CMSC 11
Pointers, Dynamic Memory Allocation, and Linked Lists

Variables

- When we define a variable in C, we have access to three information:
  1. the name of the variable
  2. the value of the variable
  3. address of the variable in memory

  ex.

```
int a = 5;
```

Memory and Addresses

- A computer has a component called memory where data used in computations are stored
- There are two basic types of memory: internal (RAM) and external (disks)
- Memory size is measured in terms of bytes (512 Mb)
- Each byte in the memory can store 8-bit of information
- Information requiring more than one byte to store in memory are stored in adjacent bytes, depending on the computer architecture
- Each byte in memory has an address

Pointer Variables or Pointers

- A pointer is a variable whose value is an address, it has the same information available for ordinary variables
  - ex:

```
int *p;
```

  - Defines a pointer variable p that *points* to a memory location that can contain an int value
- Operations that can be applied to a pointer
  - assignment

```
p = &a;
```

  - dereference (access the memory location pointed to by p)

```
int x = 6 + *p;
```

- pointer arithmetic

```
p++;
```

Pointers to Structures

- Given the structure definition below

```
typedef struct _Student{
     char name[30];
     int age;
}Student;
```

- We can declare two variables:

```
Student s;
Student *ps;
```

- To access the fields of the structure using the two variables

```
s.age
ps->age
```

Dynamic Memory Allocation

- Global variables are stored in the *global environment*
- Local variables defined inside a function are *statically allocated* in the memory area part of *local environment(stack)* of the function
- During function call, information to return control to the calling function is *dynamically allocated* in the memory area of execution environment's *stack*
- *Static memory allocation* means that before the program is run, memory is already allocated
- *Dynamic memory allocation* means that memory is allocated while the program is running
- Can we dynamically allocate memory to store data in our programs?
- `void *malloc(size_t size);`
- example:

```
(1)  #include <stdio.h>
(2)  #include <stdlib.h>
(3)  int main(){
(4)       int *p;
(5)       p = (int *)malloc(sizeof(int));
(6)       printf("Enter a number: ");
(7)       scanf("%d",p);
(8)       printf("The number you entered: %d\n", *p);
(9)       return 0;
(10  }
```

Line 4: Defines a pointer variable p

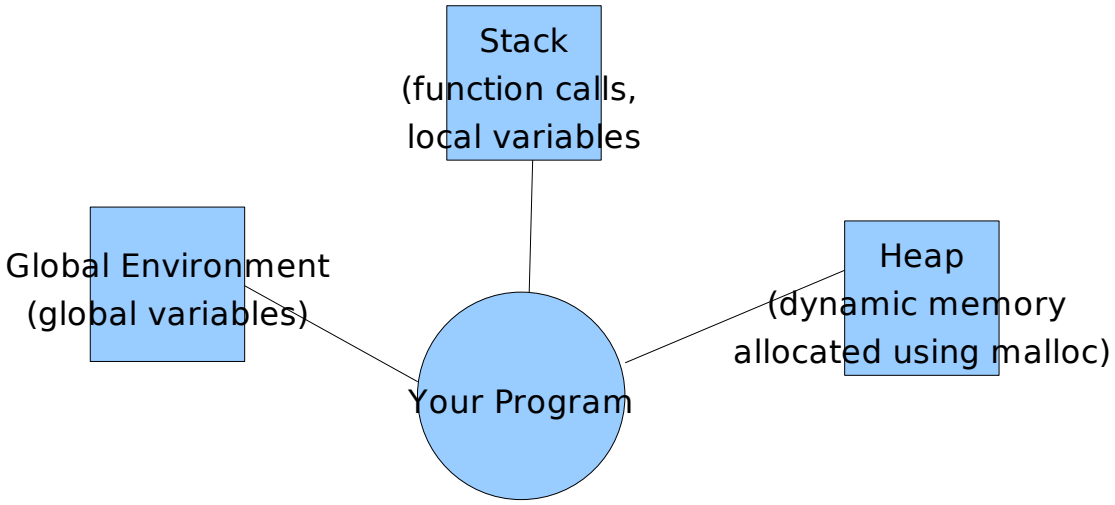
Line 5: Allocate enough memory to hold an integer value, address of memory allocated is assigned to p. `(int *)` casts (converts one type to another) the return value of `malloc()` to an integer pointer which is what p is.

Line 7: Ask user to enter an integer. Note the second parameter of `scanf()`, normally the address of operator(&) is used. Here, since p is a pointer, there is no need to use &

Line 8: Dereference p to display the value entered

- Memory allocated using `malloc()` is created in the *heap*

Memory Regions used by Programs



Linked List

- A list is a data structure(a way to represent data) that allows us to maintain a collection of items and perform operations on the items. Items in the list are ordered: there is the first element, second element and so on..
- Typical operations on a list: *create(), add(List, item), print(List), delete(List, item), search(List, item)*
- We can use arrays to implement lists!
  - However, the size of the array should be specified during *compile time.* What if we do not know what the number of items in the list will be? We'll end up allocating memory which will not be used!

