CS 210 **November 10, 2016**
Day 20

Chapter 13 Stacks, Queues, and Linked Lists

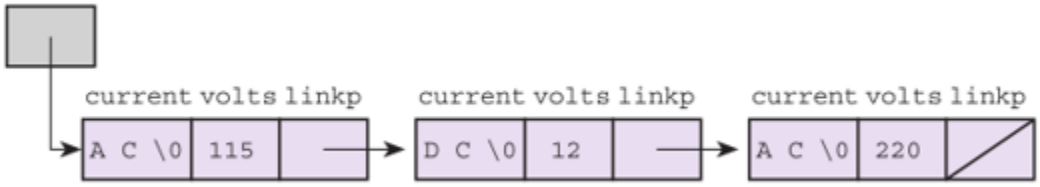
Linked Lists
A linked list is a data structure made up of nodes in which each node holds some data along with a pointer to the next node in the list.

Linked Lists

- linked list
  - a sequence of nodes in which each node but the last contains the address of the next node
- empty list
  - a list of no nodes
  - represented in C by the pointer NULL, whose value is zero
- list head
  - the first element in a linked list

In C we create a linked list using structs. For example

```
typedef struct node_s
   {char current[3];
    int volts;
    struct node_s *linkp;
   }node_t;
```



```c
/*We would like to write this typedef as:
typedef struct
   {char current[3];
    int volts;
    struct node_t *linkp;
   }node_t;
but when the compiler puts this together using
struct node_t *linkp the node_t has not yet been
defined.  We use node_s as a label as shown below to
get around this.
*/
typedef struct node_s
   {char current[3];
    int volts;
    struct node_s *linkp;
   }node_t;
#include<stdio.h>
#include<string.h>
int main()
{
   node_t n1, n2, n3;       //create 3 nodes
   node_t *n1p, *n2p, *n3p; //and 3 node pointers
   n1p = &n1;               //set up the pointers
   n2p = &n2;               // to point to the nodes
   n3p = &n3;
   n1p -> linkp = n2p;   //node 1 points to node 2
   n2p -> linkp = n3p;   //node 2 points to node 3
   n3p -> linkp = NULL;  //node 3 is last
   strcpy(n1p->current, "AC"); //put values in nodes
   n1p->volts = 120;
   strcpy(n2p->current, "DC");
   n2p->volts = 240;
   strcpy(n3p->current, "AC");
   n3p->volts = 440;
   //both of these access node 2 volts
   printf("%d\n", n2p->volts);
   printf("%d\n", n1p->linkp->volts);

   return 0;
}
```

Advantages of Linked Lists

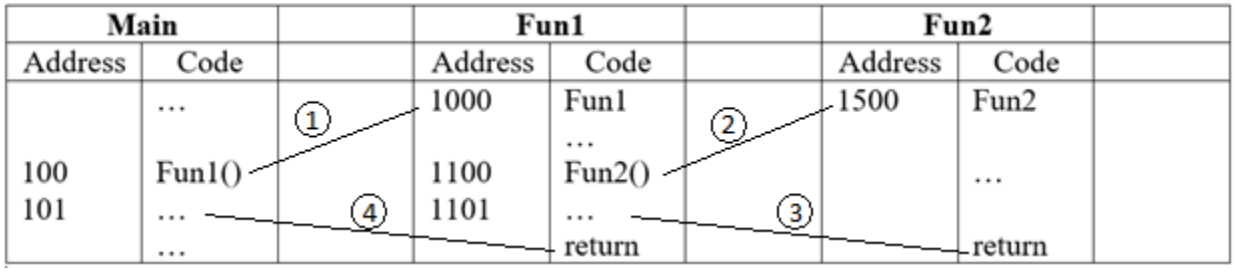
- It can be modified easily.
- The means of modifying a linked list works regardless of how many elements are in the list.
- It is easy to add or delete an element.

Stack
A stack is a first in last out buffer (or a last in first out buffer).

Nearly every computer CPU implements a stack structure in hardware with a stack pointer register on the CPU. The stack is used to save return addresses for function calls as well as pass parameters.

There are two main operators for a stack:
push – stores data on the stack and increases the stack pointer by 1
pop – decreases the stack pointer by 1 and returns the value on top of the stack.

A stack can be implements using an array (see pp. 400-401) or it can be done using a linked list (see pp. 727-731)



4→ 0 →	NULL	Top of stack
3→ 1→	101	
2→	1101	

The function calls at 1 and 2 push a return address 101 and 1101 on the stack. The return at 3 returns to 1101 since it is at the top of the stack. The stack pointer is decreased by 1 and the second return goes back to 101 which is at the top of the stack.

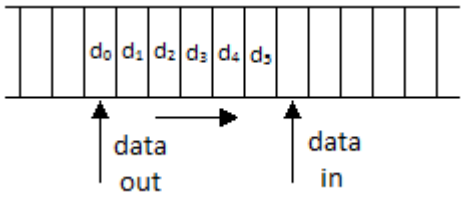
Queue
A queue is a first in first out buffer (or a last in last out buffer).

Queues are not as common or as useful as stacks but they can be used for modelling simulations or in serial data buffers.

A queue can be implemented using an array or it can be done with linked lists (see pp. 731-737). Like stacks, queues have two main operators.
Add – places a new item in the queue at the next free location and increases the input pointer.
Remove – takes an item out of the queue and increases the output pointer.

If the input and output pointers are equal the queue is empty.

A queue can be circular. This works as long as the total buffer size is larger than the amount of data to be stored at any one time.



The data item d_0 went into the queue first and d_5 went into the queue last.