EXPERIMENT - 13

Aim: To implement various storage allocation techniques algorithms

Algorithm:

Static storage allocation

- In static allocation, names are bound to storage locations.
- If memory is created at compile time then the memory will be created in static area and only once.
- Static allocation supports the dynamic data structure that means memory is created only at compile time and deallocated after program completion.
- The drawback with static storage allocation is that the size and position of data objects should be known at compile time.
- Another drawback is restriction of the recursion procedure.

Stack Storage Allocation

- In static storage allocation, storage is organized as a stack.
- An activation record is pushed into the stack when activation begins and it is popped when the activation ends.
- Activation record contains the locals so that they are bound to fresh storage in each activation record. The value of locals is deleted when the activation ends.
- It works on the basis of last-in-first-out (LIFO) and this allocation supports the recursion process.

Heap Storage Allocation

- Heap allocation is the most flexible allocation scheme.
- Allocation and deallocation of memory can be done at any time and at any place depending upon the user's requirement.
- Heap allocation is used to allocate memory to the variables dynamically and when the variables are no more used then claim it back.
- Heap storage allocation supports the recursion process.

Program :

1.) Stack

```
#include<conio.h>

int i, stk[100], top=-1, n;
void show()
{
    for(i=0;i<=top;i++)
    printf("%d\t",stk[i]);
 }
void push()
{
    int item;
    if(top == n-1)
      printf("\nStack is full.");
    else
    {
         printf("\nEnter the item: ");
         scanf("%d",&item);
         stk[++top]=item;
    }}
void pop()
{
       if(top==-1)
          printf("Stack is empty.");
       else
       {
          printf("%d is popped.",stk[top]);
          top--;
       }}
int main()
```

```
{
    int i,op;
    printf("Enter the size of the stack: ");
    scanf("%d",&n);
    do
    {
      printf("\n1 : Push");
      printf("\n2 : Pop");
      printf("\n3 : Display");
      printf("\n4 : Exit");
      printf("\nEnter your choice: ");
      scanf("%d",&op);
      switch(op)
      {
          case 1:
                push();
                break;
          case 2:
                pop();
                break;
          case 3:
                show();
                break;
      }
    }while(op!=4);
    getch();
}
```

