the array before the removal. Returned zero if the deletion failed
34 else returns a positive integer.*/
35 {
36   if (position >=size)
37     {
38       fprintf(stderr,"\narray_delete: The specified position %i is not less than
39               " size of the array which is %i\n",position,size);
40       return 0;
41     }
42   int *last=a+size-1;// points to the last element
43   a+=position;//a now points to the position of the deltetion
44   while(a<last) *(a)=*(a+1),a++;
45   return 1;
46 }
47
48 void array_reverse(int *a, size_t size)
49 /* Reverses the array 'a'. The 'size' represents the number of elements in 'a'*/
50 {
51   int i,t;
52   for(i=0; i < size/2 ; i++)
53     {
54       t=a[i]; a[i]=a[size-i-1]; a[size-i-1]=t;
55     }
56 }
57
58 void input_array(int *a, size_t size)
59 /*Inputs from cin*/
60 {
61   while(size--)
62     scanf("%i",a++);
63 }
64
65 int main()
66 {
67   const MAX = 20;
68   int a[MAX];
69
70   printf("With How many numbers do you wanto start with?");
71   size_t s;
72   scanf("%i", &s);
73   printf("Enter the array:");
74   input_array(a,s);
75   char i;
76   do{
```

```
77      printf("\nWhat␣do␣you␣want␣to␣do?"
78             "\n␣1.␣Insert␣an␣element.\n␣2.␣Delete␣an␣element␣\n␣3.␣Reverse"
79             "\n␣4.␣Display␣\n␣5.␣Exit\n");
80
81      scanf("%c", &i);
82      switch(i)
83        {
84        case '1':
85          {
86            printf("Enter␣the␣Position␣(zero-based)␣and␣the␣element␣to␣be␣inserted
87            int e;
88            size_t p;
89            scanf("%i%i",&p,&e);
90            array_insert(a,s,e,p);
91            s++;
92            printf("Done!\n");
93            break;
94          }
95        case '2':
96          {
97            printf("Enter␣the␣position␣at␣which␣you␣want␣to␣delete␣the␣element:");
98            size_t p;
99            scanf("%i",&p);
100           array_delete(a,s,p);
101           s--;
102           printf("Done!\n");
103           break;
104         }
105       case '3':
106         {
107           array_reverse(a,s);
108           printf("Done!\n");
109           break;
110         }
111       case '4':
112         {
113           printf("\n");
114           array_print(a,s);
115           printf("\n\n");
116           break;
117         }
118       }
119   }while(i!='5');
120     /*
121
122   //printf("Size:%i",s);
```

```
123     array_print(a,s);
124     array_insert(a,5,7,3);
125     printf("\n");array_print(a,s);
126     array_delete(a,6,2);
127     printf("\n");array_print(a,s);
128     array_reverse(a,5);
129     printf("\n");array_print(a,s);
130       */
131     return 0;
132 }
```

2 Program

Write a program for string manipulation

1. To find the string length
2. Reverse the string
3. To check if the string is palindrome
4. String concatenation
5. Extracting a sub string from a given string
6. To convert the string into uppercase and lowercase

```
1  #include<stdio.h>
2  char* strcat(char* dest, char* src)
3  {
4       while (*dest++);
5       dest--;
6       while (*dest++ = *src++);
7       return dest;
8  }
9  char* strncat(char* dest, char* src, int n)
10 {
11      while (*dest++);
12      dest--;
13      while ((--n>=0) && (*dest++ = *src++));
14      *dest='\0';
15      return dest;
16 }
17
18 int strcmp(char *s, char *t)
19 {
20      for ( ; *s == *t; s++, t++)
```

```
21          if (*s == '\0')
22              return 0;
23      return *s - *t;
24  }
25  char toupper(char c)
26  {
27      if (c>='a' && c<='z') return 'A'+(c-'a');
28      else return c;
29  }
30  int stricmp(char *s, char *t)
31  {
32      for ( ; toupper(*s) == toupper(*t); s++, t++)
33          if (*s == '\0')
34              return 0;
35      return toupper(*s) - toupper(*t);
36  }
37
38
39  void strcpy(char *s, char *t)
40  {
41      while (*s++ = *t++);
42  }
43  void strncpy(char *s, char *t,int n)
44  {
45      while ( --n>=0 && (*s++ = *t++));
46  }
47  size_t strlen(char *s)
48  {
49      char* t=s;
50      while (*t++);
51      t--;
52      return t-s;
53  }
54  void strrev(char* s)
55  {
56      size_t l=-1, i=0;
57      while (s[++l]);
58      for (i=0;i<l/2;++i)
59      {
60          char t=s[i];
61          s[i]=s[l-i-1];
62          s[l-i-1]=t;
63      }
64  }
65  int main(void)
66  {
```

