16.070 Introduction to Computers and Programming

April 3 **Recitation 8** **Spring 2002**

Topics:

- Next Week's Assignments
- Problem Set Issues
- Debugging Hints
- Dynamic Memory Allocation
- Abstract Data Types
  - Lists
  - Stacks
  - Queues
  - Trees
- Other

Next Week's Assignments

Exam #2 - 2 hour evening test on Wednesday 4/10 in Walker
Problem Set 8 – Out: now, Due: Friday 4/12
Final Project – Part A Out: 4/12, Due: 4/22

Problem Set Issues

1. Read all instructions carefully.
2. Follow the instructions.
3. Use the correct file names with fopen

   If we give you some file "infile.dat" to use with your program. You should call it "infile.dat" in your program.

   Example:
   ```
   FILE *input;
   input = fopen("infile.dat", "r");
   ```

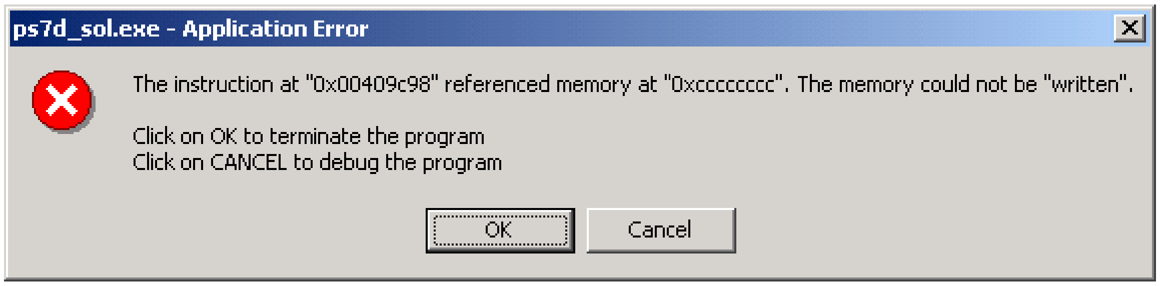
Debugging Hints

General Debugging Hint:

If you get a strange syntax error like "line(foo): syntax error '='…", check all code at and ABOVE line foo. You probably have a missing/extra symbol in that section of your code (missing () {} ; or used = in a #define).

Memory Access Errors:

1. Do you recognize this error message?


ps7d_sol.exe - Application Error

The instruction at "0x00409c98" referenced memory at "0xcccccccc". The memory could not be "written".

Click on OK to terminate the program
Click on CANCEL to debug the program

This basically means you have tried to access a piece of memory that does not belong to your program. Common culprits are

- **a.** File did not open
- **b.** Tried to use a pointer that was not initialized or was still set to NULL
- **c.** Used *ptr when you meant ptr or vice-versa (dereference v/s address)
- **d.** Exceeded array bounds
- **e.** Incorrect scanf or fscanf parameter (missing &, used *ptr instead of ptr, used array[1] instead of array)

2. Side effects of this error

   In order of increasing seriousness…

   - **a.** Nothing – program stops running at some point with this error message, but otherwise no harm done.
   - **b.** Values in your program change mysteriously during run time
   - **c.** Memory being used by your operating system or by Visual C is changed causing mysterious crashes and compile errors.

3. Solutions

   - **a.** ALWAYS check that ptr != NULL after any fopen or malloc call. The function exit(-1); (located in the stdlib.h library) will exit your program and return an error code to your operating system. A good use of exit() is

```
Ptr_file = fopen(“test.txt”, r”);
If(ptr_file == NULL)
{
        printf(“Error opening file\n”);
        exit(-1);
}/*end if*/
```

   - **b.** In the case of side effects b or c, close and reopen Visual C or in extreme cases reboot your machine.

Dynamic Memory Allocation

1. Why

More versatile – Up to now, we’ve had to hard code the size of all our arrays. This imposed an upper limit on the amount of data we put in each array at run time. We had no way of increasing the size of our arrays “on the fly” if the user suddenly needed more memory. Now we can specify as much or as little memory as is needed for each user.

