IS 2610: Data Structures

Elementary Data Structures

Jan 12, 2004

Data Type

- Is a set of values and a collection of operations on those values.
  - Inbuilt data types
    - Int
    - Float
    - Character
  - New Data types
    - Define values to operate (arguments of a function)
    - Define operation (function definition)

Sample function definition

```
#include <stdio.h>
int lg(int);
main() {
  int i, N;
  for (i = 1, N = 10; i <= 6; i++, N *= 10)
      printf("%7d %2d %9d\n, N, lg(i), N*lg(N))
}
Int lg(int N){
  int i;
  for (i = 0;N > 0; i++, N/= 2);
  return i;
}
```

Data Structure

- Goal is to build data structures that allow us to handle collections of data
  - What operations need to be performed?
  - How to implement these operations?
- Simplest way to organize data in C
  - Arrays
  - Structures

Software Engineering practice

- Interface (header file)
  - Defines data structures
  - Declare functions to be used to manipulate the data structure
- Implementation (separate c file)
  - Of the functions declared in Interface
- Client (main application program)
  - Program that uses the functions declared in the Interface to work at a higher level of abstraction

Arrays

- Most fundamental data structure
  - Fixed collection of same-type data
  - Access is made by using an index
  - Contiguously stored
    - Direct correspondence with memory systems
    - Entire memory can be considered as an array of memory locations, with memory addresses corresponding to the array indices
- In C array definition
  - `int A1[N]; int A2[N][M]; char str[50];`
    - `A1[4]?       A1[i] = *(A1+i)?`
- Suppose you have to pass huge array as an argument?

Array

- Dynamic Memory Allocation

```
#define N 1000
main() {
   int i, a[N];
   …
}
```

```
#include <stdlib.h>
main(int argc, char* argv) {
   int i, N = atoi(argv[1]);
   int *a = malloc(N*sizeof(int);
   if (a==NULL) Insufficient memory
```

- Sieve of Eratosthenes

```
#define N 20
main() {
   int i, j, a[N];
   for (i = 1; i<N; i++) a[i]=1;
   for (i = 2; i<N; i++)
        if (a[i])
           for (j = i; i*j<N; j++) a[i*j] = 0;
   for (j = 2; j<N; j++)
        if (a[i]) printf (“%4d \n“, i);
}
```

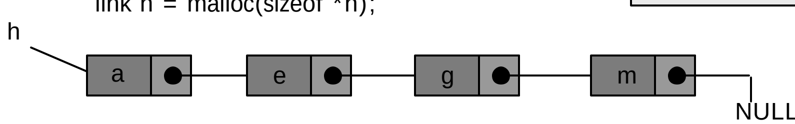
Finding primes
1 indicates prime
0 indicates nonprime

Linked List

- A set of items where each item is part of a node that also contains a link to a node
  - Self referent structures
  - Cyclic structures possible
  - C code

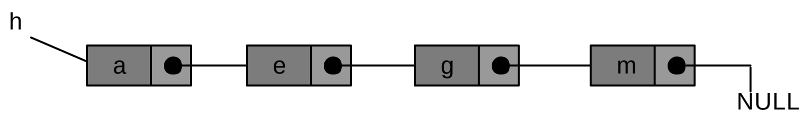
```
typedef struct node *link;
struct node {char ch; link next;}
link h = malloc(sizeof *h);
```

note that `h->next` denotes the 2nd node and `h->ch` denotes The value “a”



List traversal

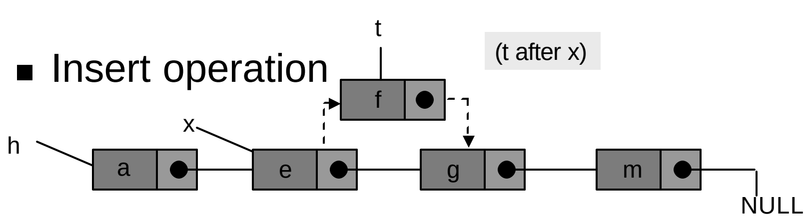
- Print each element of the list



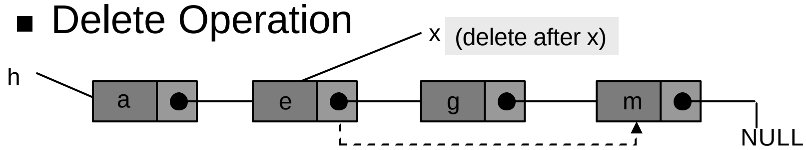
  - Use while loop
  - Use for loop
  - Inverting a list?