```
67      const size_t MAXLEN=50;
68      char s1[50]="The First String";
69      char s2[]="Second String";
70      printf("\t s1=%s \n\t s2=%s \n",s1,s2);
71
72
73
74      printf("Concatanating s1 and s2 to s1. Now,\n");
75      strcat(s1,s2);
76      printf("\t s1=%s \n\t s2=%s \n",s1,s2);
77
78
79      printf("Reversing s2:\n");
80      strrev(s2);
81      printf("\t s1=%s \n\t s2=%s \n",s1,s2);
82
83      printf("Copying s2 to s1:\n");
84      strcpy(s1,s2);
85      printf("\t s1=%s \n\t s2=%s \n",s1,s2);
86
87
88      printf("Concatenating first 5 character from s2 to s1:\n");
89      strncat(s1,s2,5);
90      printf("\t s1=%s \n\t s2=%s \n",s1,s2);
91
92
93      printf("Reversing s1:\n");
94      strrev(s1);
95      printf("\t s1=%s \n\t s2=%s \n",s1,s2);
96
97      printf("The length of s1=%i\n",strlen(s1));
98
99      strcpy(s1,"ME");
100     strcpy(s2,"me");
101
102     printf("strcmp over s1 and s2=%i",strcmp(s1,s2));
103     printf("stricmp over s1 and s2=%i",stricmp(s1,s2));
104
105     return 0;
106 }
```

3 Program

Write a program for

1. Adding two polynomials where a polynomial is implemented by an array of records.

2. Multiplying two polynomials where a polynomial is implemented by an array of records.

```
1  #include <stdio.h>
2
3  /*The Polynomial is represented by an array of its coefficients on the respectiv
4  Position of the exponent:
5  Say we have a Polynomial P(x): 14x^3 + 62x^2 + 41x + 79
6  It should be represented in the array as
7          +----+----+----+----+
8          |  0 |  1 |  2 |  3 | <----Array Subscript
9          |--------------------
10         | 79 | 41 | 62 | 14 | <----Coefficients
11         +--------------------+
12 */
13
14 void add_poly(int *a, size_t dega, int *b, size_t degb, int *c)
15 {
16   size_t i, j;
17   for (i=0; i<=dega; i++) c[i] = a[i];
18   for (j=i; j<=degb; j++) c[j] = b[j];
19   for (j=0; j<i && j<=degb; j++) c[j] += b[j];
20 }
21
22 void mul_poly(int *a, size_t dega, int *b, size_t degb, int *c)
23 {
24   size_t i;
25   for(i=0; i<=(dega+degb) ; i++)
26     c[i]=0;
27
28   for(i=0; i <= dega; i++)
29     {
30       size_t j;
31       for(j=0; j<= degb; j++)
32         {
33           c[i+j]+=a[i]*a[j];
34         }
35     }
36 }
37
38 void print_pol(int *a, size_t deg)
39 {
40   for (;; deg--)
```

```
41    {
42      if (!a[deg])              /* coefficient is zero,    */
43        if (deg) continue;      /* nothing to be printed   */
44        else break;
45
46      /* print coefficient */
47      if (deg != 0 && (a[deg] == -1 || a[deg] == 1))
48        printf("%c", (a[deg] < 0 ? '-':'+'));
49      else
50        printf("%+d", a[deg]);
51
52      if (deg != 0)   /* print variable? */
53      {
54        printf("X");
55
56        if (deg != 1) /* print exponent? */
57          printf("^%d", deg);
58      }
59      else     /* last term of polynomial */
60        break;
61    }
62
63    printf("\n");
64  }
65
66
67  int main(void)
68  {
69    int pol1[]={1,1,2};
70    size_t deg1=((sizeof pol1)/sizeof(*pol1))-1;
71
72    int pol2[]={1,1};
73    size_t deg2=((sizeof pol2)/sizeof(*pol2))-1;
74    int pol3[7];
75    add_poly(pol1,deg1,pol2,deg2,pol3);
76    print_pol(pol3,2);
77
78    mul_poly(pol1,deg1,pol2,deg2,pol3);
79    print_pol(pol3,deg1+deg2);
80    return 0;
81  }
```

4 Program

Write a program for adding two sparse matrices and transposing a sparse matrix where a sparse matrix is implemented by an array of records.

```
1  #include <stdio.h>
2
3  struct SparseNode
4  {
5    int data;
6    size_t x;
7    size_t y;
8  };
9
10 /*
11 A sparse matrix is represented as an array with all the Non-zero listed. Each el
12 the array is basically a SparseNode which contains the actual data and the posit
13 element as x and y.
14
15 */
16 struct SparseNode* sparse_node_at_xy(struct SparseNode A[], size_t size, size_t
17 {
18   size_t i;
19   for (i=0; i < size; i++)
20     if( A[i].x==x && A[i].y==y)
21       return A+i;
22   return NULL;
23 }
24 size_t sparse_add(struct SparseNode A[], size_t sizeA, struct SparseNode B[], si
25 {
26   /* result[] is assumed to have a length at least sizeA+sizeB */
27   size_t i;
28   for (i=0; i< sizeA; i++)
29     result[i]=A[i];
30   size_t j, sizeC=sizeA;
31   for (j=0; j < sizeB; j++)
32     {
33       struct SparseNode* currentnode= sparse_node_at_xy(result, sizeA, B[j].x, B
34       if (currentnode)
35         currentnode->data += B[j].data;
36       else
37         {
38           result[i+j]=B[j];
39           sizeC++;
40         }
41     }
```