Save Space – Sometimes (maybe even often) we don’t need to use the entire array we declared at compile time. If we have several large arrays declared this could get very wasteful. It’s better to allocate just enough memory for your application, especially if you’re going to have many copies of an array in use.

Implement ADT – Dynamic memory allocation is used to implement dynamic abstract data types that grow and shrink as you insert and delete from them.

2. How

The functions for dynamic memory allocation are located in the stdlib.h library.
The ones you'll use most often are

(void *) malloc(int)
free(void *)

Other useful functions are calloc() and realloc().

To create an array of array_size floats:

Allocates memory → ptr_floatarray = (float *) malloc(array_size * sizeof(float));

.
.   Pointer   Type of memory to return   How many bytes do you want?
.

Releases memroy → free(ptr_floatarray);

Example1: http://web.mit.edu/16.070/www/recitation/R8/dynam1.c

```
/*********************
* 16.070 Recitation 8
* Dynamic Mem example1
*********************/

#include <stdio.h>
#include <stdlib.h>

int main(void)
{
        int *ptr_int = NULL;
        int size_array = 0;
        int i = 0;

        printf("How many ints? ");
        scanf("%d", &size_array);

        /*allocate memory*/
        ptr_int = (int*) malloc(size_array * sizeof(int));

        /*error checking*/
        if(ptr_int == NULL)
        {
                printf("memory allocation failed\n");
        }
        else
        {
                /*populate array*/
                for(i=0; i<size_array; i++)
                {
                        *(ptr_int + i) = i;
                }

                /*print array*/
                for(i=0; i<size_array; i++)
                {
                        printf("%d ", *(ptr_int + i));
                };

                /*free memory*/
                free(ptr_int);
        }
        return 0;
}/*end main*/
```

Example 2: Dynamic Matrix Multiplication
http://web.mit.edu/16.070/www/recitation/R8/dynam_matrix_mult.c

3. Memory Leak

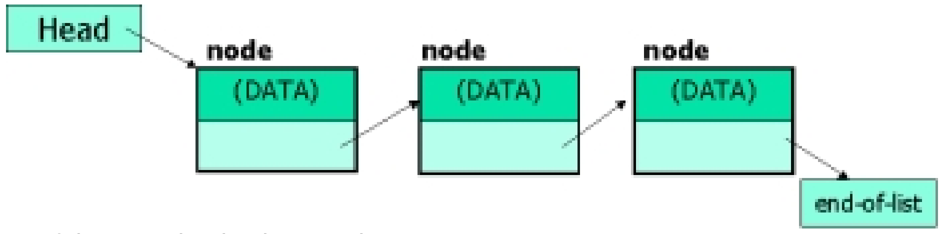
Keep track of your pointers, and always free memory when you are done using it. If you somehow lose the pointer to a block of memory, then you have lost that block of memory until you restart your program. If your program is designed to run for a long time (or maybe indefinitely) like an operating system, this is bad. You may notice the amount of system memory available to Windows decreases the longer it's been since you rebooted. This is memory leak at work.

Abstract Data Types (summary)

An Abstract Data Type (ADT) is a mathematical object and a set of operations on that object. ADT's are defined by what they do, not how they are implemented. ADT's are widely studied so there are lots of algorithms and code segments we can "steal". Figuring out when to use which abstraction, however, is two or three other courses worth of information.

Lists

What is it?



A list is one of the most basic abstract data types.
The properties of lists are

1. Nodes: data and a link to at least one other node or the end-of-list
2. Head: points to the first node in the list or the end-of-list
3. Insert: adds a node to the list
4. Delete: removes a node from the list

The data in a list usually must be accessed sequentially. Compare this to a (one dimensional) array, where you can jump to any element of the array.

In the most general case you can Insert to or Delete from any node in the list.