Linked Lists

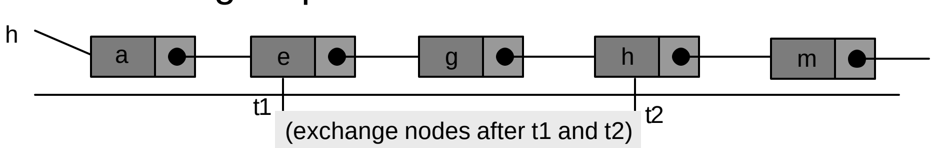
- Insert operation



- Delete Operation

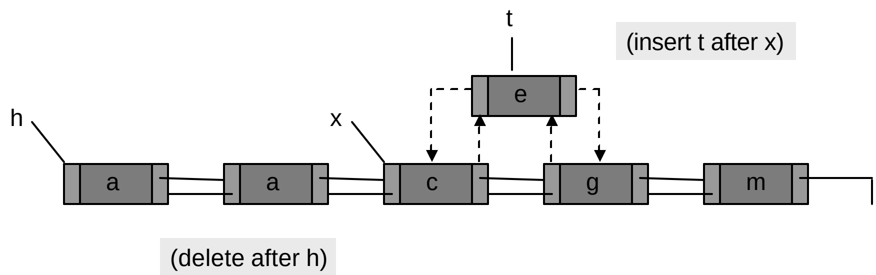


- Exchange Operation



Doubly Linked List

```
typedef struct node *link;
struct node {char ch; link prev; link next;}
link h = malloc(sizeof *h);
```



String

- Variable length array of characters
  - Has a starting point and a string-termination character (‘\0’) at the end
    - Array based implementation in C
    - Array of characters different from string – associated with *length*
      - String *length* may change
  - Many applications involve processing textual data
  - Computers provide access to bytes of memory that correspond directly to characters of strings

Common String functions

- strlen(a)

```
for(i=0; a[i] != 0; i++); return i;
```

- strcpy(a, b)

```
for(i=0; (a[i] = b[i]) != 0; i++);      while (*a++ = *b++);
```

- strcmp(a, b)

```
for(i=0; (a[i] == b[i]) != 0; i++);
        if (a[i] == 0) return 0;
return a[i] – b[i]
```

- strcat(a, b)

```
strcpy(a+strlen(a), b)
```

Abstraction

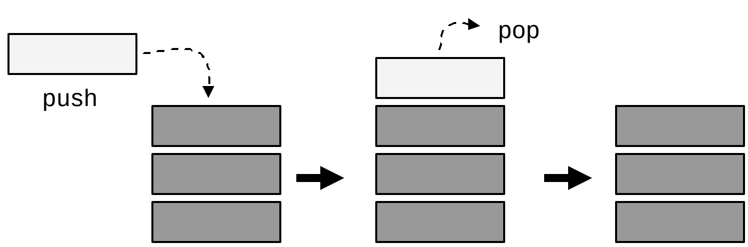
- Layers of abstraction
  - Abstract model of a bit with binary 0-1 values
  - Abstract model of a machine from from dynamic properties of the values of a certain set of bits
  - Abstract model of a programming language that we realize by controlling the machine with a machine –language program
  - Abstract notion of algorithm implemented in C
- Abstract data types
  - Develop abstract mechanisms for certain computational tasks at a higher level
- New layer of abstraction
  - Define objects we want to manipulate
    - Represent data in data structures
  - Define operations that we perform on them
    - Implement the algorithm

Abstract Data Type

- A data type that is access *only* through an *interface*
- Refer to a program that uses ADT as a *client* and program that specifies the data type as an *implementation*
  - Interface is opaque – clients cannot see implementation
  - Benefits of ADTs
    - Provide an effective mechanism for organizing large software systems
    - Provide ways to limit the size and complexity of interface between algorithms and associated data structures and programs that use the algorithms and data structures
    - ADTs interface defines a precise means of communication

Pushdown Stack ADT

- An ADT that comprises two basic operations: insert (push) a new item, and delete (pop) the item that was most recently inserted
  - Last in- first out (LIFO Queue)



Pushdown-stack ADT interfaces

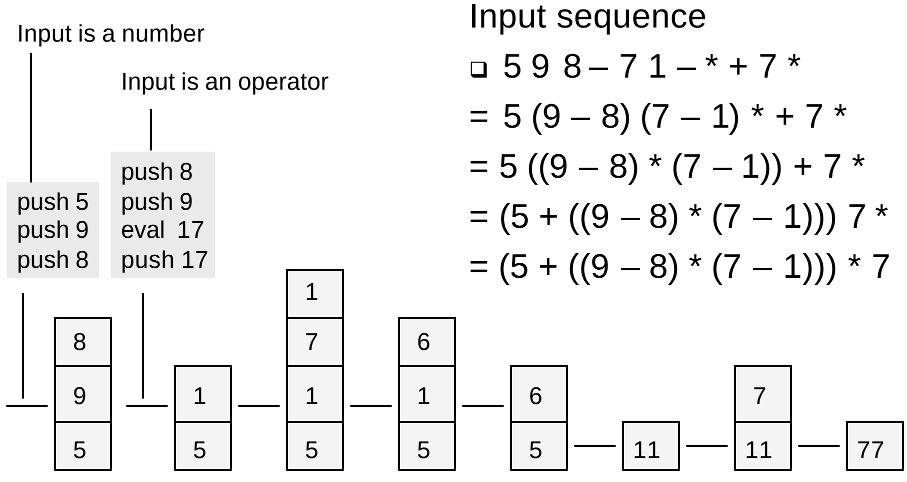
- Use in evaluation of arithmetic expression
  - Infix expression (customary way)
    - Operator comes between the operands
    - 4 + 5 is written as 4 5 +
  - Postfix expression
    - Operator comes after the operands
    - 4 + 5 is written as 4 5 +
  - Postfix expression
    - Operator comes after the operands
    - 4 + 5 is written as 4 5 +
- Interfaces: Client may use the four operations
  - store in STACK.h

```
void STACKinit(int);
  int STACKempty();
void STACKpush(Item);
Item STACKpop();
```