```
42    return sizeC;
43  }
44  void sparse_transpose(struct SparseNode A[], size_t size)
45  {
46    while(size--)
47      {
48        size_t t= A[size].x;
49        A[size].x=A[size].y;
50        A[size].y=t;
51      }
52  }
53  size_t sparse_input(struct SparseNode A[], size_t max)
54  {
55
56    size_t size;
57    printf("Enter␣the␣number␣of␣non-zero␣elements␣(Maximum␣non␣zero␣elements␣must␣
58    scanf("%u",&size);
59    //  printf("You entered a size %u",size);
60    if(size>max)
61      {
62        fprintf(stderr,"ERROR.␣The␣number␣of␣non-zero␣element␣exceeded␣%u.␣Can't␣i
63        return 0;
64      }
65
66    size_t i;
67    for(i=0; i < size; i++)
68      {
69        size_t x,y;
70        int data;
71        printf("Enter␣the␣Data:");
72        scanf("␣%i", &data);
73        //  printf("You entered %i\n",data);
74        printf("Enter␣its␣position␣in␣Matrix␣as␣x␣and␣y␣(zero-based)␣separated␣by␣
75        scanf("%u%u",&x,&y);
76        struct SparseNode node={data,x,y};
77        A[i]=node;
78      }
79    return i;
80  }
81  void sparse_output(struct SparseNode A[],size_t size, size_t x, size_t y)
82  {
83    size_t i;
84    for(i=0; i<x; i++)
85      {
86        size_t j;
87        for( j=0; j<y ;j++)
```

```
88          {
89            struct SparseNode *node=sparse_node_at_xy(A,x*y,i,j);
90            printf("%i\t",  (node!=NULL)? node->data : 0);
91          }
92        printf("\n");
93      }
94  }
95  void sparse_dump(struct SparseNode A[], size_t size)
96  {
97    size_t i;
98    for(i=0; i<size; i++)
99      printf("%u[%i,%u,%u]--",i,A[i].data,A[i].x,A[i].y);
100 }
101 int main()
102 {
103   const MAX = 50;
104   struct SparseNode A[MAX],B[MAX],C[2*MAX];
105   size_t sizeA=0,sizeB=0,sizeC=0;
106   printf("This program implements Addition and transposition of Sparse Matrix\n"
107   do
108     {
109       printf("Please choose one of the following operations:\n"
110              "1. Enter the Sparse Matrix A (Maximum non zero elements must not e
111              "2. Enter the Sparse Matrix B (Maximum non zero elements must not e
112              "3. Add Sparse Matrix A and B to C\n"
113              "4. Transpose Matrix A, B or C\n"
114              "5. Display Matrix A, B and C as Two dimensional figure\n"
115              "6. Exit\n"
116              "Enter an option:", MAX, MAX);
117       char ch;
118       ch=getchar();
119       switch(ch)
120         {
121         case '1':
122           sizeA=sparse_input(A,MAX);
123           break;
124         case '2':
125           sizeB=sparse_input(B,MAX);
126           break;
127         case '3':
128           sizeC=sparse_add(A,sizeA, B,sizeB, C);
129           printf("Added Matrix A and B to C");
130           break;
131         case '4':
132           printf("Which Sparse Martix do you want to transpose? Enter A, B or C
133           scanf(" %c",&ch);
```

```
134             switch(ch)
135               {
136               case 'A':  sparse_transpose(A,sizeA); break;
137               case 'B':  sparse_transpose(B,sizeB); break;
138               case 'C':  sparse_transpose(C,sizeC); break;
139               default : printf("There is no such Matrix as %c. Matrix Names are ca
140               }
141             break;
142           case '5':
143             {
144               size_t xassume,yassume;
145               printf("Enter the Dimension of matrix which needs to be assumed whil
146               scanf("%u%u", &xassume, &yassume);
147
148               printf("\nPrinting A:\n");
149               sparse_output(A,sizeA,xassume, yassume);
150               //sparse_dump(A,sizeA);
151               printf("\nPrinting B:\n");
152               sparse_output(B,sizeB,xassume, yassume);
153               //sparse_dump(B,sizeB);
154               printf("\nPrinting C:\n");
155               sparse_output(C,sizeC,xassume, yassume);
156               //sparse_dump(C,sizeC);
157             }
158             break;
159           case '6':
160             return 0;
161           default :
162             printf("Wrong Choice! Try Again\n");
163           }
164       }while(1);
165     return 0;
166 }
```

5 Program

Repeat exercise 1 for ordered list which are implemented by single connected linked list

1. Singly connected Linked list.
2. Doubly connected Linked list.
3. Circular Linked list.

