UNIVERSITY OF MASSACHUSETTS
DARTMOUTH

ECE160: Foundations of Computer Engineering I

Lecture #27 – Exam #3 Review

Instructor: Dr. Liudong Xing
SENG-213C, lxing@umassd.edu
ECE Dept.

Exam #3

- Time: **9:00am ~ 10:30am, Friday, April 21**
- Please arrive at the class on time; no make up time will be given for late arrivals.
- Form:
  - Open book, open notes
  - Calculators are NOT allowed
  - Visual Studio is NOT allowed
  - Chat GPT is NOT allowed
- Preparation:
  - Lecture notes #20 - #26 prepared by Dr. Xing (available on class website)
  - Lab #9 - #11

Exam#1: Lectures #2 - #10

- Number systems (L#2)
- Introduction to C programming (L#3)
- Data types and variables (L#4)
- Constants (L#5)
- Formatted input/output (L#6 & 7)
- Expressions (L#8 & 9)
- Two-way selection: if…else (L#10)

Exam#2: Lectures #12 - #18

- Multi-way selection: switch and if-else-if (L#12)
- Loops (L#13)
- Functions (L#14 ~ 17)
- Files (L#18)

Exam#3: Lectures #20 - #26

- Files (L#20)
- Arrays (L#21-23)
- Array sorting (L#24)
- Strings (L#25)
- Pointers (L#26)

Files (L#20)

- **How to declare a file pointer**

```
FILE *file_pointer;
```

- **How to open a file**

```
file_pointer = fopen("file_name", "mode");
```

  - To create a link between a file stored in actual disk and a file pointer

- **How to read data from a file**

```
fscanf(file_pointer, "format_string", address_list);
```

- **How to write data to a file**

```
fprintf(file_pointer, "format_string", data_list);
```

- **How to close a file**

```
int fclose(FILE *file_pointer);
```

Arrays (L#21-23)

- An array is a fixed-size, sequenced collection of elements of the same data type.
- Index of the first element is 0!
- The array elements are stored in contiguous and increasing memory locations.
- Before use, an array has to be defined and declared.
  - Reserve memory space for the elements in the array!

Array Declaration and Definition (Syntax)

```
element_type   array_name [number _of_elements];
```

- element_type: type of the array's elements, e.g., int, float **(cannot be void type!)**
- array_name: the name of the array
- number_of_elements: the length/size of the array
  - Must be an integer or integer expression greater than 0
  - Can be defined explicitly:

    ```
    int a[100];
    ```
  - Use a constant defined in a preprocessor directive (L#5):

    ```
    #define N 100
    int a[N];
    ```
  - Can be an integer expression:

    ```
    int a[N+30];
    ```

Array Initialization (3 ways)

- At the definition time

```
int myarray[5]={1,2,10,15,0};
int myarray[] = {1,2,10,15,0};
```

- Inputting values form the keyboard

```
int myarray[5];
for(int i=0; i< 5; i++)
{       scanf("%d", &myarray[i]); }
```

- Assigning values

```
int myarray[5];
for(int i=0; i< 5; i++)
{          myarray[i]=i*2+1;  }
```

Exchanging Values of Two Array Elements

- Solution: to use a temporary variable

```
int num_array[6];

temp = num_array[2];
num_array[2] = num_array[4];
num_array[4] = temp;
```

Multi-Dimensional Arrays

- Before use, a multi-D array has to be defined and declared

```
element_type   array_name [num _of_rows][num_of-columns];
```

```
int myarray[3][2];
```

- The first index for both row and column is ZERO!
- Three ways to initialize a multi-D array
  - At the definition time
  - Inputting values form the keyboard using nested loops
  - Assigning values using nested loops

Arrays and Functions

- When passing an individual array element, treat the single array element like a simple variable!
  - Pass by values: pass the values of the element without having it changed in the function
  - Pass by reference: change the value of the array element in the function
- When passing the whole array to a function
  - In the calling function, use the array name as the input parameter passed to the called function
  - In the called function, specifically, the function header, and function declaration, declare the parameter as an array

