Introduction to Data Structures

July 25, 2005

Review: Using Pointers

- When are pointers useful?
  - Pass-by-reference in functions
    - Changing data structure values in functions
    - Not making copies of large data structures, e.g., arrays

Review: Strings

- strings == char array == char pointer?
  - **String***: only* when the char array or pointer ends in '\0'
  - But, char arrays and char pointers equivalent when passing as parameters to functions
    - Examples:
      - void modArray(int array[]);
      - void modArray(int *array);

Choosing to use pointers or arrays in functions

- Pointers
  - Can't use until initialized
    - String literal
    - Dynamically allocated
  - Example:

    char *string = NULL;

    string[5] = ?

    Likely will cause a seg. fault

    Can't assign to "each" character --> *string* is just a memory location
- Arrays
  - Can do anything
  - Example:

    char string[10];

    string[5] = ?

    Garbage but will not cause a seg fault

    Could assign to each character in the array

Choosing to use pointers or arrays in functions

- Pointers
  - Can't use until initialized
    - String literal
    - Dynamically allocated
  - Can't modify the string constant
    - Example:

      char *string = "constant";
- Arrays
  - Can do anything
  - Example:

    char string[10];

    string[5] = ?

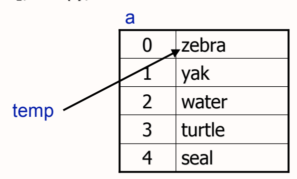
    Garbage but will not cause a seg fault
  - Could assign to each character in the array

  If you're having trouble, go with arrays

Swapping Strings

```
void swap( char a[][STRLENGTH], int index1, int index2 ) {
  char *temp = a[index1];
  strcpy(a[index1], a[index2]);
  strcpy(a[index2], temp);
}
```

index1 = 0
Index2 = 4

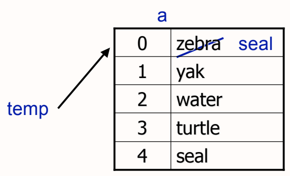


swapstrings.c
sortstrings.c

Swapping Strings

```
void swap( char a[][STRLENGTH], int index1, int index2 ) {
  char *temp = a[index1];
  strcpy(a[index1], a[index2]);
  strcpy(a[index2], temp);
}
```

index1 = 0
Index2 = 4

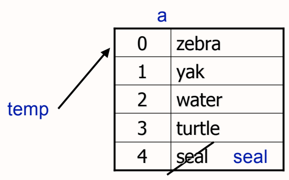


swapstrings.c
sortstrings.c

Swapping Strings

```
void swap( char a[][STRLENGTH], int index1, int index2 ) {
  char *temp = a[index1];
  strcpy(a[index1], a[index2]);
  strcpy(a[index2], temp);
}
```

index1 = 0
Index2 = 4



Write the *correct* code to swap strings

swapstrings.c
sortstrings.c

Quiz

More Practice Using Strings

- Printing a string backwards
  - backwardstring.c
- Removing the spaces from a word
  - remspaces.c

Returning pointers from functions

- A function can return a pointer
- Syntax:
  - data_type *functionName(*param_list*);

returnptr.c

Giving Executables Different Names

- Could copy a.out into a different name
  - cp a.out myprog
- Or give cc a command line argument
  - cc -o myprog program.c
  - -o: means the output file is the next string

Linking Data

Team

Baltimore	Boston	New York	Toronto

Nickname

Orioles	Red Sox	Yankees	Blue Jays

Wins

50	54	51	49

- What if need to reorganize arrays by number of wins?
  - Would have to swap associated team and nickname arrays too

Structs: Organizing Data

- Groups together data of different types
- Can be used like any other datatype
- Create a new struct (the struct *definition*):

Struct name, tend to start with a capital letter

```
struct MLBTeam {
    char name[40];
    char nickname[40];
    int wins;
};
```

Struct members

Using Structs

- Declare one variable of type MLBTeam:
  - struct MLBTeam orioles;
- Declare an array of MLBTeams:
  - struct MLBTeam americanLeague[14];
- Access struct members using the "dot" operator
  - orioles.wins = 50;
  - strcpy(orioles.name, "Baltimore");
- Note: you'll always need "struct" before the name of the structure type

Using Structs in Functions

- Members' use in functions:
  - void incrementWins(int *wins);
  - incrementWins(&orioles.wins);
- Whole structs in functions:
  - double calculateWinPercentage(struct MLBTeam team);
  - double winpercentage = calculateWinPercentage(orioles);
- Returning structs from functions:
  - struct MLBTeam findLeader(struct MLBTeam league[], int size);