```
1  #include <iostream>
2  using namespace std;
3  class linklist
4  {
5       private:
6
7                 struct node
8            {
9                   int data;
10                 node *link;
11           }*p;
12
13      public:
14
15           linklist();
16           void append( int num );
17           void add_as_first( int num );
18           int addafter( int c, int num );
19           int del( int num );
20           void display();
21           int count();
22           ~linklist();
23 };
24
25 linklist::linklist()
26 {
27      p=NULL;
28 }
29
30 void linklist::append(int num)
31 {
32      node *q,*t;
33
34     if( p == NULL )
35     {
36          p = new node;
37        p->data = num;
38        p->link = NULL;
39     }
40     else
41     {
42          q = p;
43        while( q->link != NULL )
44             q = q->link;
45
46        t = new node;
```

```
47          t->data = num;
48          t->link = NULL;
49          q->link = t;
50      }
51  }
52
53  void linklist::add_as_first(int num)
54  {
55        node *q;
56
57      q = new node;
58      q->data = num;
59      q->link = p;
60      p = q;
61  }
62
63  int linklist::addafter( int c, int num)
64  {
65        node *q,*t;
66      int i;
67      for(i=0,q=p;i<c;i++)
68      {
69            q = q->link;
70          if( q == NULL )
71          {
72                cout<<"\nThere are less than "<<c<<" elements.";
73             return 0;
74          }
75      }
76
77      t = new node;
78      t->data = num;
79      t->link = q->link;
80      q->link = t;
81      return 1;
82  }
83
84  int linklist::del( int num )
85  {
86        node *q,*r;
87      q = p;
88      if( q->data == num )
89      {
90            p = q->link;
91          delete q;
92          return 1;
```

```
93      }
94
95      r = q;
96      while( q!=NULL )
97      {
98            if( q->data == num )
99          {
100               r->link = q->link;
101              delete q;
102              return 1;
103         }
104
105         r = q;
106         q = q->link;
107     }
108     cout<<"\nElement "<<num<<" not Found.";
109     return 0;
110 }
111
112 void linklist::display()
113 {
114       node *q;
115     cout<<endl;
116
117     for( q = p ; q != NULL ; q = q->link )
118           cout<<endl<<q->data;
119
120 }
121
122 int linklist::count()
123 {
124       node *q;
125     int c=0;
126     for( q=p ; q != NULL ; q = q->link )
127           c++;
128
129     return c;
130 }
131
132 linklist::~linklist()
133 {
134       node *q;
135     if( p == NULL )
136           return;
137
138     while( p != NULL )
```

```
139     {
140           q = p->link;
141        delete p;
142        p = q;
143     }
144  }
145
146  int main()
147  {   char ch;
148     linklist ll;
149  do{
150
151
152     //cout<<"No. of elements = "<<ll.count();
153     cout <<"\n\tMENU"
154          <<"\n1. Create a Linked List"
155          <<"\n2. Add Element At Last"
156          <<"\n3. Add Element At First"
157          <<"\n4. Add Element in  Between"
158          <<"\n5. Delete a Element"
159          <<"\n6. Display the List"
160          <<"\nPress 0(zero) to exit";
161          cin>>ch;
162          switch(ch)
163          {
164          case '1':        int N,V;
165                           cout<<"\nEnter the the Number of Elements you want to st
166                           cin>>N;
167                           cout<<"\nNow Enter the Elements(Press Enter after Each e
168                           for(int i=0;i<N;i++)
169                              {     cin>>V;
170                                    ll.append(V);
171                              }
172                           cout<<"\nThe following List has been Created:\n";
173                           ll.display();
174                           break;
175          case '2':
176                           cout<<"\nEnter the Element to be added at Last:";
177                           cin>>V;
178                           ll.append(V);
179                           cout<<"\nThe Element has been Added at Last";
180                           break;
181          case '3':
182                           cout<<"\nEnter the Element to be added at First:";
183                           cin>>V;
184                           ll.add_as_first(V);
```

```
185                             cout<<"\nThe Element has been Added at First";
186                             break;
187             case '4':       cout<<"\nEnter a Position after which you want to add(1
188                             cin>>N;
189                             cout<<"\nEnter the Element:";
190                             cin>>V;
191                             if(ll.addafter(N-1,V))
192                             cout<<"\nElement is been added after "<<N<<"th Position"
193                             break;
194             case '5':
195                             cout<<"\nEnter the Element to be Deleted:";
196                             cin>>V;
197                             if(ll.del(V))
198                             cout<<"Deleted";
199                             break;
200             case '6':       cout<<"\nDisplaying Linked List:\n";
201                             ll.display();
202                             break;
203             case '0':       return 0;
204         }
205         cout<<"\nDo You want to Continue?(Y/N):";
206         cin>>ch;
207         }while(ch=='y'||ch=='Y');
208         return 0;
209 }
```

6 Program

Write a program using separate functions to implement the following operations for stack data structure

1. Insert a number
2. Delete a element
3. Display the list

```
#include<iostream>
#include<string>
const int MAX_SIZE=20;

class Stack
{
  int data[MAX_SIZE];
  unsigned int top;
 public:
```

```
  void push(int);
  int pop();
  int peek();
  int isfull();
  int isempty();
  void display();
  Stack()
  {
    top=0;
  }
};

int Stack::isfull()
{
  if (top<MAX_SIZE) return 0;
  return 1;
}
int Stack::isempty()
{
  if (top==0) return 1;
  return 0;
}
void Stack::push(int d)
{
  if (!isfull())
    data[top++]=d;
  else{
    std::cout<<"Stack is FULL\n";
  }
}
int Stack::pop()
{
  if(!isempty())
    return data[--top];
  else{
    std::cout<<"Stack is Empty\n";
    return -1;
  }
}
void Stack::display()
{
  for(int i=0;i<top;++i)
    std::cout<<data[i]<< ' ';
}
int Stack::peek()
{
```

