Reviewing the C basics and More Data Structures

August 1, 2005

Announcements

- No new assigned lab this week
  - Work on project 2
    - **Must** have something started to get help
  - Complete course evaluations
- Project 2 due date: August 10
  - Demos in lab on August 11
- Questions about structs or project?

Schedule Alternatives

- Option 1: the current schedule (more time for studying for final)
  - 8/8/05: review for final
  - 8/11/05: project demos
  - 8/12/05: final, 7-9 p.m.
- Option 2: an alternative schedule (more time for project)
  - 8/8/05: final
  - 8/11/05: project demos
    - maybe could push demos to 8/12/05

Quiz

Review: using strtok

- What does strtok do?
  - Breaks a string up into tokens, separated by some delimiter

Review: Binary Search

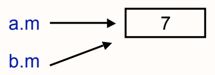
- What type of algorithm is binary search?
  - Divide and conquer
- What property of the array does binary search assume?
  - The array must already be sorted
- What is the most number of function calls required to find a value?
  - Consider an array of 10 elements
  - Consider an array of 20 elements… 100?

Review: Structs

- How are structs useful?
  - Collect data of different types into one “bundle”
  - Can manipulate data as a group
- How do you access a member of a struct?
  - The dot operator
- How do you access a member of a struct, if you’re using a pointer to the struct?
  - ->

Assigning structs

- Can use = operator
  - Copies all of the struct’s members
  - Equivalent to
    - b.m1 = a.m1;
    - b.m2 = a.m2;
    - …
    - b.mn = a.mn;
  - Note that for pointers, this is just copying the address
    - both pointers will point to the same location in memory
    - If you change the value of *a.m, it changes the value of *b.m
- ==, however, does not work



Typedef: “type definition”

- Can define your own types using typedef
- Make a new datatype for structs
  - Don’t need to use “struct” keyword all the time
- Examples:
  - typedef int employee_id;
    - Creates a special type for employee_ids, but it’s just an int
  - typedef struct MLB_Team_Struct {

```
        char name[NAME_LENGTH];
        char nickname[NAME_LENGTH];
        int wins;
        int losses;
} MLBTeam;
```

usetypedef.c

Typedef: “type definition”

- Can define your own types using typedef
- Make a new datatype for structs
  - Don’t need to use “struct” keyword all the time
- Usually found at top of the program
  - Outside of any functions
- Valid from “typedef” to end of program file

usetypedef.c

More on Pointers

- You should initialize pointers to NULL
  - Sets memory address to 0
  - If not initialized, the memory address is garbage
  - Helps find errors faster
    - If do *p, will cause a segmentation fault
  - Error Example:

```
int x, *p;
x = 10;
*p = x;  /* p was not initialized.  Dereferences a
  “garbage” location in memory */
```

More on Pointers

- You should initialize pointers to NULL
  - Sets memory address to 0
  - If not initialized, the memory address is garbage
  - Helps find errors faster
    - If do *p, will cause a segmentation fault
  - Error Example:

```
int x, *p = NULL;
x = 10;
p = &x;
*p = x;  /* p was not initialized.  Dereferences a
  “garbage” location in memory */
```

More on Pointers

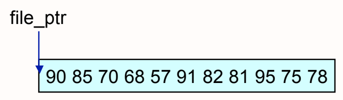
- Bad initialization:

```
int *p, x;
x=10;
p = x;  /* compiler should catch because not
  assigning an address to p */
```

File Pointers

- FILE *file_ptr;
  - A pointer to where you are in the file
  - At first, fopen opens and reads the file into memory
    - Pointer is at beginning of file

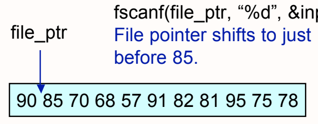
Before any read:



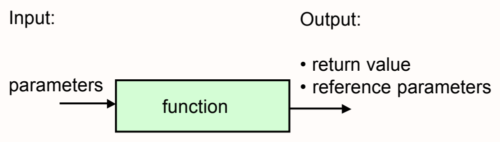
File Pointers

- FILE *file_ptr;
  - A pointer to where you are in the file
  - At first, fopen opens and reads the file into memory
    - Pointer is at beginning of file

After one call to
fscanf(file_ptr, “%d”, &input);
File pointer shifts to just before 85.



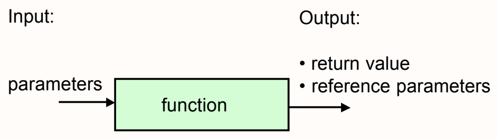
Functions



How do we keep track of the function’s output?

```
answer = function();
function( &min, &max );
```

Constant parameters



Sometimes, we pass something by reference (so we don’t have to copy it), but we don’t want it to change.
• Add the const keyword before each parameter that you don’t want to change
• Guarantees caller that a parameter passed won’t be modified
• Example: char *strtok(char *string, const char *delimiter);

Constant variables

- Use the const keyword to make constant variables
  - An error to change the variable
  - Example: days in a month
    - const int days[] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
- Constant parameters are treated as const within the function
- How different from #define?
  - #define is global, const is within a smaller scope

Getting Help

- Additional references
  - www.cplusplus.com
    - Check out the standard C libraries

Reading other people's code

- sales.c
  - Fill in comments for the code
  - Analyze what you like and don't like about the code in terms of readability, efficiency, style, etc.
  - How would you improve this code?