Practice with Structs

- Major League Baseball
  - Using sizeof on a struct

baseball.c

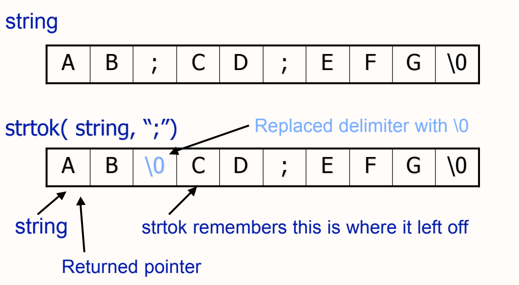
Using strtok

- char *strtok(char *string, char *delimiter);
  - returns a pointer to a char
- Each call to strtok replaces the delimiter with \0
  - First use: strtok(string, ";")
  - Subsequent uses: strtok(NULL, ";")
    - Remembers that string was originally passed to strtok

string

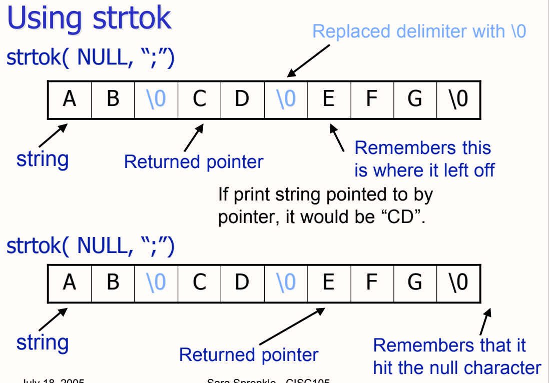
A	B	;	C	D	;	E	F	G	\0

Using strtok



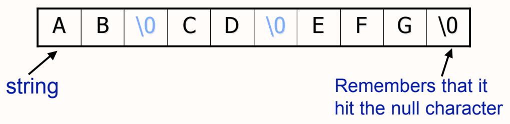
The string pointed to by returned pointer is "AB".

Using strtok



If print string pointed to by pointer, it would be "CD".

Using strtok



If call strtok(NULL, ";") again, it will return NULL, so you know you reached the end of the string.

string is just "AB" because the \0 has been added

usingstrtok.c

Pointers to Structures

- When using pointers to structs
  - access members using ->

baseball.c

Dynamic Memory Allocation

- Efficient use of memory
  - Plan for the worst-case (allocate lots of memory)
  - Our fixed-size arrays may be allocating more memory than necessary
- Instead of allocating all memory at once, wait until know how much is needed
- Good when you want data to stick around beyond the scope of a function
  - Example: function returns a char*

malloc

- Reserves memory during program execution
  - malloc(number_of_bytes)
    - Initializes memory to 0
  - If cannot reserve that amount of memory, returns NULL
- Cast the result of malloc to the appropriate datatype
  - Examples:

```
char *stringarray; int *intarray[10];
stringarray = (char *) malloc( 5*sizeof(char) );
intarray[0] = (int *) malloc( 4*sizeof(int) );
```

dynmem.1.c

calloc

- Reserves memory during program execution
  - Most commonly used for arrays
  - Need to specify
    - how many elements you need space for
    - the amount of memory needed for each element
  - calloc(num_elements, bytes_per_element)
    - Total memory: num_elements * bytes_per_element
    - Initializes memory to 0
  - If cannot reserve that amount of memory, returns NULL

dynmem.2.c

free

- Cancels the memory reservation from calloc or malloc
  - Memory can then be allocated to other data
- Use: free(pointer)
- Good practice to free memory before exiting program
- Only on dynamically allocated memory

dynmem.1.c
dynmem.2.c

Allocating Memory to a Pointer

- A pointer cannot be used until initialized
  - One way is with dynamic memory allocation
- Practical applications:
  - Allocate memory to a string
  - Allocate memory to a struct

dynmem.3.c
dynmem.4.c
dynmem.5.c

Using C

- You now know most of the components in C
- We will work on combining those components to create useful programs
  - Also allows us to practice using those components

Header files

- Examples of header files
  - stdio.h, math.h, string.h
    - Available throughout the system
  - The .h extension means that it's a header file
- Why use header files?
  - Keep related, commonly used functions and structs in one file
    - Can be easily used by other programs by including the header file that defines the function
    - Don't rewrite code
  - Cleans up code

Creating your own header files

- Example: created header file mylib.h for your commonly used function prototypes (printing arrays, etc.) and structs
  - #include "mylib.h"
  - Note use of quotes instead of <>
    - Quotes mean that the header file is user-defined
    - Compiler looks for the header file in the present working directory
  - The definitions for your functions will remain in a separate .c file

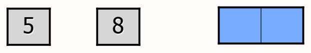
baseball.h

Recursion Review

- Divide and conquer algorithms
  - Break a problem into smaller, more managable pieces
  - Example: calculating n^x
    - Don't know the answer to n^x but know that n^x is $n*n^{x-1}$
    - Ask a smart friend the answer to n^{x-1}

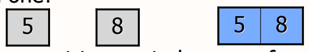
Sort more efficiently with recursion?

