W5.1(revised)

Memory Allocation

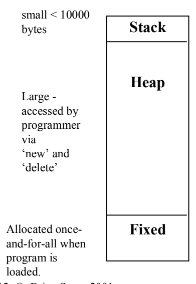
Ways of Using Structure Member and Pointer Operators

```
#include <stdio.h>
struct card {
   char *face;
   char *suit;
};
main(){
   struct card a;
   struct card *aPtr;
   a.face = "Ace";
   a.suit = "Spades";
   aPtr = &a;
   printf("%s%s%s\n%s%s%s\n%s%s%s\n",
         a.face, " of ", a.suit,
         aPtr->face, " of ", aPtr->suit,
         (*aPtr).face, " of ", (*aPtr).suit);
   return 0;
}
```

N.B. here ! Dereferencing pointers.

Memory and its Management

- **new** & **delete**
- Allow us to allocate memory to data structures during program execution
- May allocate memory for arrays, structs and basic types etc.



small < 10000 bytes
Stack
Heap
Large - accessed by programmer via 'new' and 'delete'
Fixed
Allocated once-and-for-all when program is loaded.

Heap, Stack, Memory and Visibility

- Consider the program fragment below...

```
int globalA[1000];                 // Fixed Memory
static int globalB[1000];          // Fixed Memory
int funcA(int x,int y)             // x & y are created from the stack
{
   int localA[10];                 //Stack memory
   static int localB[1000];        //Fixed memory
   int *p;                         //Stack memory
   p=new int[100000];              //Heap memory
   .....
   delete p;                       // Return to the heap
                                   // on exit all memory taken from the the stack
}                                  // is restored
static int funcB(int x)            // module scope!!!
{              .......}
```

What's the Visibility Here ?

```
int globalA[1000];               // potentially visible to all modules
static int globalB[1000];        // visible throughout this module only

int funcA(int x,int y)           // x & y are visible only inside this function
{
   int localA[10];               //visible within this function
   static int localB[1000];      //visible within this function
   int *p;
   p=new int[100000];            //visible within this function
   .....
   delete p;                     // p is about to be destroyed ......
}

static int funcB(int x)
{
           ..........
}
```

Visibility…..

In a second module

```
extern int globalA[];          //Valid
extern int globalB[];          // Illegal
extern int funcA(int,int);     //Valid
extern int funcB(int);         //Illegal
```

The keyword **static** is a little overused!

Pulling It All Together.

- A simple stack program which constructs and releases a stack is a good illustrator of many of the programming constructs seen so far.
- Remember, a stack places new data on top of the structure, and deletions may only occur at the top, like a stack of plates.
- Consider the following code…See problems?

```
#include <iostream.h> //MS
#define N 1000
#define BOS -1
struct stack
{   int size;
   int *array;
   int top;
};
void construct(struct stack s){
   s.size=N;
   s.top=BOS;
   s.array=new int [N];
}
int is_empty(struct stack s){
   return (s.top==BOS);
}
int is_full(struct stack s){
   return (s.top==s.size-1);
}
void push(int value,struct stack s){
   if (is_full(s)) return;
   s.array[++s.top]=value;
}
```

```
int pop(struct stack s)
{
   if (is_empty(s))
   {
      cout << "No elements in stack\n";
      return 0;
   }
   return s.array[s.top--];
}   //function pop
void main()
{
   struct stack astack;
   int v,i,size;
   construct(astack);
for (i=0;i<N;i++)
      push(i,astack);
   size=astack.top;
   for (i=0;i<=size;i++)
   {
      v=pop(astack);
      cout << "element = " << v << endl;
   }   //end for
}   //end main
```

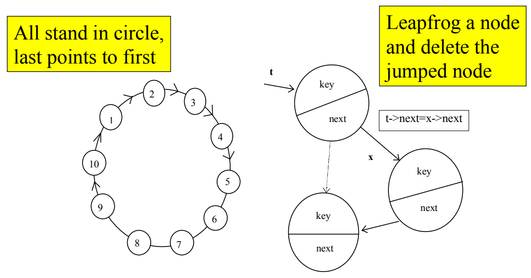
Graded Assignment

- This program **does not work**, flawed in a simple and fundamental way!!!
- There are at least three ways of fixing it.
  - Some better than others!
  - Produce two (good ones).
- When it is working, break it up into two files. A set of utility functions in one file and main in the other. Use header files as appropriate.
- TIP: reduce the size of the stack to debug this.

Another Problem to Solve

The year is 66AD. A group of desperate people have decided to commit mass suicide, but without anyone killing themselves. Their leader is one Josephus. He doesn't want to die, but his followers will kill him unless he agrees. Josephus proposes the following. All the men will stand in a circle, numbered 1 to N. They will then proceed to kill the M^{th} man in the circle proceeding clockwise. The survivors close up the gap, and continue to kill the M^{th} man, until only one is left. Josephus needs a short program for his lap-top which will predict the outcome, so that he can decide the choice of M, so that he is the last survivor (hypocrisy again).

The Problem of Josephus


All stand in circle, last points to first
Leapfrog a node and delete the jumped node
t
key
next
t->next=x->next
x
key
next
key
next
1
2
3
4
5
6
7
8
9
10

A Solution for Josephus

Bit of C++ I\O here, it simplifies!

```
#include <iostream.h>
/* use a circular linked list type of
node structure to represent the men.
 Each man has a number 'key' and
a pointer to the next man */
struct node
{
    int key;
    struct node *next;
};
```

Self-type referential pointer

leapfrog

```
void main(){
int i,N,M;
struct node *t,*x;
cout << "Enter Number of people= ";
cin >> N;
cout << "Enter Killing order= ";
cin >> M;
t=new node;
t->key=1;   x=t;
for (i=2;i<=N;i++)
   {  // build linked list
    t->next=new node;
    t=t->next;
    t->key=i;
    }
t->next=x; // last points to first
while (t != t->next)
   { // Kill every M-th man
   for (i=1;i<M;i++)    t=t->next;
   cout << t->next->key << ' ';
   x=t->next;    t->next=x->next;
   delete x;
   }
cout << t->key << '\n';
} //end main
```

Semester Project (Stage 1)

To be rewritten later using OO Constructs.

The Dictionary Project

Using whatever data structures and algorithms that you are familiar with and feel appropriate, write a program which acts as a dictionary, or word store. Your program should allow the user to type in a word and store it in the dictionary. Duplicate words are not allowed. The user should also be allowed to check the presence of a word in the dictionary and be told whether that word is present or not. The application should also be able to list the words in the word store in alphabetical order, on the screen. You must also allow deletions of words.
Do not use Objects and Classes.