```
  if(!isempty())
    return data[top-1];
  else
    std::cout<<"Stack is Empty\n";
}

int main()
{
  Stack s;
  do
    {
      std::cout<<"Static Stack Implementation. Maximum stack size is "<<MAX_SIZE
      std::cout<<                                          \
        "Menu\n"
        "1.Push\n2.Pop\n3.Peek\n4.Display\n5.Exit";
      int opt;
      std::cin>>opt;
      switch(opt)
        {
        case 1:
          std::cout<<"Enter an element to push: ";
          int e;
          std::cin>>e;
          s.push(e);
          break;
        case 2:
          std::cout<<"The poped element is:"<<s.pop();break;
        case 3:
          std::cout<<"The Peeked elementis:"<<s.peek();break;
        case 4:
          std::cout<<std::endl;
          s.display();
          break;
        case 5: return 0;
        default:
          std::cout<<"Enter a valid input\n";
        }
    }while(1);
}
```

7 Program

Write a program using separate functions to implement the following operations for Queue data structure

1. Insert a number
2. Delete an element
3. Display the list

```
#include <iostream>
using namespace std;
int queue[10];
void add(int &f,int &r)
{
    if (r==-1 && f==-1)
    {
        f=r=0;
        cout<<"\nEnter the element:";
        cin>>queue[r];
    }
    else
    {
        if (r==9)
            cout<<"\nQueue is full!";
        else
        {
            r++;
            cout<<"\nEnter the element:";
            cin>>queue[r];
        }
    }
    cout<<"\n";
}

void del(int &f,int &r)
{
    if (f==-1 && r==-1)
        cout<<"\nQueue is empty!";
    else
    {
        if (f==0 && r==0)
        {
            cout<<"\nDeleted element:"<<queue[f];
            f=r=-1;
        }
        else
        {
            if (f!=0 && r!=0 && f==r)
            {
```

```
                cout<<"\nDeleted element:"<<queue[f];
                f=r=-1;
            }
            else
            {
                cout<<"\nDeleted element:"<<queue[f];
                f++;
            }
        }
    }
    cout<<"\n";
}

void display(int &f,int &r)
{
    int i;
    if (r==-1 && f==-1)
        cout<<"\nQueue is empty!";
    else
    {
        if (f==0 && r==0)
            cout<<"\n"<<queue[r];
        if (f!=0 && r!=0 && f==r)
            cout<<"\n"<<queue[r];
        if (f!=r)
        {
            cout<<"\nQueue(Front...to...Rear)\n";
            for (i=f;i<r;i++)
                cout<<queue[i]<<" <- ";
            cout<<queue[r];
        }
    }
    cout<<"\n";
}

int main()
{
    int choice,front,rear;
    front=rear=-1;
    do
    {
        cout<<"\n***************QUEUE***************"
            <<"\n1.Add an element"
            <<"\n2.Delete an element"
            <<"\n3.Display the queue"
            <<"\n4.Exit\n";
```

```
        cin>>choice;
        switch (choice)
        {
        case 1:
             add(front,rear);
             break;
        case 2:
             del(front,rear);
             break;
        case 3:
             display(front,rear);
             break;
        default:
          return 0;
        }
    }
    while (choice!=4);

}
```

8 Program

Write a C program to convert an infix expression to postfix and evaluate a postfix expresssion.

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

/* MACRO FUNCTION TO CHECK WHETHER GIVEN CHARACTER IS AN OPERAND OR NOT */
#define operand(x) (x>='a' && x<='z' || x>='A' && x<='Z' || x>='0' && x<='9')
char infix[30],postfix[30],stack[30];
int top,i=0;

/* FUNCTION TO INITIALIZE THE STACK */
void init()
{
  top=-1;
}

/* FUNCTION TO PUSH AN OPERATOR ON TO THE STACK */
void push(char x)
{
  stack[++top]=x;
}
```

```
/* FUNCTION TO POP A CHARACTER STORED ONTO THE STACK */
char pop()
{
  return(stack[top--]);
}

/* FUNCTION TO RETURN IN STACK PRIORITY OF A CHARACTER */
int isp(char x)
{
  int y;
  y=(x=='('?0:x=='^'?4:x=='*'?2:x=='/'?2:x=='+'?1:x=='-'?1:x==')'?6:-1);
  return y;
}

/* FUNCTION TO RETURN INCOMING CHARACTER'S PRIORITY */
int icp(char x)
{
  int y;
  y=(x=='('?4:x=='^'?4:x=='*'?2:x=='/'?2:x=='+'?1:x=='-'?1:x==')'?6:-1);
  return y;
}

/* FUNCTION TO CONVERT THE GIVEN INFIX TO PREFIX EXPRESSION */
void infixtopostfix()
{
  int j,l=0;
  char x,y;
  stack[++top]='\0';
  for (j=0; (x=infix[i++])!='\0'; j--)
    if (operand(x))
      postfix[l++]=x;
    else
      if (x==')')
        while ((y=pop())!='(')
          postfix[l++]=y;
      else
        {
          while (isp(stack[top])>=icp(x))
            postfix[l++]=pop();
          push(x);
        }
  while (top>=0)
    postfix[l++]=pop();
}

/* MAIN PROGRAM */
```

```
int main()
{
  init();
  printf("Enter an infix expression :\n");
  scanf("%s",infix);
  infixtopostfix();
  printf("The resulting postfix expression is %s",postfix);
  return 0;
} // End of main
```