Analyzing sales.c

Data Structures

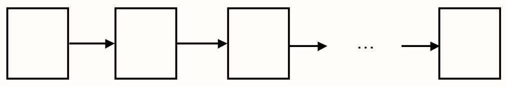
- Can create useful, reusable C data structures
- Common Data Structures
  - Manipulate to perform common tasks
  - Write once and reuse in programs that call for those tasks

Linked List

- Allows you to add and remove from the list easily
- Useful when don't know beforehand how many items are in an array

Linked List

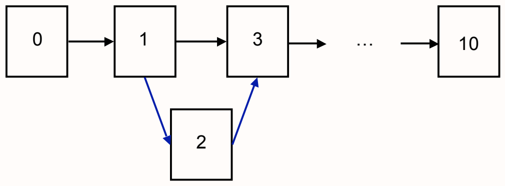
- Each item in the list includes a pointer to the next item in the list



linked_list.c

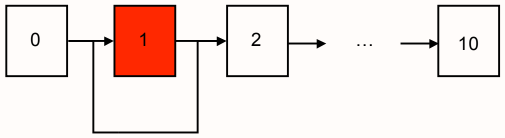
Inserting into a Linked List

- Need to update the pointers
  - What order should we update them in?



Removing from a Linked List

- Update the pointers



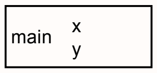
Could we use an array instead?

- Could we implement a linked list in an array?
- What are the possible limitations of the array?
- How can we address those limitations?

Data Structure: Stack

- A First In, Last Out data structure
  - Stacks of plates
  - Conversations
  - What we use to manage the program's variables

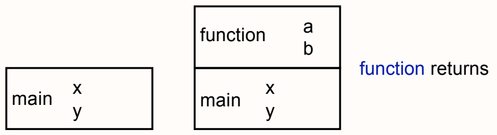
Make a call to function



Data Structure: Stack

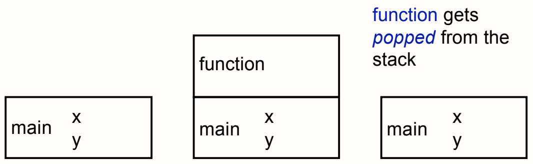
- A First In, Last Out data structure
  - Stacks of plates
  - Conversations
  - What we use to manage the program's variables

function gets *pushed* onto the stack



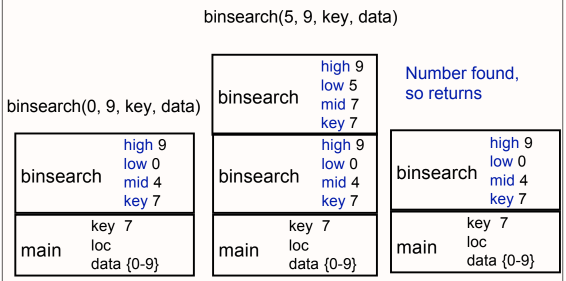
Data Structure: Stack

- A First In, Last Out data structure
  - Stacks of plates
  - Conversations
  - What we use to manage the program's variables



Data Structure: Stack

- Example: recursive calls to binary search



Stack Operations

- push: adds an element to the top of the stack
- pop: removes the top element from the stack

stack.c

Programming Practice

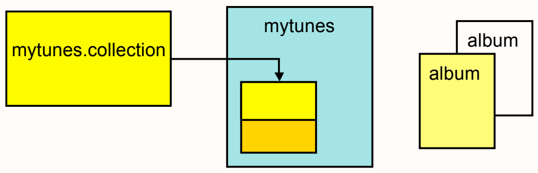
- Reversing directions
  - Original directions (miles, route, direction)
    - 30 896 N
    - 25 372 W
    - 28 74 N
    - .5 10 E
  - Reverse route (miles, route, direction)
    - .5 10 W
    - 28 74 S
    - 25 372 E
    - 30 896 S

Could we use an array instead?

- Could we implement a stack using an array?
  - What do we need to keep track of?
  - What are the limitations of an array?
- How is this implementation different from the linked list?