- Linked list is an implementation of the list data structure that allows us to avoid the problem with the array implementation.
- Linked list uses *structures*, *pointers,* and *dynamic memory allocation*
- A linked list is composed of *nodes.* Each node contains a list *item* and a *pointer to the next node* in the list. An example declaration of a node is given below:

```
typedef struct _Node{
     int data;
     struct _Node *next;
}*Node;
```

- Nodes are normally dynamically allocated, we can write a function that dynamically creates a node and returns the address of the allocated memory.

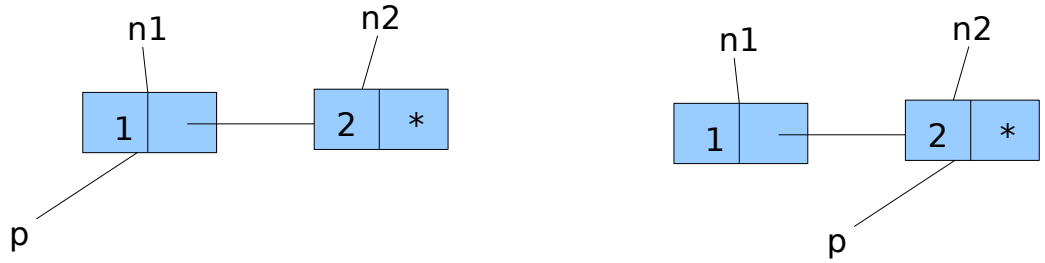
```
Node createNode(int i){
     Node n;
     n = (Node)malloc(sizeof(struct _Node));
     n->data = i;
     n->next = NULL;
     return n;
}
```

- Lets create two nodes and try to link them!

```
int main(){
     Node n1=NULL,n2=NULL;
     n1 = createNode(1);
     n2 = createNode(2);
     n1->next = n2;
     printf("n1->next: %X\n",n1->next);
     printf("n2: %X\n",n2);
}
```

- Graphical Representation

- Traversing a linked list requires a temporary pointer that moves from one node to another until the end of list is reached



- The movement is illustrated below

```
Node p=NULL;
p=n1;
while (p != NULL){
     printf("data: %d\n",p->data);
     p = p->next;
}
```

example:

```
/* linklist.h */
#include <stdio.h>
#include <stdlib.h>
#ifndef __LINKLIST_H__
#define __LINKLIST_H__
typedef struct _Node{
     int data;
     struct _Node *next;
}*Node;
typedef struct _List{
     struct _Node *head;
}*List;
void printNode(Node n);
Node createNode(int i);
Node search(List l, int i);
#endif
```

```
/* list.h */
#include <stdio.h>
#include <stdlib.h>
#ifndef __LIST_H__
#define __LIST_H__
#include "linklist.h"
List create();
void add(List l, int i);
void delete(List l, int i);
void print(List l);
#endif

/* linklist.c */
#include <stdio.h>
#include <stdlib.h>
#include "list.h"
List create(){
     List l;
     l = (List)malloc(sizeof(struct _List));
     l->head = NULL;
     return l;
}

Node createNode(int i){
     Node n;

     n = (Node)malloc(sizeof(struct _Node));
     n->data = i;
     n->next = NULL;
     return n;
}

void add(List l, int i){
     Node n,p;
     n = createNode(i);
     if (l->head == NULL){
          l->head = n;
     }else{
          p = l->head;
          while (p->next != NULL){
               p = p->next;
          }
          p->next = n;
     }
}
Node search(List l, int i){
     Node p;
     p = l->head;
```

```
     while (p != NULL){
          if (p->data == i){
               return p;
          }
          p = p->next;
     }
     return NULL;
}
void printNode(Node n){
     if (n == NULL){
          return;
     }
     printf("%d\n",n->data);
}
void delete(List l, int i){
     Node p,n;
     n = search(l,i);
     p = l->head;
     if (n !=NULL){
          if (p == n){
               p = p->next;
               l->head=p;
          }else{
               while (p->next != n){
                    p = p->next;
               }
               p->next = n->next;
          }
          free(n);
     }
}
void print(List l){
     Node p;
     p = l->head;
     while (p != NULL){
          printNode(p);
          p = p->next;
     }
}
```

```
/* linktest.c */
#include <stdlib.h>
#include "list.h"
int main(){
     List l;
     printf("Creating list...\n");
     l = create();
     printf("Adding items on the list...\n");
     add(l, 1);
     add(l, 2);
     add(l, 3);
     add(l, 4);
     printf("Content of list...\n");
     print(l);
     printf("Searching 3 on the list..\n");
     printNode(search(l,3));
     printf("Deleting 4 from list...\n");
     delete(l,4);
     printf("Content of list with 4 deleted...\n");
     print(l);
     return 0;
}
```

Build

```
$gcc -o linktest.exe linktest.c linklist.c
```

Output

```
$./linktest.exe
Creating list...
Adding items on the list...
Content of list...
1
2
3
4
Searching 3 on the list..
3
Deleting 4 from list...
Content of list with 4 deleted...
1
2
3
```