Passing the Entire Array (Example)

```
#include "stdio.h “

void add(int arr[]);

void main(void)
{
    int myarray[5]= {1,2,9,3,6};
   add(myarray); /* Pass the whole array to a function */
    printf(“The value of myarray[2] is: %d\n",myarray[2]);
}

void add(int arr[])
{
    arr[2] = arr[2] + 100;
}
```

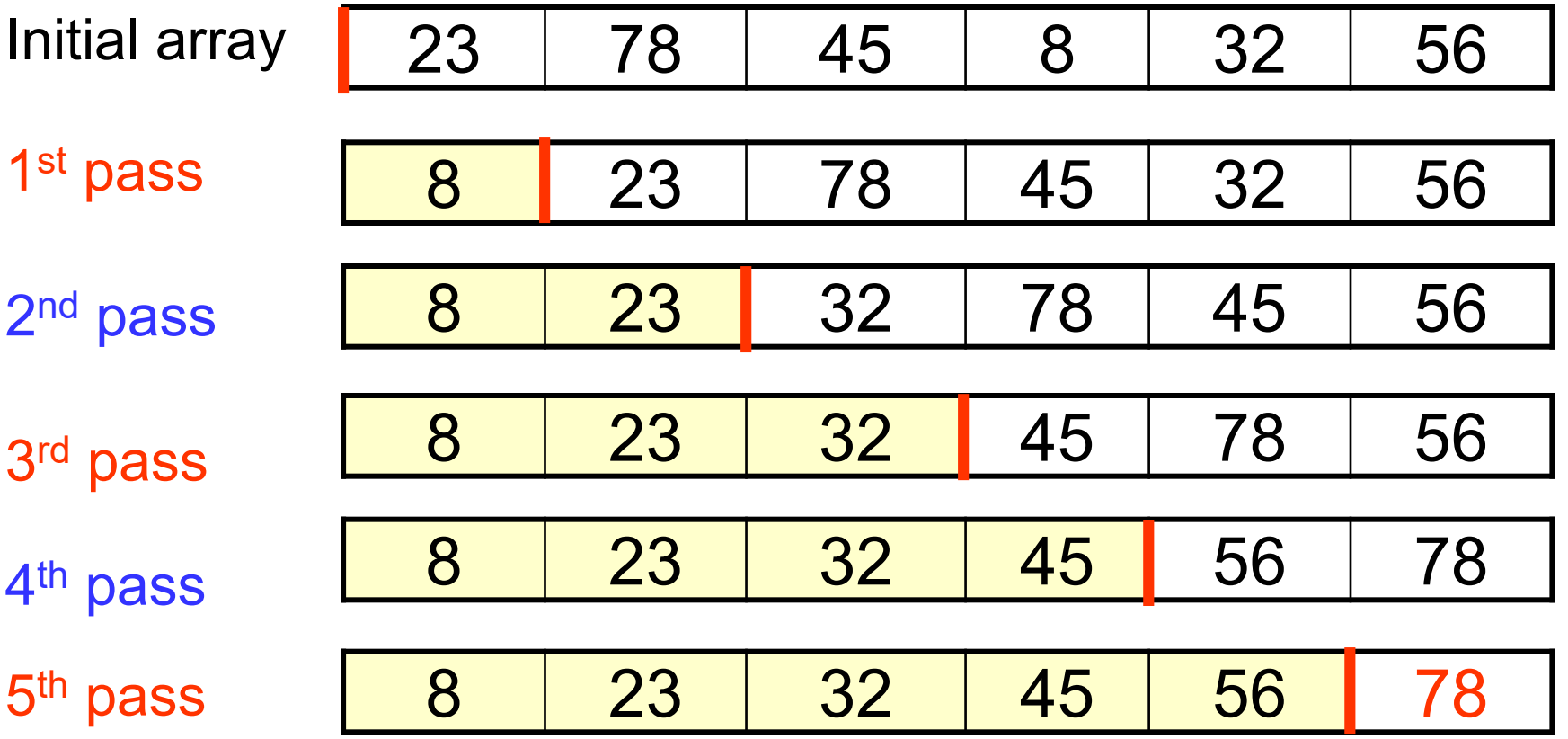
A variable with brackets [] in function prototype and header indicate the parameter is an array!

