CS1100
Introduction to Programming

Sorting Strings and Pointers

Madhu Mutyam
Department of Computer Science and Engineering
Indian Institute of Technology Madras

Lexicographic (Dictionary) Ordering

- Badri < Devendra
- Janak < Janaki
- Shiva < Shivendra
- Seeta < Sita
- Based on the ordering of characters
  - A < B ... < Y < Z < a < b < c < . . . < y < z

upper case before lower case

Lexicographic Ordering

- What about blanks?
  - “Bill Clinton” < “Bill Gates”
  - “Ram Subramanian” < “Ram Subramanium”
  - “Ram Subramanian” < “Rama Awasthi”
- In ASCII the blank (code = 32) comes before all other characters. The above cases are taken care of automatically.
- Exercise: Look up ASCII codes on the web.

Lexicographic Ordering

- What if two names are identical?
- There is a danger that the character arrays may contain some unknown values beyond ‘\0’
- Solutions
  - One could begin by initializing the arrays to blanks before we begin.
  - One could explicitly look for the null character ‘\0’
  - When the two names are equal it may not matter if either one is reported before the other. Though in stable sorting there is a requirement that equal elements should remain in the original order.

Comparing Strings (char Arrays)

- Given two strings *A*[*i*][] and *A*[*j*][] of length *n*, return the index of the string that comes earlier in the lexicographic order

```
int strCompare(char A[ ][MAX_SIZE], int i, int j, int
MAX_SIZE){
        int k=0;
        while ((A[i][k] == A[j][k]) && k<MAX_SIZE) k++;
        if (A[i][k] == ‘\0’) return i;
        if (A[j][k] == ‘\0’) return j;
        if (A[i][k] < A[j][k]) return i;
        else return j;
}
```

Skip common characters if any

If one string is *prefix* of the other return that

Built-in String Comparison

- ***#include*** <string.h>
- ***int*** *strcmp*(***const char*** *s1, ***const char*** *s2);
- ***int*** *strncmp*(***const char*** *s1, ***const char*** *s2, size_t n);
- ***int*** *strcmp*(char*, char*) – compares two strings (char arrays)
- The return values are:
  - 0 – If both strings are equal
  - 1 – If first string is lexicographically greater than second
  - -1 – If second string is lexicographically greater than first

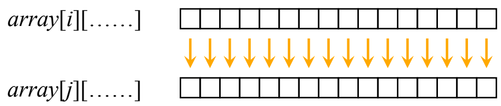
Pointers - address of char array – we will look at them later

Other Built-in String Functions

- ***char*** * *strcat*(***char*** * dest, ***char*** * src)
  - *Strcat* combines two strings and returns a pointer to the destination string. In order for this function to work (and not seg fault), you must have enough room in the destination for both strings.
- ***char*** * *strcpy*(***char*** * dest, ***char*** * src)
  - *Strcpy* copies one string to another. The destination must be large enough to accept the contents of the source string.
- ***int*** *strlen*(***const char*** * s)
  - *Strlen* returns the length of a string, excluding '\0'

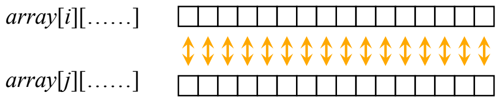
ArrayCopy

Copies content of i^{th} row of array into the j^{th} row



```
void arrayCopy (char array[ ][MAX_SIZE], int i, int j,
int MAX_SIZE) {
        int k;
        for (k =0; k < MAX_SIZE; k++)
                array[j][k] = array[i][k];
}
```

ArraySwap



```
void arraySwap (char array[ ][MAX_SIZE], int i, int j,
   int MAX_SIZE){
     for (k =0; k < MAX_SIZE; k++)
           swap(array, i, j, k);
}
```

Note: We exchange the entire arrays. If we knew the length of the longer string, we could have a different end condition.

