Lecture 6

Dynamic data structure

A “link list”

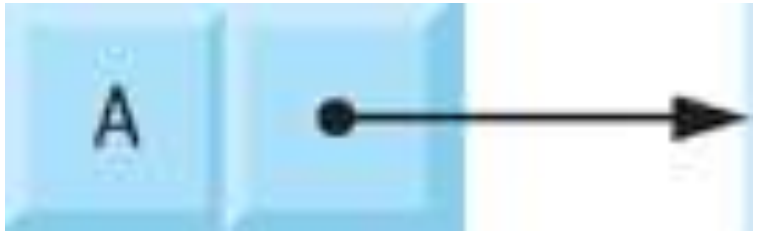
Array versus link list

- An array is a structure whose size is fixed at run time; e.g. empdb
- In an array to delete a record it has to be marked as deleted.
- If the array is full cannot add any more records
- These restriction are overcoming using a dynamic “link List” of records: the size can be increased/decrease during run time.
- This requires the use of malloc to create a new node and free to remove a dynamically created note
- It also requires some way of linking these nodes :
- Hence a structure refer to as a node

The node

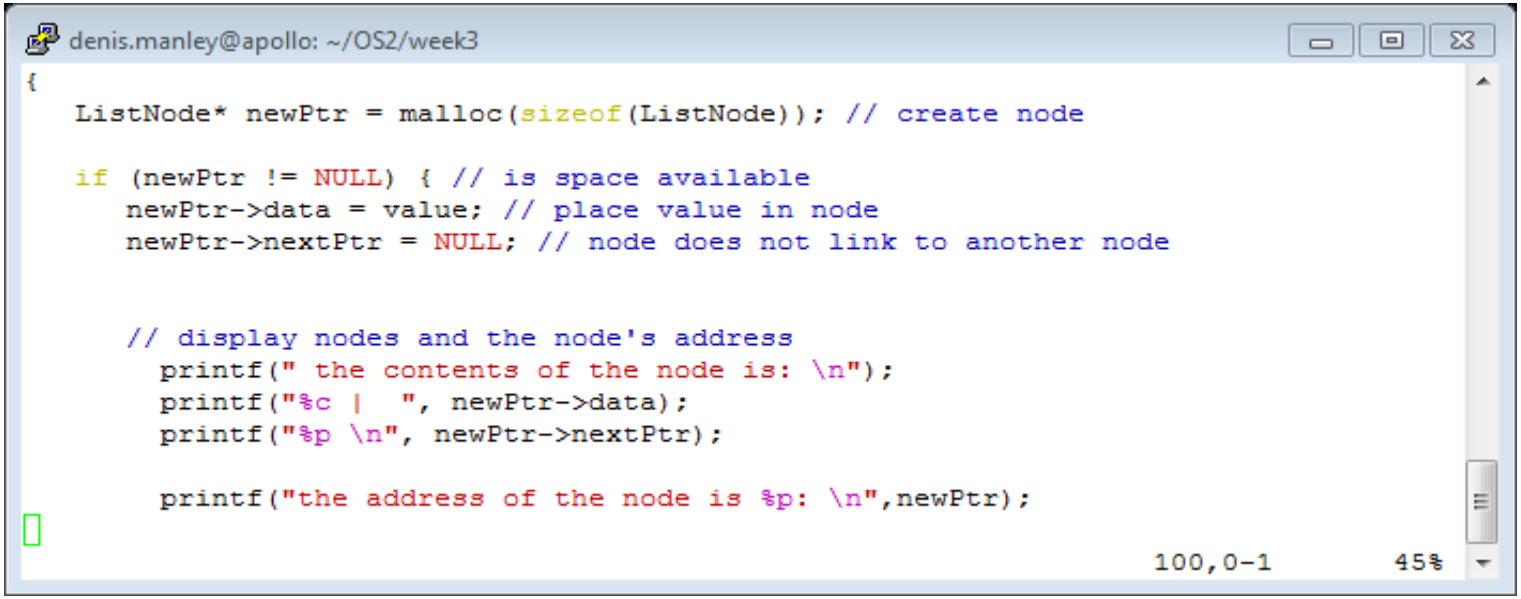
- A node had two parts:
  - one for containing data (an integer; an employee record….)
  - A pointer to another node (link between nodes)
- The following c code is a struct for such a node: data in this example is only an int

```
denis.manley@apollo: ~/OS2/week3

// self-referential structure
struct listNode {
   char data; // each listNode contains a character
   struct listNode *nextPtr; // pointer to next node
};

typedef struct listNode ListNode; // synonym for struct listNode
//typedef ListNode* ListNode*; // synonym for ListNode*
                                                    13,1          2%
```



Dynamically create a node

- Create node and insert data using Malloc:
- Display the contents of the node and the address of the node
- Refer to ListNode.c