Exam#3: Lectures #20 - #26

- ✓ Files (L#20)
- ✓ Arrays (L#21-23)
- Array sorting (L#24)
  - Sorting problem is a problem to sort/arrange a sequence of numbers into non-decreasing or non-increasing order
  - Bubble sort and selection sort
- Strings (L#25)
- Pointers (L#26)

Bubble Sort (Example)

Bubble sort works by repeatedly comparing adjacent elements and swapping adjacent elements that are out of order

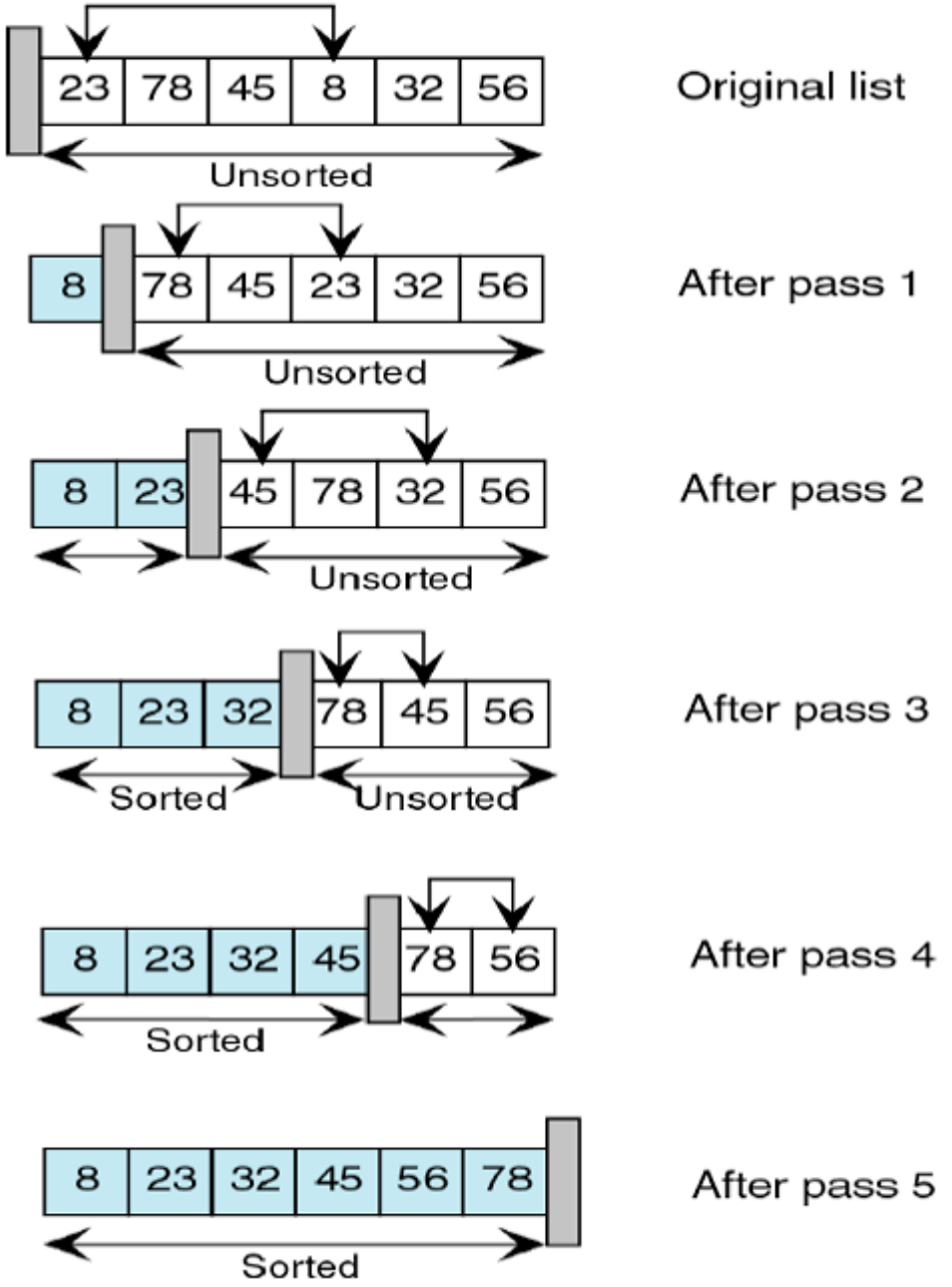


A Function to Implement Bubble Sort

```
void bubbleSort(int list[], int last)  /*last = (array size -1) */
{
     int current, walker, temp;
     for(current=0; current < last; current++)
            for(walker=last; walker > current; walker--)
                        if(list[walker] < list[walker-1])
                        {
                           temp = list[walker];
                           list[walker] = list[walker-1];
                           list[walker-1] = temp;
                        }
}
```

Selection Sort (Example)

Selection sort works by repeatedly selecting the smallest/largest remaining element



A Function to Implement Selection Sort

```
void selectionSort(int list[], int last)
{
      int current, walker, temp, min;
      for(current=0; current < last; current++)
      {