Sorting String Arrays

- Modify *InsertionSort* to sort array *names*[ ] of names
- Use *strCompare* to compare names
- Use *arrayCopy* to move names
- In the exercise where *names*[ ] and *marks*[ ] have to be sorted in concert, modify the sorting algorithm to
  - compare in one array
    - *names*[ ] for alphabetic order
    - *marks*[ ] for decreasing marks order
  - move elements of both

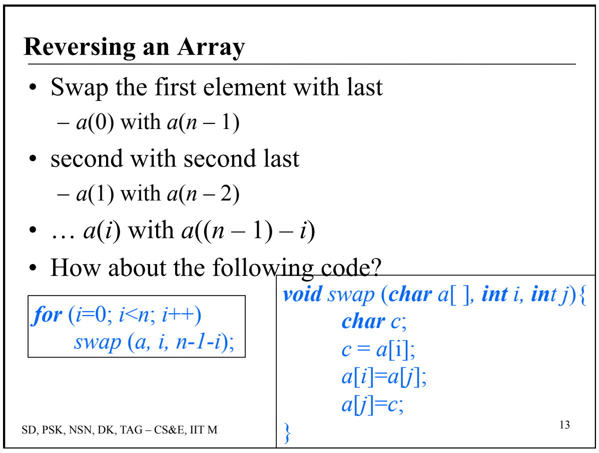
Compact structures to hold both to be explored later

Printing a Reversed String

```
main( ) {
int i = 4;
char c;
do{
        c = "hello"[i];
        printf("%c",c);
        i --;
    }while(i >= 0);
printf("\n");
}
```

Palindromes

- Strings/sequences that read the same left to right or right to left
- string == reversed string
  - malayalam
  - god live evil dog
  - able was I ere I saw elba
  - don't nod
  - never odd or even
- notice that we ignore blanks (4, 5) and other characters (4)
  - preprocess the string to remove them

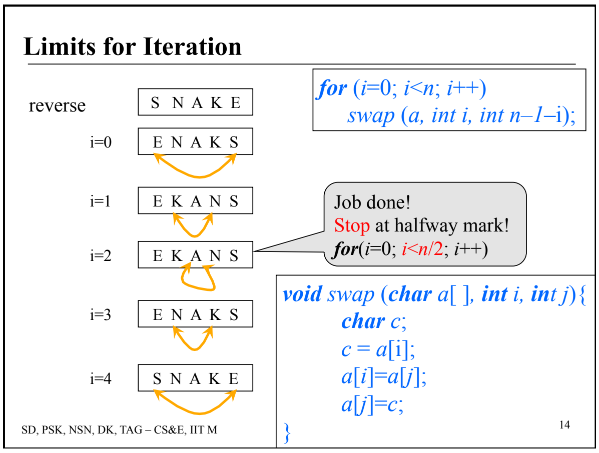


Reversing an Array

- Swap the first element with last
  - $a(0)$ with $a(n-1)$
- second with second last
  - $a(1)$ with $a(n-2)$
- … $a(i)$ with $a((n-1)-i)$
- How about the following code?

```
for (i=0; i<n; i++)
      swap (a, i, n-1-i);
```

```
void swap (char a[ ], int i, int j){
        char c;
        c = a[i];
        a[i]=a[j];
        a[j]=c;
}
```



Limits for Iteration

```
for (i=0; i<n; i++)
    swap (a, int i, int n–1–i);
```