2.) Heap

```
#include <iostream>
```

```
#include <cstdlib>
#include <vector>
#include <iterator>
using namespace std;
/*
 * Class Declaration
 */
class Heap
{
    private:
        vector <int> heap;
        int left(int parent);
        int right(int parent);
        int parent(int child);
        void heapifyup(int index);
        void heapifydown(int index);
    public:
        Heap()
        {}
        void Insert(int element);
        void DeleteMin();
        int ExtractMin();
        void DisplayHeap();
        int Size();
};
/*
 * Return Heap Size
 */
int Heap::Size()
{
    return heap.size();
```

```
}

/*
 * Insert Element into a Heap
 */
void Heap::Insert(int element)
{
    heap.push_back(element);
    heapifyup(heap.size() -1);
}
/*
 * Delete Minimum Element
 */
void Heap::DeleteMin()
{
    if (heap.size() == 0)
    {
        cout<<"Heap is Empty"<<endl;
        return;
    }
    heap[0] = heap.at(heap.size() - 1);
    heap.pop_back();
    heapifydown(0);
    cout<<"Element Deleted"<<endl;
}

/*
 * Extract Minimum Element
 */
int Heap::ExtractMin()
{
```

```
    if (heap.size() == 0)
    {
        return -1;
    }
    else
        return heap.front();
}

/*
 * Display Heap
 */
void Heap::DisplayHeap()
{
    vector <int>::iterator pos = heap.begin();
    cout<<"Heap -->  ";
    while (pos != heap.end())
    {
        cout<<*pos<<" ";
        pos++;
    }
    cout<<endl;
}

/*
 * Return Left Child
 */
int Heap::left(int parent)
{
    int l = 2 * parent + 1;
    if(l < heap.size())
        return l;
```

```
    else
        return -1;
}

/*
 * Return Right Child
 */
int Heap::right(int parent)
{
    int r = 2 * parent + 2;
    if(r < heap.size())
        return r;
    else
        return -1;
}

/*
 * Return Parent
 */
int Heap::parent(int child)
{
    int p = (child - 1)/2;
    if(child == 0)
        return -1;
    else
        return p;
}

/*
 * Heapify- Maintain Heap Structure bottom up
 */
```

```
void Heap::heapifyup(int in)
{
    if (in >= 0 && parent(in) >= 0 && heap[parent(in)] > heap[in])
    {
        int temp = heap[in];
        heap[in] = heap[parent(in)];
        heap[parent(in)] = temp;
        heapifyup(parent(in));
    }
}

/*
 * Heapify- Maintain Heap Structure top down
 */
void Heap::heapifydown(int in)
{


    int child = left(in);
    int child1 = right(in);
    if (child >= 0 && child1 >= 0 && heap[child] > heap[child1])
    {
       child = child1;
    }
    if (child > 0)
    {
        int temp = heap[in];
        heap[in] = heap[child];
        heap[child] = temp;
        heapifydown(child);
    }
}
```

```
/*
 * Main Contains Menu
 */
int main()
{
    Heap h;
    while (1)
    {
        cout<<"------------------"<<endl;
        cout<<"Operations on Heap"<<endl;
        cout<<"------------------"<<endl;
        cout<<"1.Insert Element"<<endl;
        cout<<"2.Delete Minimum Element"<<endl;
        cout<<"3.Extract Minimum Element"<<endl;
        cout<<"4.Print Heap"<<endl;
        cout<<"5.Exit"<<endl;
        int choice, element;
        cout<<"Enter your choice: ";
        cin>>choice;
        switch(choice)
        {
        case 1:
            cout<<"Enter the element to be inserted: ";
            cin>>element;
            h.Insert(element);
            break;
        case 2:
            h.DeleteMin();
            break;
        case 3:
```

```
            cout<<"Minimum Element: ";
            if (h.ExtractMin() == -1)
            {
                cout<<"Heap is Empty"<<endl;
            }
            else
                cout<<"Minimum Element:  "<<h.ExtractMin()<<endl;
            break;
        case 4:
            cout<<"Displaying elements of Hwap:  ";
            h.DisplayHeap();
            break;
        case 5:
            exit(1);
        default:
            cout<<"Enter Correct Choice"<<endl;
        }
    }
    return 0;
}
```

Output:

1.)Stack

```
Enter the size of the stack: 5

1 : Push
2 : Pop
3 : Display
4 : Exit
Enter your choice: 1

Enter the item: 2

1 : Push
2 : Pop
3 : Display
4 : Exit
Enter your choice: 1

Enter the item: 3

1 : Push
2 : Pop
3 : Display
4 : Exit
Enter your choice: 1

Enter the item: 5

1 : Push
2 : Pop
3 : Display
4 : Exit
Enter your choice: 2
5 is popped.
1 : Push
2 : Pop
3 : Display
4 : Exit
Enter your choice: 1
```

```
Enter the item: 7

1 : Push
2 : Pop
3 : Display
4 : Exit
Enter your choice: 3
2        3        7
1 : Push
2 : Pop
3 : Display
4 : Exit
Enter your choice: 4


...Program finished with exit code 0
Press ENTER to exit console.
```

2.) Heap

```
Enter your choice: 1
Enter the element to be inserted: 4
------------------
Operations on Heap
------------------
1.Insert Element
2.Delete Minimum Element
3.Extract Minimum Element
4.Print Heap
5.Exit
Enter your choice: 1
Enter the element to be inserted: 9
------------------
Operations on Heap
------------------
1.Insert Element
2.Delete Minimum Element
3.Extract Minimum Element
4.Print Heap
5.Exit
Enter your choice: 4
Displaying elements of Hwap:  Heap -->  4 7 4 9
------------------
Operations on Heap
------------------
1.Insert Element
2.Delete Minimum Element
3.Extract Minimum Element
4.Print Heap
5.Exit
Enter your choice: 5


...Program finished with exit code 0
Press ENTER to exit console.
```

Result : Different Algorithms for various storage allocation algorithms were implemented successfully