9 Program

Write a C program to

1. Create a Binary search tree
2. Search an element an element from Binary Searcch tree
3. Insert an element in a binary search tree
4. Delete a node from a binary search tree

```
1  /*bst.h*/
2  #include<stdlib.h>
3  struct bst_node {
4    int data;
5    struct bst_node *link[2];
6  };
7  struct bst_tree {
8    struct bst_node *root;
9  };
10 struct bst_tree* make_bst()
11 {
12   struct bst_tree* bst = malloc(sizeof (struct bst_tree));
13   bst->root=NULL;
14   return bst;
15 }
16 struct bst_node* make_node ( int data )
17 {
18   struct bst_node *newnode = malloc(sizeof (struct bst_node));
19   newnode->data=data;
20   newnode->link[0]=NULL;
21   newnode->link[1]=NULL;
22 }
23
24 struct bst_node *bst_insert_r ( struct bst_node *root, int data )
```

```
25 {
26   if ( root == NULL )
27     root = make_node ( data );
28   else if ( root->data == data )
29     return root;
30   else {
31     int dir = root->data < data;
32     root->link[dir] = bst_insert_r ( root->link[dir], data );
33   }
34
35   return root;
36 }
37 int bst_find_r ( struct bst_node *root, int data )
38 {
39   if ( root == NULL )
40     return 0;
41   else if ( root->data == data )
42     return 1;
43   else {
44     int dir = root->data < data;
45     return bst_find_r ( root->link[dir], data );
46   }
47 }
48
49
50 int bst_remove ( struct bst_tree *tree, int data )
51 {
52   if ( tree->root != NULL ) {
53     struct bst_node head = {0};
54     struct bst_node *it = &head;
55     struct bst_node *p, *f = NULL;
56     int dir = 1;
57
58     it->link[1] = tree->root;
59
60     while ( it->link[dir] != NULL ) {
61       p = it;
62       it = it->link[dir];
63       dir = it->data <= data;
64
65       if ( it->data == data )
66         f = it;
67     }
68
69     if ( f != NULL ) {
70       f->data = it->data;
```

```
71          p->link[p->link[1] == it] = it->link[it->link[0] == NULL];
72          free ( it );
73        }
74        else
75          return 0;
76
77        tree->root = head.link[1];
78      }
79
80      return 1;
81    }
82     void bst_destroy_r ( struct bst_node *root )
83     {
84       if ( root != NULL ) {
85         bst_destroy_r ( root->link[0] );
86         bst_destroy_r ( root->link[1] );
87         free ( root );
88       }
89     }
90
91    int bst_insert ( struct bst_tree *tree, int data )
92    {
93      tree->root = bst_insert_r ( tree->root, data );
94      return 1;
95    }
96
97    int bst_find ( struct bst_tree *tree, int data )
98    {
99      return bst_find_r ( tree->root, data );
100   }
101
102    void bst_destroy ( struct bst_tree *tree )
103    {
104      bst_destroy_r ( tree->root );
105    }
106
107   void bst_structure_r ( struct bst_node *root, int level )
108   {
109     int i;
110
111     if ( root == NULL ) {
112       for ( i = 0; i < level; i++ )
113         putchar ( '\t' );
114       puts ( "~" );
115     }
116     else {
```

```
117       bst_structure_r ( root->link[1], level + 1 );
118
119       for ( i = 0; i < level; i++ )
120         putchar ( '\t' );
121       printf ( "%d\n", root->data );
122
123       bst_structure_r ( root->link[0], level + 1 );
124     }
125   }
126
127   void bst_structure ( struct bst_tree *tree )
128   {
129     bst_structure_r ( tree->root, 0 );
130   }
```

```
 1  #include<stdio.h>
 2  #include "bst.h"
 3  int main()
 4  {
 5    struct bst_tree *bst1=make_bst();
 6
 7    do
 8      {
 9        printf("\n1. Create a binary search tree with initial elements\n"
10               "2. Search for an element in binary search tree\n"
11               "3. Insert an element in binary search tree\n"
12               "4. Delete an element in binary search tree\n"
13               "5. Display the binary search tree\n"
14               "6. Exit\n"
15               "Enter your Choice:");
16        char ch;
17        scanf("%c",&ch);
18        switch(ch)
19          {
20          case '1':
21            {
22              size_t n;
23              printf("Enter the number of elements to start with: ");
24              scanf("%u",&n);
25              printf("Now enter the elements (separated by space):");
26              while(n--)
27                {
28                  int d;
29                  scanf("%i",&d);
30                  bst_insert(bst1,d);
31                }
```

```
32            }
33            break;
34          case '2':
35            {
36              printf("Enter the element you need to search for:");
37              int d;
38              scanf("%i",&d);
39              if(bst_find (bst1,d))
40                printf("Item found.\n");
41              else
42                printf("Item not found.\n");
43            }
44            break;
45          case '3':
46            {
47              printf("Enter an element to be inserted: ");
48              int d;
49              scanf("%i",&d);
50              bst_insert(bst1,d);
51              printf("Element inserted\n");
52            }
53            break;
54          case '4':
55            {
56              printf("Enter an element to be removed: ");
57              int d;
58              scanf("%i",&d);
59              if(bst_remove(bst1,d))
60                printf("Element removed\n");
61              else
62                printf("Element Not removed. It might not be found in the tree.\n"
63            }
64            break;
65
66          case '5':
67            printf("Printing the Binary Search Tree\n");
68            bst_structure(bst1);
69            break;
70          case '6':
71            free(bst1);
72            return 0;
73          }
74      }while(1);
75    free(bst1);
76    return 0;
77  }
```