reverse S N A K E

i=0 E N A K S

i=1 E K A N S

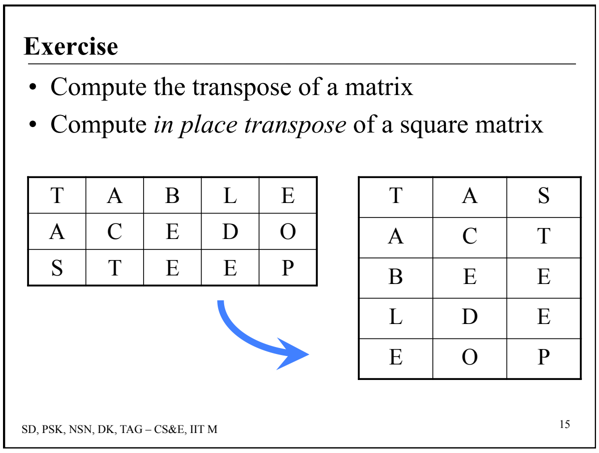
i=2 E K A N S

i=3 E N A K S

i=4 S N A K E

Job done!
Stop at halfway mark!
for(i=0; i<n/2; i++)

```
void swap (char a[ ], int i, int j){
        char c;
        c = a[i];
        a[i]=a[j];
        a[j]=c;
}
```

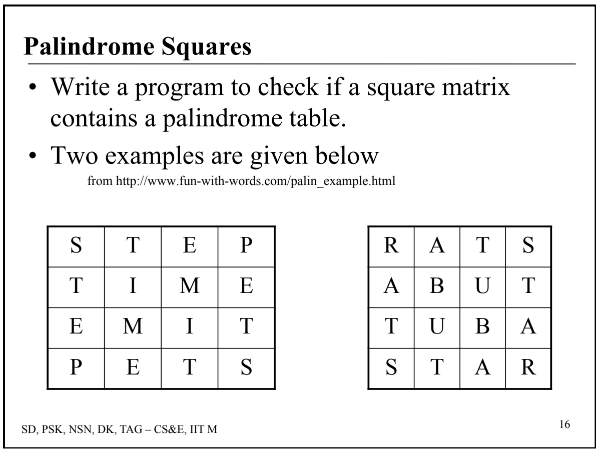


Exercise

- Compute the transpose of a matrix
- Compute *in place transpose* of a square matrix

T	A	B	L	E
A	C	E	D	O
S	T	E	E	P

T	A	S
A	C	T
B	E	E
L	D	E
E	O	P



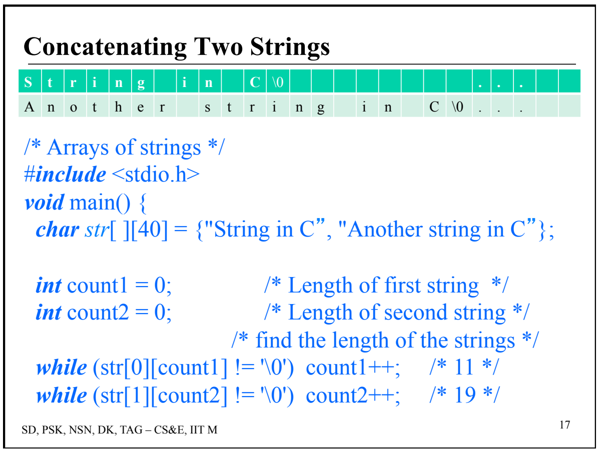
Palindrome Squares

- Write a program to check if a square matrix contains a palindrome table.
- Two examples are given below