            min=current;

            for(walker=current+1; walker <=last; walker++)
                  if(list[walker] < list[min])
                        min=walker;

            /*smallest selected: exchange with current element*/
              temp = list[current];
              list[current] = list[min];
              list[min] = temp;
      }
}
```

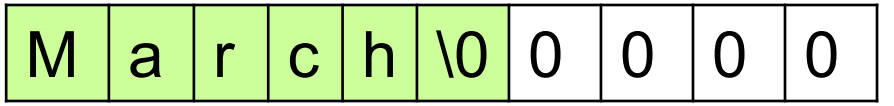
Exam#3: Lectures #20 - #26

- ✓ Files (L#20)
- ✓ Arrays (L#21-23)
- ✓ Array sorting (L#24)
- Strings (L#25)
- Pointers (L#26)

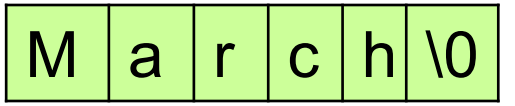
Strings

- In C, a string is a variable-length array that is DELIMITED BY THE NULL CHARACTER (\0).
- Four ways to initialize a string

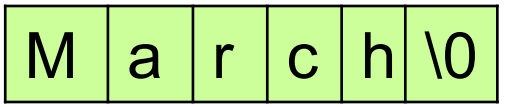
char month[10] = “March”;



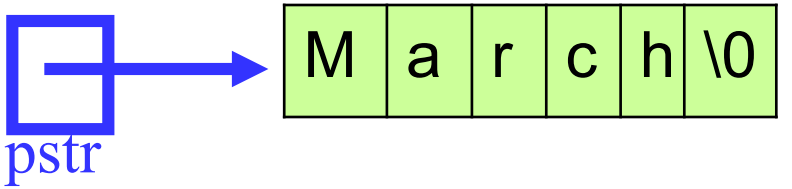
char month[] = “March”;



char month[6] = {‘M’,‘a’, ‘r’,‘c’, ‘h’, ‘\0’};



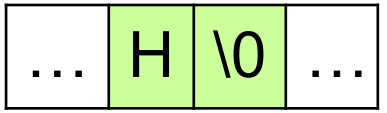
char *pstr=“March”;



String Literals *vs* Character Literals

- A string literal is a sequence of characters enclosed in double quotes
- Stored in an array of characters terminated by the null character '\0'
- Example:

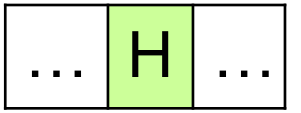
"H"



2 bytes

- A character literal is enclosed in single quotes!
- Example:

'H'