ListNode.c sample code

```c
// self-referential structure
struct listNode {
   char data;
   struct listNode *nextPtr; // pointer to next node
}; typedef struct listNode ListNode; // synonym for struct listNode

int main(void)
{

        char value;

   // create a node and add data to the node
   ListNode* newPtr = malloc(sizeof(ListNode)); // create node

   // enter a character
        printf("enter a character: ");
        scanf("%c",&value);

   // if nodes is created assign values to node
   if(newPtr != NULL)
   {
           newPtr->data = value;
           newPtr->nextPtr = NULL; // node does not link to another node
   }


        // **********************************print contents of node ********************************************

        printf("the contents of the node: \n");
        printf("data ID is: %c\n",newPtr->data);
        printf("the pointer element points to %p", newPtr->nextPtr);

    printf("the address of the node is: %p\n", newPtr);
        printf("the address of newPtr (the variable that stores the location of the node is): %p\n", &newPtr);

        return 0;
```

3 sample runs of ListNode.c

```
denis.manley@apollo:~/OS2/week3$ ./ListNode
enter a character: D
the contents of the node:
data ID is: D
the pointer element points to (nil)the address of the node is: 0x15d1010
the address of newPtr (the variable that stores the location of the node is): 0x7ffd87468678
denis.manley@apollo:~/OS2/week3$ ./ListNode
enter a character: A
the contents of the node:
data ID is: A
the pointer element points to (nil)the address of the node is: 0x17aa010
the address of newPtr (the variable that stores the location of the node is): 0x7ffc1e18ffa8
denis.manley@apollo:~/OS2/week3$ ./ListNode
enter a character: S
the contents of the node:
data ID is: S
the pointer element points to (nil)the address of the node is: 0x20b3010
the address of newPtr (the variable that stores the location of the node is): 0x7ffdee3edec8
denis.manley@apollo:~/OS2/week3$
```

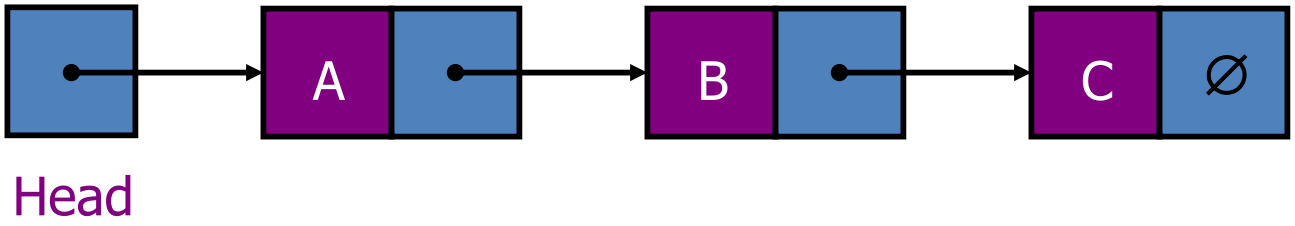
Class Question

- What is stored in newPtr

- In statement newPtr -> value = 8
  - what data type is stored in newPtr ?

- In statement newPtr -> nextPtr = NULL
  - What data type is nextPtr ?
- In statement newPtr -> data = NULL
  - What data type is data?

Linked Lists



- A *linked list* is a series of connected *nodes*
- Each node contains at least
  - A data field (any data type; e.g. int)
  - Node Pointer field: A pointer variable that stores the address of the the next node in the list
  - It has an address (&node)
- *Head*: "pointer" to the first node
- The last node points to `NULL`

Link List Overview

- Linked lists: a set of interconnected nodes
- Basic operations of linked lists
  - Insert, find, delete, print, edit….
- Variations of linked lists
  - Stacks : e.g. the program/thread stack: (L.I.F.O.)
  - Queues: used for algorithms for scheduling processes to run on a CPU: (F.I.F.O.)

A Simple Linked List “Node” structure

- Linked lists are made up of Nodes
- Declare `Node` structure for each node
  - `data`: `char`-type data in this example
  - `next`: a pointer to the next node in the list

- // c code to define listNode structure
- **struct listNode {**
- ** char data; // each listNode contains a character**
- ** struct listNode *nextPtr; // pointer to next node**
- **};** /typedef struct listNode ListNode; // synonym for struct listNode

Operations on Linked List

- sPtr is a “ListNode” double pointer variable
- It must be a double pointer in order to pass a pointer by reference
- currentPtr is a single pointer as we only pass the pointer by value
- In this code the pointer variable stores the address of the first node of the list.

```
// prototypes
void insert(ListNode* *sPtr, char value);
char delete(ListNode* *sPtr, char value);
void printList(ListNode* currentPtr);
void menu(void);

int main(void)
```

Code to initialise and call a insert fnt

```
// declare and initalise variables of link list

ListNode* startPtr = NULL; // initially there are no nodes
char item; // char entered by user
```

```
// call functions to add a node and print list
 insert(&startPtr, item); // insert item in list
 printList(startPtr);
```

The “double pointer” variable sPtr in the insert function is assigned the Address of startPtr: a variable that stores the address of the first node in the list. If the list is empty in contains a NULL value.

Home Work Question

- Write a C program that creates a new node which contains: an employee record (the data) and a pointer to a node.
- Assign any values to the data element of the node and NULL to the pointer part
- Display the contents of this node and the address of the node