10 Program

Write a C program to implement Merge sort recursively

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  void merge ( int a[], int first, int mid, int last )
4  {
5    int i = first, j = mid, k = 0;
6    int *save = malloc ( ( last - first ) * sizeof *save );
7
8    while ( i < mid && j < last ) {
9      if ( a[i] <= a[j] )
10       save[k++] = a[i++];
11     else
12       save[k++] = a[j++];
13   }
14
15   while ( i < mid )
16     save[k++] = a[i++];
17
18   while ( j < last )
19     save[k++] = a[j++];
20
21   for ( i = 0; i < ( last - first ); i++ )
22     a[first + i] = save[i];
23
24   free ( save );
25 }
26 void mergesort_r ( int a[], int first, int last )
27 {
28   if ( first < last - 1 ) {
29     int mid = ( first + last ) / 2;
30     mergesort_r ( a, first, mid );
31     mergesort_r ( a, mid, last );
32     merge ( a, first, mid, last );
33   }
34 }
35
36 void mergesort ( int a[], int n )
37 {
38   mergesort_r ( a, 0, n );
39 }
40
41 const int MAX=20;
42 size_t ARRAY_SIZE;
```

```
43 int main() {
44   int A[MAX];
45
46   printf("How many elements? (MAX: %i)",MAX);
47
48   scanf("%i",&ARRAY_SIZE);
49   int i;
50   printf("Enter the Elements:\n");
51   for (i=0; i<ARRAY_SIZE; ++i)
52     scanf("%i",A+i);
53
54
55   mergesort( A, ARRAY_SIZE );
56
57   printf("\n\nSorted array is:  ");
58   for(i = 0; i < ARRAY_SIZE; ++i)
59     printf(" %d ", A[i]);
60   printf("\n");
61 }
```

11 Program

Write a C program to implement Quick sort

1. Recursively
2. Non-recursively

Recursive:

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void swap(int A[], int key1, int key2);                    /* swaps two arr
5  int partition( int A[], int left, int right);              /* partition an
6  void quicksort( int A[], int left, int right);             /* uses partitio
7
8  /*void print();                                            *//* auxiliary:
9
10
11 void swap(int A[], int key1, int key2){
12   int tmp;
13   tmp = A[key2];
14   A[key2] = A[key1];
15   A[key1] = tmp;
16 }
```

```
17
18 int partition( int A[], int left, int right) {
19   int pivot;
20   pivot = A[right];
21   while(1){
22     while( A[right] > pivot){                    // while the elements starting f
23       right--;                                  // divide the problem
24     }
25     while( A[left] < pivot){                     // while the elements starting f
26       left++;                                   // divide the problem (A[] consi
27     }
28     if( left < right ){
29       swap(A,left,right);                       // swap the elements OR
30     }else{
31       return left;                              // return a new left "pointer"
32     }
33   }
34 }
35
36 void quicksort( int A[], int left, int right){
37   int m;
38   if( left < right ) {                                  // make sure that we are
39     m = partition( A, left, right);                     // divide and conquer
40     quicksort( A, left, m-1);                  // sort both subarrays
41     quicksort( A, m+1, right);
42   }
43 }
44
45 const int MAX=20;
46
47 size_t ARRAY_SIZE;
48
49 int main() {
50 int A[MAX];
51
52   printf("How many elements? (MAX: %i)",MAX);
53
54   scanf("%i",&ARRAY_SIZE);
55   int i;
56   printf("Enter the Elements:\n");
57   for (i=0; i<ARRAY_SIZE; ++i)
58     scanf("%i",A+i);
59
60
61   quicksort( A, 0, ARRAY_SIZE - 1);
62
```

```
63    printf("\n\nSorted array is: ");
64    for(i = 0; i < ARRAY_SIZE; ++i)
65      printf(" %d ", A[i]);
66    printf("\n");
67  }
68  /*
69  void print(int A[]){
70    int i;
71    for(i = 0; i < ARRAY_SIZE; ++i)
72      printf(" %d ", A[i]);
73    printf("\n");
74  }
75  */
```