- Our sorting algorithms were a little slow
  - Required lots of comparisons and swaps
- Can we break the problem down?
  - How would we sort and combine two arrays, each of size one?

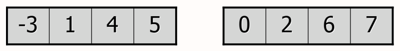


Sort more efficiently with recursion?

- Our sorting algorithms were a little slow
  - Required lots of comparisons and swaps
- Can we break the problem down?
  - How would we sort and combine two arrays, each of size one?

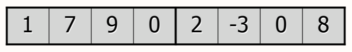


  - How would we sort two sorted arrays of equal size?



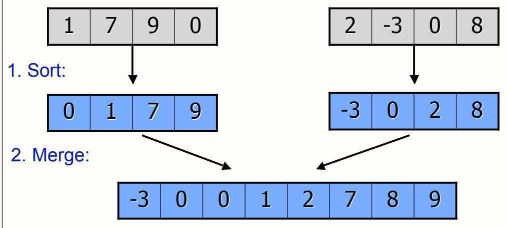
Merge Sort

- Sort each half of the array
- Merge the two halves into one array



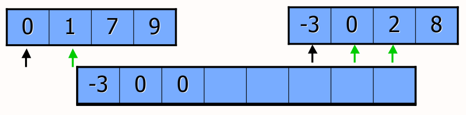
Merge Sort

1. Sort each half of the array
2. Merge the two halves into one array



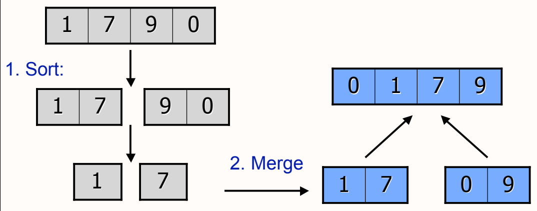
Close-up of the merge step

- Compare the first element in each array
  - Copy the smaller one into the merged array
  - Shift the ptr into the array
  - Repeat



Merge Sort

- Each sort is actually a merge sort
  - Subdivide array until reach base case (size 1)
  - Merge sorted arrays as move back "up"



High-Low Game

- I pick a number between 0 and 100
- You try to guess the number
  - I will tell you if my number is higher or lower than your guess

Binary Search

- Divide and Conquer algorithm
- Find the middle of the array
- Check if the key equals the value at mid
  - If so, report the location
- Check if the key is higher or lower than value at mid
  - Search the appropriate half of the array

-3	0	0	1	2	7	8	9
0	1	2	3	4	5	6	7

Searching for 7

- Find the middle of the array
  - Size is 8, so mid is 4
- Check if the key equals the value at mid (2)
  - If so, report the location
- Check if the key is higher or lower than value at mid
  - Search the appropriate half of the array

-3	0	0	1	2	7	8	9
0	1	2	3	4	5	6	7

7 > 2, so look in upper half

Binary Search

- mid is 6, array[6] is 8

7	8	9
5	6	7

7 < 8, so look in lower half

- mid is 5, array[5] is 7

7
5

7==7, so print an appropriate message

- What if searched for 6 instead of 7?

Searching for 6

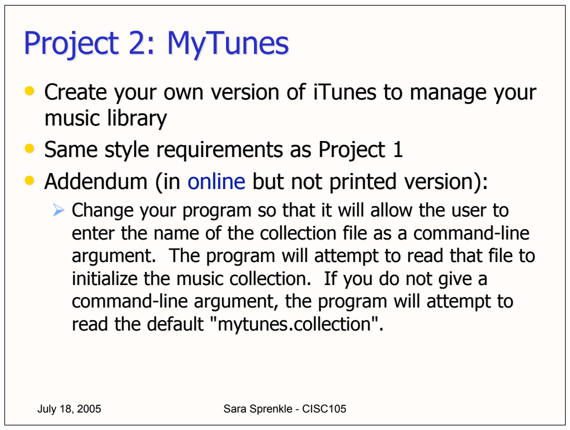
- Will follow same program flow, but 6 is not in the array
- mid is 6, array[6] is 8

7	8	9
5	6	7

- mid is 5, array[5] is 7

7
5

6 < 7, so will try to look in lower half of the array, but we've already determined it's not there.
How do we know to stop looking?



Project 2: MyTunes

- Create your own version of iTunes to manage your music library
- Same style requirements as Project 1
- Addendum (in online but not printed version):
  - Change your program so that it will allow the user to enter the name of the collection file as a command-line argument. The program will attempt to read that file to initialize the music collection. If you do not give a command-line argument, the program will attempt to read the default "mytunes.collection".