How to implement in C

See Terry's handout.
C files located at http://web.mit.edu/tezzer/Public

```
/*  Node definition */
struct node{
        int data;
        struct node * next;
};
typedef struct node NODE;

/*  Function Prototypes: */
NODE * NewNode(void);           /*creates a new, empty node */
NODE * Insert( NODE * list, NODE * object);       /*adds a node to the tail of a list */
NODE * Delete( NODE * list, int data);   /*removes the first node with a given data value*/
```

```
void PrintList(NODE * list)          /* traverses a list and prints data */
```

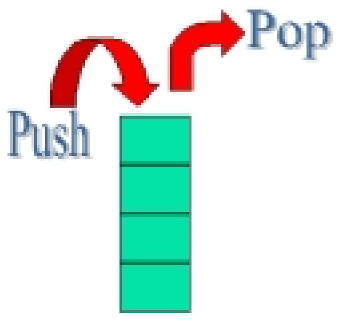
How to use
Hand Example:
Create a list of 3 integers: 5 42 34

Delete 42

Code Example: http://web.mit.edu/16.070/www/recitation/R8/list_example.c

```
int num = 0;
int i = 0;
NODE * MyList = NULL, *temp = NULL;

/******************create list******************/
for(i=1; i<=3; i++)
{
    /*create a new node*/
    temp = NewNode();

    /*get an integer*/
    printf("Enter an integer:");
    scanf("%d", &num);

    /*insert the number into the list*/
    temp->data = num;
    MyList = Insert(MyList, temp);
 }/*end for*/

 /*************Remove an entry*************/
 printf("\nEnter a number to remove: ");
 scanf("%d", &num);
 MyList = Delete(MyList, num);
```

Stacks
What is it?



A type of list that Inserts only to the Head (Push) and Deletes only from the Head (Pop).

Last-In-First_Out (LIFO): the last item “pushed” onto the stack is the first item “poped” off the stack.

How to implement in C
See Terry’s handout.
C files located at http://web.mit.edu/tezzer/Public

```
/***  Node definition */
```

```
struct stack{
  int items_on_stack;
  struct node * top;
};

struct node{
  int data;
  struct node * next;
};

typedef struct stack STACK;
typedef struct node NODE;

/*  Function Prototypes:*/
NODE * NewNode(void);            /* called by Push to create stack item */
void Push( STACK * st, int data); /*adds a data item to the stack */
int Pop(STACK * st);      /* removes item at top of stack and returns it */
void PrintStack(STACK * st);      /* Prints all data on stack */
```

How to use

Hand Example:

Push 3 integers onto the stack.

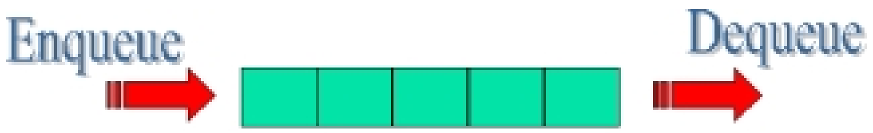
What order do they come off in?

Code Example: http://web.mit.edu/16.070/www/recitation/R8/stack_example.c

```
int num, i;
STACK MyStack;

  /*initilize the stack*/
  MyStack.items_on_stack = 0;
  MyStack.top = NULL;

 /***************Push 3 ints******************/
  for(i=0;i<3;i++)
  {
        printf("Enter an integer: ");
        scanf("%d", &num);
        Push(&MyStack, num);
        PrintStack (&MyStack);
  }

/******************Pop 3 ints*****************/
  for(i=0;i<3;i++)
  {
        num = Pop(&MyStack);
        printf("removed data %d from stack\n",num);
        PrintStack (&MyStack);
  }
```

Queues

What is it?



A type of list that Inserts only to the Tail (Enqueue) and Deletes only from the Head (Dequeue).

First-In-First_Out (FIFO): the first item "enqueued" onto the stack is the first item "dequeued" off the stack.

How to implement in C

See Terry's handout.
C files located at http://web.mit.edu/tezzer/Public

```
/***  Node definition*/
struct node{
  int data;
  struct node * next;
};
typedef struct node NODE;

/*  Function Prototypes:*/
NODE * NewNode(void);            /*creates a new, empty node */
void Enqueue( NODE ** queue, NODE * object); /*adds a node to the end of the
                                                      queue*/
int  Dequeue( NODE ** queue); /*removes a node from the head of a queue and
                                  returns the data value */
void PrintQueue(NODE ** queue);    /* prints the data on a queue */
```

How to use

Hand Example:

Enqueue 3 integers.