from http://www.fun-with-words.com/palin_example.html

S	T	E	P
T	I	M	E
E	M	I	T
P	E	T	S

R	A	T	S
A	B	U	T
T	U	B	A
S	T	A	R

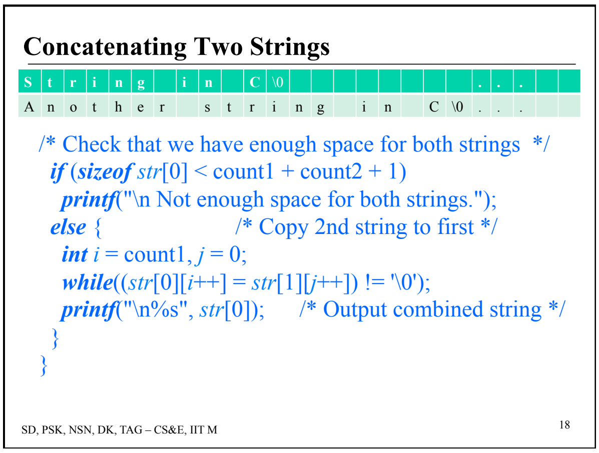


Concatenating Two Strings

S	t	r	i	n	g		i	n		C	\0									.	.	.		
A	n	o	t	h	e	r		s	t	r	i	n	g		i	n		C	\0	.	.	.		

```
/* Arrays of strings */
#include <stdio.h>
void main() {
 char str[ ][40] = {"String in C”, "Another string in C”};

 int count1 = 0;             /* Length of first string  */
 int count2 = 0;             /* Length of second string */
                          /* find the length of the strings */
 while (str[0][count1] != '\0')  count1++;     /* 11 */
 while (str[1][count2] != '\0')  count2++;     /* 19 */
```



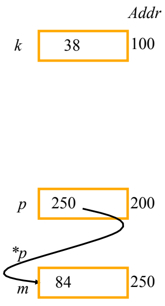
Concatenating Two Strings

S	t	r	i	n	g		i	n		C	\0									.	.	.		
A	n	o	t	h	e	r		s	t	r	i	n	g		i	n		C	\0	.	.	.		

```
 /* Check that we have enough space for both strings  */
  if (sizeof str[0] < count1 + count2 + 1)
   printf("\n Not enough space for both strings.");
  else {                  /* Copy 2nd string to first */
   int i = count1, j = 0;
   while((str[0][i++] = str[1][j++]) != '\0');
   printf("\n%s", str[0]);     /* Output combined string */
  }
 }
```

What is a Pointer?

- *Recap*: a variable ***int*** *k*
  - Names a memory location that can hold one value at a time
  - Memory is allocated statically at compile time
  - One name ⇔ one value
- A pointer variable ***int*** **p*
  - Contains the address of a memory location that contains the actual value
  - Memory can be allocated at runtime
  - One name ⇔ many values



l-value and *r*-value

- Given a variable *k*
  - Its *l*-value refers to the address of the memory location
  - *l*-value is used on the left side of an assignment
    - Ex. *k* = expression
  - Its *r*-value refers to the value stored in the memory location
  - *r*-value is used in the right hand side of an assignment
    - Ex. *var* = *k* + …
- Pointers allow one to manipulate the *l*-value!

Pointer Variables

- Pointer variables are variables that store the address of a memory location
- Memory required by a pointer variable depends upon the size of the memory in the machine
  - one byte could address a memory of 256 locations
  - two bytes can address a memory of 64K locations
  - four bytes can address a memory of 4G locations
  - modern machines have RAM of 1GB or more…
- The task of allocating this memory is best left to the system

Declaring Pointers

- Pointer variable – precede its name with an asterisk
- Pointer type - the type of data stored at the address
  - For example, ***int*** **p;*
  - *p* is the name of the variable. The '*' informs the compiler that *p* is a pointer variable
  - The ***int*** says that *p* is used to point to an integer value

Ted Jenson's tutorial on pointers
http://pweb.netcom.com/~tjensen/ptr/cpoint.htm

Contents of Pointer Variables

- In ANSI C, if a pointer is declared outside any function, it is initialized to a *null* pointer
  - For example,