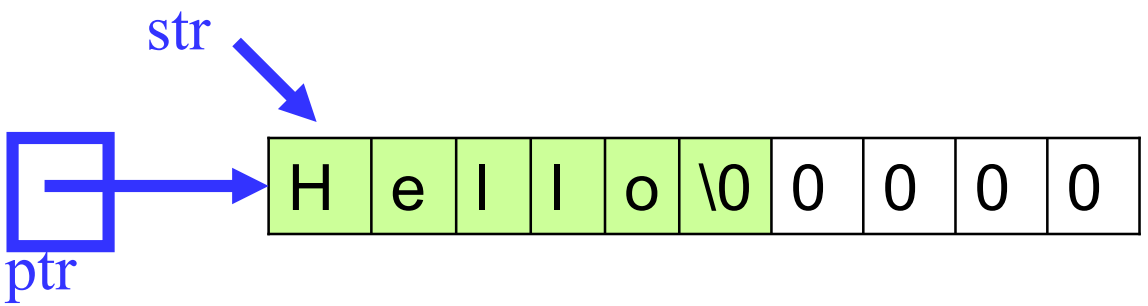
1 byte

Referencing String Literals

Array name indicates the address of the first element of the array

String itself is a pointer to the first element/character of the string

```
char str[10] = "Hello";
char *ptr;
ptr=str;
```



str[0]=ptr[0]="Hello"[0]='H'
str[3]=ptr[3]="Hello"[3]='l'

Pointers

- A pointer variable can be declared using * in the declaration statement
- A pointer to (or the address of) a variable can be obtained using &
- Pointers provide us a way to work with addresses symbolically.

```
#include "stdio.h"
void main(void)
{
        int x=3;
        int *p= &x;
        printf("%d\n",x);
        printf("%d\n",*p);
        printf("%d\n",p);
}
```

Output:

3

3

1244884

Using Pointers (L#26)

- Using pointers
  - to increment a number
  - to test for equality using pointers
  - to add two numbers
- Use multiple pointers for one variable
- Use pointers that point to other pointers

Ways to increment a number

- Assume

```
int a=0;
int *p=&a;
```

we need to add 1 to a:

```
a++;
++a;
a=a+1;
*p=*p+1;
(*p)++;
++(*p);
```

Pointers to Pointers

```
#include "stdio.h"

void main(void)
{
    int x = 10;
    int *p;         /*p is a pointer to an integer*/
    p = &x;
    int **q;        /*q is a pointer to an integer pointer*/
    q= &p;
    printf("%d\n", x);
    printf("%d\n", *p);
    printf("%d\n", **q);
}
```

```
10
10
10
```

To refer to x using q, you have to dereference it twice to get to the integer x because there are two levels of indirection / pointers involved!

Pointers and Functions

- Pointers can be arguments to a function (pass by reference)

```
#include “stdio.h”
void swap(int x, int y);
void main(void)
{
    int a=3;
    int b=7;
    swap(a,b);
    printf(“%d %d\n”, a, b);
}
void swap(int x, int y)
{
    int temp;
    temp=x;
    x=y;
    y=temp;
}
```

!Not Working!

```
#include “stdio.h”
void swap(int *x, int *y);
void main(void)
{
    int a=3;
    int b=7;
    swap(&a,&b);
    printf(“%d %d\n”, a, b);
}
void swap(int *x, int *y)
{
    int temp;
    temp=*x;
    *x=*y;
    *y=temp;
}
```

Functions returning pointers

- Pointers can be returned from a function
- When you return a pointer, it must point to data in the calling function
- It's an error to return a pointer to a local variable in the called function because when the function terminates, its memory can be used by other parts of the program!

An Example (Correct Program)
To determine the larger of two numbers

```
#include "stdafx.h"
int *max(int *pa, int *pb);
void main(void)
{
    int a;
    int b;
    int *pmax=NULL;

    printf("Enter first number:\n");
    scanf("%d",&a);
    printf("Enter second number:\n");
    scanf("%d",&b);

    pmax=max(&a, &b);
    printf("The maximum is %d\n",
    *pmax);
 }
```

```
int* max(int *pa, int *pb)
{
    if (*pa > *pb)
        return pa;
    else
        return pb;
}
```

Exam #3

- Time: **9:00am ~ 10:30am, Friday, April 21**
- Please arrive at the class on time; no make up time will be given for late arrivals.
- Form:
  - Open book, open notes
  - Calculators are NOT allowed
  - Visual Studio is NOT allowed
  - Chat GPT is NOT allowed
- Preparation:
  - Lecture notes #20 - #26 prepared by Dr. Xing (available on class website)
  - Lab #9 - #11

Good Luck!