Non-recursive:

```
1   /*stack.h*/
2   struct Stack
3   {
4     int data;
5     struct Stack *next;
6   };
7   struct Stack* stack_push(struct Stack* stack_start, int e)
8   {
9     struct Stack* new_node = malloc(sizeof (struct Stack));
10    if(new_node==NULL)
11      return 0;
12    new_node->data=e;
13    new_node->next=stack_start->next;
14    stack_start->next=new_node;
15    return new_node;
16  }
17  int stack_pop(struct Stack* stack_start)
18  {
19    if(stack_start->next!=NULL)
20      {
21        struct Stack* to_be_deleted = stack_start->next;
22        stack_start->next=stack_start->next->next;
23        int e=to_be_deleted->data;
24        free(to_be_deleted);
25        return e;
26      }
27    return (int)(-1);
28  }
29  struct Stack* stack_new()
30  {
31    struct Stack* new_one=malloc(sizeof (struct Stack));
```

```
32    new_one->next=NULL;
33    new_one->data=0;
34    return new_one;
35  }
36  int stack_empty(struct Stack* stack)
37  {
38    return (stack->next==NULL);
39  }
```

```
 1  #include <stdio.h>
 2  #include <stdlib.h>
 3  #include "stack.h"
 4  void swap(int A[], int key1, int key2);                          /* swaps two arr
 5  int partition( int A[], int left, int right);                    /* partition an
 6  void quicksort_nr( int A[], int left, int right);                      /* uses
 7
 8  const int MAX=20;
 9  size_t ARRAY_SIZE;
10
11  int main() {
12    int A[MAX];
13
14    printf("How␣many␣elements?␣(MAX:␣%i)",MAX);
15
16    scanf("%i",&ARRAY_SIZE);
17    int i;
18    printf("Enter␣the␣Elements:\n");
19    for (i=0; i<ARRAY_SIZE; ++i)
20      scanf("%i",A+i);
21
22    quicksort_nr( A, 0, ARRAY_SIZE - 1);
23
24    printf("\n\nSorted␣array␣is:␣␣");
25    for(i = 0; i < ARRAY_SIZE; ++i)
26      printf("␣%d␣", A[i]);
27    printf("\n");
28  }
29
30  void swap(int A[], int key1, int key2){
31    int tmp;
32    tmp = A[key2];
33    A[key2] = A[key1];
34    A[key1] = tmp;
35  }
36
37  int partition( int A[], int left, int right) {
```

```
38    int pivot;
39    pivot = A[right];
40    while(1){
41      while( A[right] > pivot){                    // while the elements starting f
42        right--;                                 // divide the problem
43      }
44      while( A[left] < pivot){                     // while the elements starting f
45        left++;                                  // divide the problem (A[] consi
46      }
47      if( left < right ){
48        swap(A,left,right);                      // swap the elements OR
49      }else{
50        return left;                             // return a new left "pointer"
51      }
52    }
53  }
54  void quicksort_nr(int A[], int l, int r)
55  {
56    struct Stack* S = stack_new();
57    stack_push(S,l); stack_push(S,r);
58    while (!stack_empty(S))
59      {
60        r = stack_pop(S); l = stack_pop(S);
61        if (r <= l) continue;
62        int i = partition(A, l, r);
63        if (i-l > r-i) { stack_push(S,l); stack_push(S,i-1); }
64        stack_push(S,i+1); stack_push(S,r);
65        if (r-i >= i-l) { stack_push(S,l); stack_push(S,i-1); }
66      }
67    free(S);
68  }
```

12 Program

Write a C program to implement Heap sort.

```
1  #include <stdio.h>
2  void heapSort(int [], int);
3  void siftDown(int [], int, int);
4  int main()
5  {
6    const int MAX=20;
7    int a[MAX];
8    /* int a[]={5,1,4,88,82,1,42,1,4,5,14,74,5,4,48}; */
9    /* size_t size=(sizeof a)/(sizeof *a); */
```

```
10    printf("How many elements? (MAX: %i)",MAX);
11    size_t size;
12    scanf("%i",&size);
13    int i;
14    printf("Enter the Elements:\n");
15    for (i=0; i<size; ++i)
16      scanf("%i",a+i);
17
18    heapSort(a,size);
19
20    for(i=0; i<size; ++i)
21      printf("[%i]",a[i]);
22    printf("\n");
23    return 0;
24  }
25
26  void heapSort(int numbers[], int array_size)
27  {
28    int i, temp;
29
30    for (i = (array_size / 2)-1; i >= 0; i--)
31      siftDown(numbers, i, array_size);
32
33    for (i = array_size-1; i >= 1; i--)
34      {
35        temp = numbers[0];
36        numbers[0] = numbers[i];
37        numbers[i] = temp;
38        siftDown(numbers, 0, i-1);
39      }
40  }
41
42
43  void siftDown(int numbers[], int root, int bottom)
44  {
45    int done, maxChild, temp;
46
47    done = 0;
48    while ((root*2 <= bottom) && (!done))
49      {
50        if (root*2 == bottom)
51          maxChild = root * 2;
52        else if (numbers[root * 2] > numbers[root * 2 + 1])
53          maxChild = root * 2;
54        else
55          maxChild = root * 2 + 1;
```

```
56
57          if (numbers[root] < numbers[maxChild])
58            {
59              temp = numbers[root];
60              numbers[root] = numbers[maxChild];
61              numbers[maxChild] = temp;
62              root = maxChild;
63            }
64          else
65            done = 1;
66        }
67    }
```