Dynamic Data Structures

Lecture 6

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
- Dynamically allocated memory may be freed during execution

Dynamic Memory Allocation

- Memory is allocated using
  - malloc

  void *malloc(size_t size);

  Allocate a block of `size` bytes,
  return a pointer to the block
  (NULL if unable to allocate block)

malloc

- The function 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

Dynamic Memory Allocation

- Memory is released using
  - Free

  void free(void *pointer);

  Given a pointer to a block of allocated memory, deallocate the memory

Allocating Arrays Dynamically

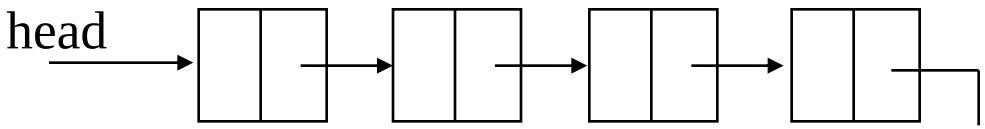
```
int             npts = 500;
double          *x;
int             *p;

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

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

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
  - with dynamically-obtained structures

Linked Lists

- Linked lists can be implemented
  - with dynamically-obtained structures

Dynamic linked lists

Dynamic Linked Lists

- Defining the nodes
  - Example

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

Dynamic Linked Lists

- Creating an empty list

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

Unordered Linked Lists

- Common Operations
  - Insert a node
    - Insert before the head
  - Search a node
    - Need to search the entire list
  - Delete a node
    - Need to search the entire list
  - Output the list

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

Unordered Linked Lists

- Functions
  - Insert at the head
  - Search a specified node
  - Delete a specified node
  - List all the nodes

Ordered Linked Lists

- Linked list in which the nodes follow some ordering
  - Example: names in alphabetical order

Ordered Linked Lists

- Some operations are implemented differently
  - Insert a node
    - Need to find the right spot
  - Search a node
    - Search ends when the first node out of range is reached
  - Delete a node
    - Search ends when the first node out of range is reached

Ordered Linked Lists

- Functions
  - Insert at the head
  - Search a specified node
  - Delete a specified node
  - List all the nodes