

Dynamic Data Structures

Lecture 4

Dynamic Data Structures

- ❑ Arrays and structures are static and contiguous.
- ❑ Dynamic data structures are dynamic and may be non-contiguous.
- ❑ Dynamic data structures may grow or shrink during the execution of the program.

Dynamic Memory Allocation

- ❑ Used to support dynamic data structures
- ❑ Dynamically allocated memory is determined at runtime
- ❑ A program may create as many or as few variables as required, offering greater flexibility
- ❑ Dynamic memory is finite
- ❑ Dynamically allocated memory may be freed during execution

Dynamic Memory Allocation

- ❑ Memory is allocated using
 - malloc (memory allocation)
 - calloc (cleared memory allocation)
- ❑ Memory is released using
 - free
- ❑ The size of memory requested by malloc or calloc can be changed using
 - realloc

malloc and calloc

- Both functions return a pointer to the newly allocated memory
- The pointer returned by these functions is declared to be a void pointer
 - Use a cast operator to coerce it to the proper pointer type
- If memory cannot be allocated, the value returned will be a NULL pointer

Allocating Arrays Dynamically

```
int     npts = 500;
double  *x;
int     *p;

/* Allocate memory for 500 doubles. */
x = (double *)malloc (npts * sizeof(double));

/* Allocate memory for 500 integers. */
p = (int *)calloc (npts * sizeof(int));
```

Allocating 2D Arrays Dynamically

```
int     rows = 10;
int     cols = 20;
double *x;

/* Allocate a matrix for 200 doubles. */
x = (double *)malloc (rows * cols * sizeof(double));
```

- To access an element [a][b]
 - $x + a * \text{cols} + b$
 - A pointer reference to `s[0][1]` would be `*(x+1)`
 - A pointer reference to `s[1][1]` would be `*(x+21)`

Allocating 2D Arrays Dynamically

```
int     rows = 10;
int     cols = 20;
double **x;

/* Allocate a matrix for 200 doubles. */
x = (double **)malloc (rows * sizeof(double *));
for (i = 0; i < rows; i++)
{
    x[i] = (double *)malloc (cols * sizeof(double));
}

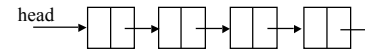
□ To access an element [a][b]
  ➤ x[a][b]
```

Linked Lists

- Group of structures (nodes), connected by pointers.
- A node consists of data (one or more variables) and a pointer to the next node.
- Nodes may be ordered
 - According to some specific rule about the data in the node.

Linked Lists

- Head pointer points to the 1st node.
- Nodes are accessed through the head pointer.
- The pointer in the last node is NULL.



Linked Lists

- Linked lists can be implemented
 - in an array
 - with dynamically-obtained structures

Linked Lists

- Linked lists can be implemented
 - in an array
 - with dynamically-obtained structures

Dynamic linked lists

Dynamic Linked Lists

□ Defining the nodes

```
#define NODE struct node
...
struct node
{
    int    info;
    NODE  *next;
}
...
```

Dynamic Linked Lists

□ Creating an empty list

```
NODE *head = NULL;
```

Dynamic Linked Lists

□ Common Operations

- Insert a node
- Delete a node
- Search a node
- Print the list

Dynamic Linked Lists

□ Insertion

- Nodes are inserted at the front of the list
- Each node is obtained with a call to malloc

```
NODE *p;
...
if ((p = (NODE *)malloc (sizeof (NODE))) == (NODE *)NULL)
{
    printf ("memory could not be allocated\n");
    return 0;
}

p->info = number;
p->next = head;
head = p;
```

Ordered Linked Lists

- Linked list in which the nodes follow some ordering
 - Example: names in alphabetical order
- Some operations are implemented differently
 - Insert a node
 - Delete a node
 - Search a node