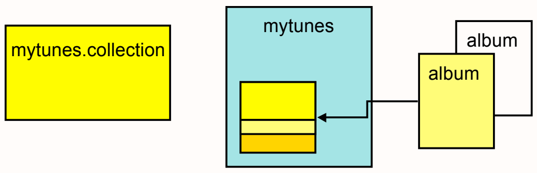
MyTunes

- Read songs into collection
  - Store each song



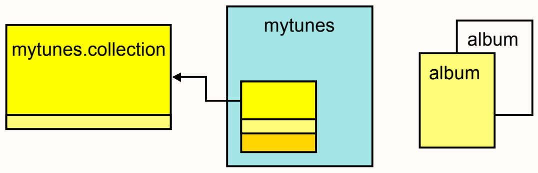
MyTunes

- Read in album



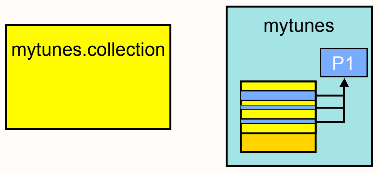
MyTunes

- Write out new collection (including new album) to the file



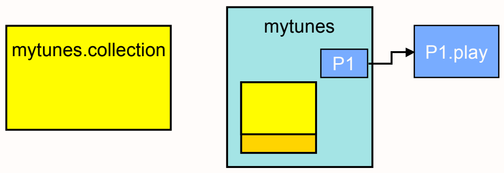
MyTunes

- Create a new playlist
  - Add songs using an identifier



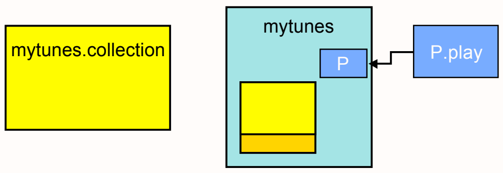
MyTunes

- Export the playlist
  - For each song in playlist…



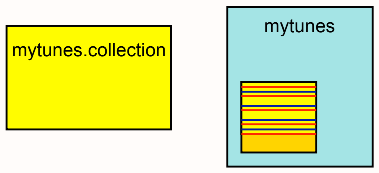
MyTunes

- Import playlist
  - For each song in playlist …



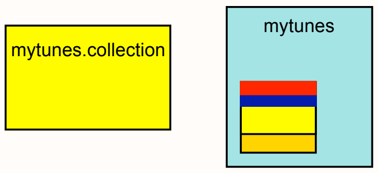
MyTunes

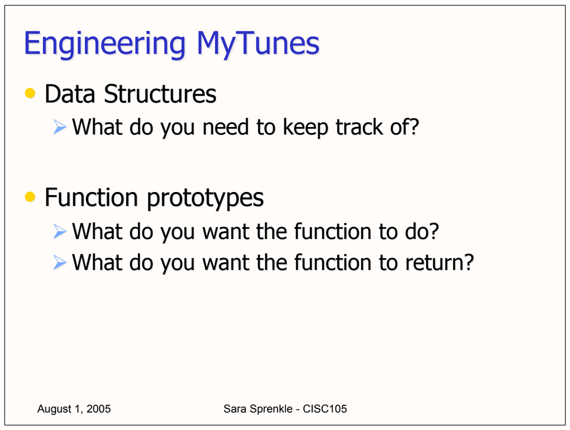
- Sort the collection
  - What are you sorting on?



MyTunes

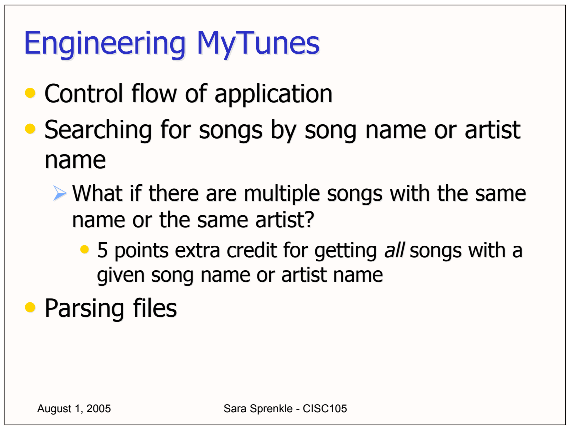
- Sort the collection
  - What are you sorting on?





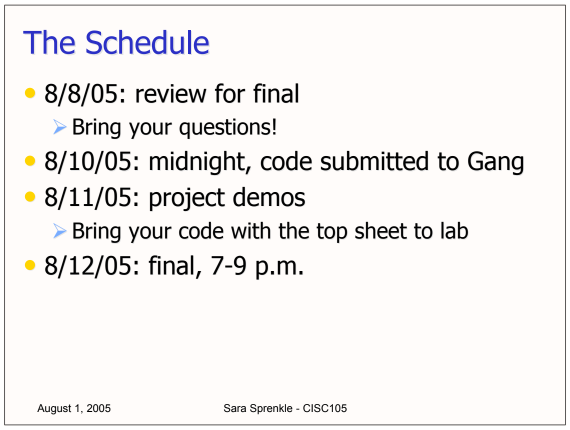
Engineering MyTunes

- Data Structures
  - What do you need to keep track of?
- Function prototypes
  - What do you want the function to do?
  - What do you want the function to return?



Engineering MyTunes

- Control flow of application
- Searching for songs by song name or artist name
  - What if there are multiple songs with the same name or the same artist?
    - 5 points extra credit for getting *all* songs with a given song name or artist name
- Parsing files



The Schedule

- 8/8/05: review for final
  - Bring your questions!
- 8/10/05: midnight, code submitted to Gang
- 8/11/05: project demos
  - Bring your code with the top sheet to lab
- 8/12/05: final, 7-9 p.m.