Postfix notation

- What is the postfix for the following infix
  - 6 + 5 * 9 ?
- What is the infix for the following postfix
  - 5 9 8 + 4 6 * * 7 + * ?
  - 5 9 8 – 7 1 - * + 7 * ?
- Note parentheses are not necessary in postfix

Postfix notation and Pushdown Stack



Input sequence

- 5 9 8 – 7 1 – * + 7 *

= 5 (9 – 8) (7 – 1) * + 7 *

= 5 ((9 – 8) * (7 – 1)) + 7 *

= (5 + ((9 – 8) * (7 – 1))) 7 *

= (5 + ((9 – 8) * (7 – 1))) * 7

Stack Implementation (Array)

```
static Item *s;
static int N;

void STACKinit(int maxN)
   {s = malloc(maxN*sizeof(Item)); N = 0;}

  int STACKempty()
   {return N==0;}

void STACKpush(Item item)
   {s[N++] = item;}

Item STACKpop()
   { return s[--N];}
```

```
void STACKinit(int);
  int STACKempty();
void STACKpush(Item);
Item STACKpop();
```

Stack Implementation (Linked-list)

- Assume auxiliary function

```
typedef struct STACKnode* link;
struct STACKnode {Item item; link next;}
static link head;
link NEW(Item item, link next;}
  { link x = malloc(sizeof *x);
    x->item = item; x->next = next;
    return x;
  }
```

```
void STACKinit(int);
  int STACKempty();
void STACKpush(Item);
Item STACKpop();
```

- Write the functions

First-In First Out Queues

- An ADT that comprises two basic operations: insert (put) a new item, and delete (get) the item that was least recently used

```
void QUEUEinit(int);
  int QUEUEempty();
void QUEUEput(Item);
Item QUEUEget();
```

```
typedef struct QUEUEnode* link;
struct QUEUEnode {Item item; link next;}
static link head;
link NEW(Item item, link next;}
  { link x = malloc(sizeof *x);
    x->item = item; x->next = next;
    return x;
  }
```

First-class ADT

- Clients use a single instance of STACK or QUEUE
- Only one object in a given program
- Could not declare variables or use it as an argument

- A first-class data type is one for which we can have potentially many different instances, and which can assign to variables whichcan declare to hold the instances

First-class data type – Complex numbers

- Complex numbers contains two parts
  - $(a + bi)$ where $i^2 = -1$;
  - $(a + bi)(c + di) = (ac - bd) + (ad + bc)i$

```
Typedef struct {float r; float i;} Complex;
Complex COMPLEXinit(float, float)
      float Re(float, float);
      float Im(float, float);
Complex COMPLEXmult(Complex, Complex)
```

```
Complex t, x, tx;
…
   t = COMPLEXInit(cos(r), sin(r))
   x = COMPLEXInit(?, ?)

  tx = COMPLEXInit(t, x)
```

First-class data type – Queues

```
typedef struct queue *Q;

void QUEUEdump(Q);
   Q QUEUEinit(int);
  int QUEUEempty(Q);
void QUEUEput(Q, Item);
Item QUEUEget(Q);
```

```
void QUEUEinit(int);
  int QUEUEempty();
void QUEUEput(Item);
Item QUEUEget();
```

```
Q queues[M];
for (i=0; i<M; i++)
   queues[i] = QUEUEinit(N);
.
   printf("%3d ", QUEUEget(queues[i]));
```

Recursion and Trees

- Recursive algorithm is one that solves a problem by solving one or more smaller instances of the same problem
  - Recursive function calls itself
  - Factorial?
  - Fibonacci numbers?

Euclid's method for finding the greatest Common divisor

```
int gcd(int m, int n){
        if (n==0) return m;
        return gcd(n, m%n);

}
```

Tree traversal (binary tree)

- Preorder
  - Visit a node,
  - Visit left subtree,
  - Visit right subtree
- Inorder
  - Visit left subtree,
  - Visit a node,
  - Visit right subtree
- Postorder
  - Visit left subtree,
  - Visit right subtree
  - Visit a node