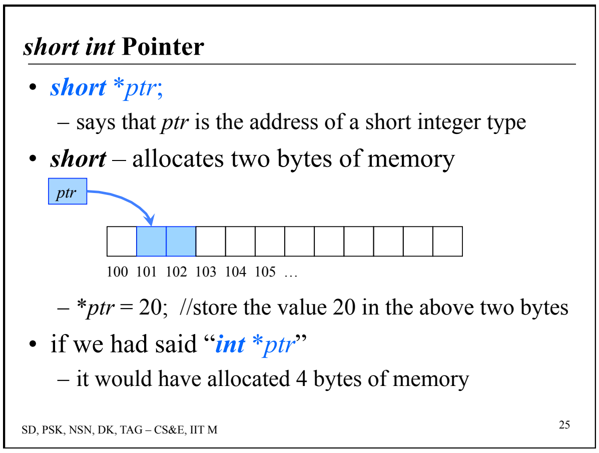
```
int k;
int *p;
p = &k;              //assigns the address of int k to p
if (p == NULL)       //tests for a null pointer
    p = malloc(sizeof(int));  //dynamic allocation,
                              //creates an anonymous int
                              // in memory at runtime
```

Dereferencing Operator

- The asterisk symbol is the "dereferencing operator" and it is used as follows

  **ptr = 7;*

  - Will copy 7 to the address pointed to by *ptr*
  - Thus if *ptr* "points to" (contains the address of) *k*, the above statement will set the value of *k* to 7
- Using '*' is a way of referring to the value in the location which *ptr* is pointing to, but not the value of the pointer itself
  - ***printf***("%d\n",**ptr*); --- prints the number 7

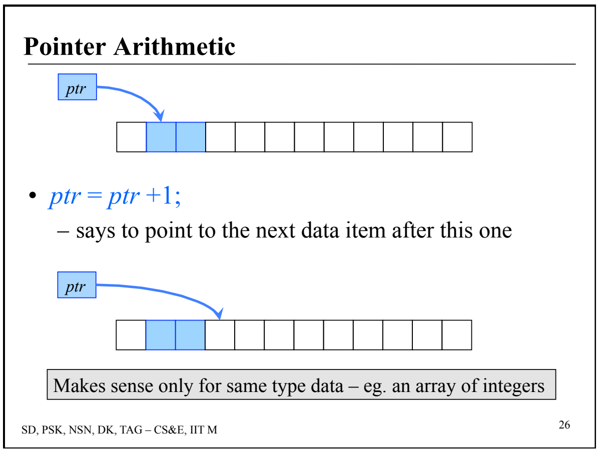


short int Pointer

- ***short*** ******ptr*;
  - says that *ptr* is the address of a short integer type
- ***short*** – allocates two bytes of memory

ptr
100 101 102 103 104 105 …

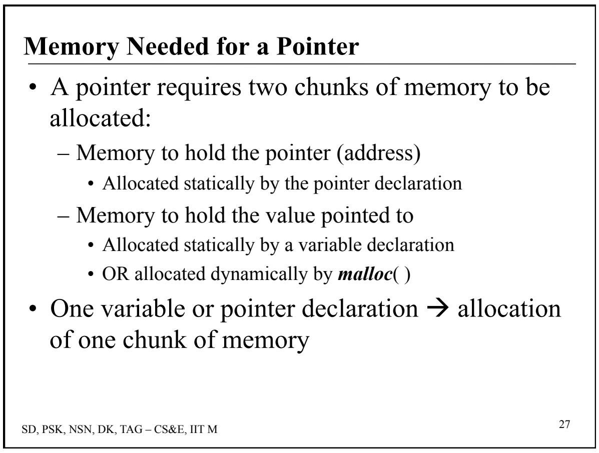
  - **ptr* = 20; //store the value 20 in the above two bytes
- if we had said "***int*** **ptr*"
  - it would have allocated 4 bytes of memory



Pointer Arithmetic

- *ptr* = *ptr* +1;
  - says to point to the next data item after this one

Makes sense only for same type data – eg. an array of integers



Memory Needed for a Pointer

- A pointer requires two chunks of memory to be allocated:
  - Memory to hold the pointer (address)
    - Allocated statically by the pointer declaration
  - Memory to hold the value pointed to
    - Allocated statically by a variable declaration
    - OR allocated dynamically by ***malloc***( )
- One variable or pointer declaration → allocation of one chunk of memory