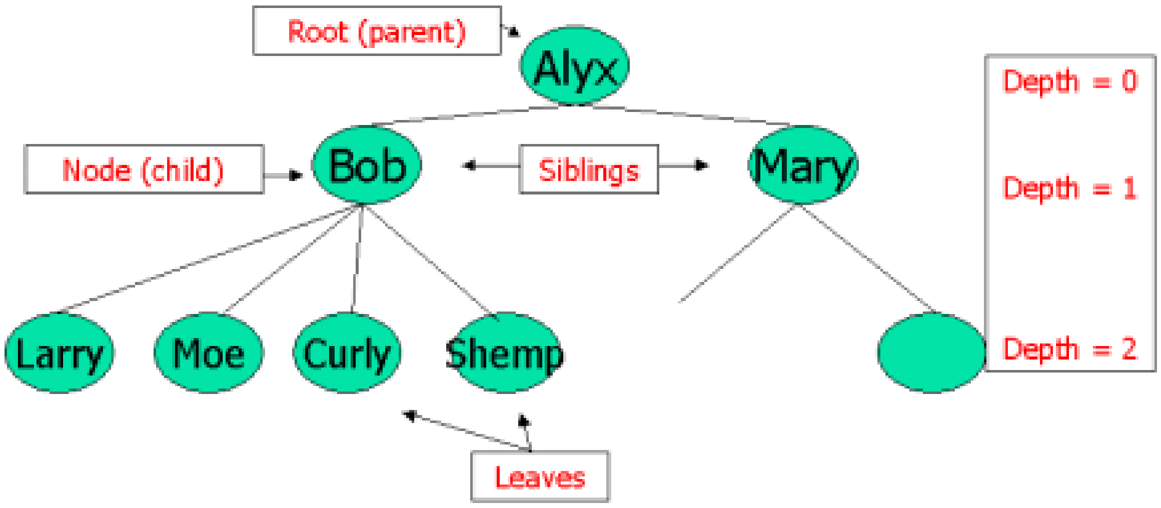
What order do they Dequeue?

Code Example: http://web.mit.edu/16.070/www/recitation/R8/queue_example.c

```
int num = 0, i = 0;
NODE * MyQueue = NULL, *temp = NULL;

/******************enqueue stuff*****************/
  for(i=0;i<3;i++)
  {
        printf("Enter integer: ");
        scanf("%d", &num);
        temp = NewNode();
        temp->data = num;
        Enqueue(&MyQueue, temp);
        PrintQueue (&MyQueue);
  }

/***************dequeue stuff******************/
for(i=0;i<3;i++)
{
        num = Dequeue(&MyQueue);
        printf("Dequed number %d\n",num);
        PrintQueue (&MyQueue);
  }
```

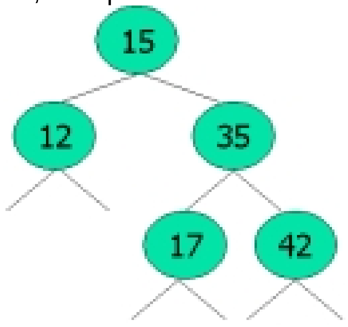
Trees

What is it?



- The first node in a tree is the **Root Node**
- Nodes attached to a Root are **Children**
- Nodes at the same depth in a tree are **Siblings**
- The deepest nodes (nodes with no children) are called **Leaves**

Binary Tree: If Node > Root Node, then place Node to the right.
If Node < Root Node, then place Node to the left.



It is much faster to search a binary tree than it is to search a generic list.

How to implement in C

Complicated

How to use

Hand Example:

Create a Binary Tree with nodes 9 7 26 43 15 1

Is 15 on the tree?

How would we find 15 in an equivalent list?