CS1003: Intro to CS, Summer 2008

Final review
Lab #08

Instructor: Arezu Moghadam
arezu@cs.columbia.edu

6/30/2008

Final review
Procedures

- A set of instructions for performing a task
- Can be used as an abstract tool
- Control transfer

Example

- How does printf("Hello World!"); work?
  - Someone else has written the code to handle printing
  - These procedures may take parameters and may return a result
  - Note – many parameters, single result!
- Called functions in C
Algorithms review

- Strategies of coming up with algorithms...
  - “Get foot in the door”: try to get an intuitive grasp on the problem first, conceptually
  - Stepwise refinement: take the big picture and break into smaller pieces
  - Determine if there are any iterative structures to be implemented
  - Keep boundary conditions in mind!

Iterative and recursive structures

1. Print out the first $n$ numbers, and keep a running total
2. Print out the first $n$ Fibonacci numbers
3. Write a function that calculates $x^y$ (i.e., raise $x$ to the $y$ power)
4. Reverse a list (array) of numbers
Iterative structures

- A collection of instructions is repeated in a looping manner
- Elements of an interactive structure
  - A loop; **while** or **for** loop
  - A condition determining the loop’s termination or continuation

Recursive structures

- Another way of looking at repetition
- In iterative structures a set of instructions are completed and then repeated again
- Recursion involves repeating the set of instructions as a subtask of itself
- Instead of one-after-the-other, one is performed within the other
- Example: \( \text{fib}(n) = \text{fib}(n-1) + \text{fib}(n-2); \)
Fibonacci code snippet

```c
fib (int n) {
  if  (n == 1) return 0;
  if  (n == 2) return 1;
  int sum=0;
  sum = fib(n-1) + fib(n-2);
  return sum;
}
```

We need some base case(s) condition(s) to stop the recursion.

First, come up with the recursive statement.

Big-Theta notation

Basic intuition:
- Find the number of steps in terms of \( n \) or other variables
- Drop any constants or additive lower-order terms
- Put a \( \Theta(\ ) \) around the result
- Common: \( \Theta(1), \Theta(\log \ N), \Theta(\ N), \Theta(\ N^2), \Theta(2^N) \)
Search algorithms

- Sequential search
- Binary search

Sort algorithms

- Insertion sort
- Bubble sort
- Merge sort
Sorting the list Fred, Alex, Diana, Byron, and Carol alphabetically

Running time of the insertion sort

- Worst case scenario
- At each iteration the pivot is compared with all previous entries
  - \[1 + 2 + 3 + \ldots + (n-1) = \frac{1}{2}n^2 = \Theta(n^2)\]
"Bubbling Up" the Largest Element

- Traverse a collection of elements
  - Move from the front to the end
  - "Bubble" the largest value to the end using pair-wise comparisons and swapping

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>42</td>
<td>77</td>
<td>35</td>
<td>12</td>
<td>101</td>
<td>5</td>
</tr>
</tbody>
</table>

Swap

Swap

"Bubbling Up" the Largest Element

- Traverse a collection of elements
  - Move from the front to the end
  - "Bubble" the largest value to the end using pair-wise comparisons and swapping

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>42</td>
<td>35</td>
<td>77</td>
<td>12</td>
<td>101</td>
<td>5</td>
</tr>
</tbody>
</table>
"Bubbling Up" the Largest Element

- Traverse a collection of elements
  - Move from the front to the end
  - "Bubble" the largest value to the end using pair-wise comparisons and swapping

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>42</td>
<td>35</td>
<td>12</td>
<td>77</td>
<td>101</td>
<td>5</td>
</tr>
</tbody>
</table>

No need to swap
"Bubbling Up" the Largest Element

- Traverse a collection of elements
  - Move from the front to the end
  - "Bubble" the largest value to the end using pair-wise comparisons and swapping

```
1          2          3          4            5            6
42 35 12 77 5 101
```

Largest value correctly placed
"Bubbling" All the Elements

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>42</td>
<td>35</td>
<td>12</td>
<td>77</td>
<td>5</td>
<td>101</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>35</td>
<td>12</td>
<td>42</td>
<td>5</td>
<td>77</td>
<td>101</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>12</td>
<td>35</td>
<td>5</td>
<td>42</td>
<td>77</td>
<td>101</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>12</td>
<td>5</td>
<td>35</td>
<td>42</td>
<td>77</td>
<td>101</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>12</td>
<td>35</td>
<td>42</td>
<td>77</td>
<td>101</td>
</tr>
</tbody>
</table>

Bubble sort algorithm

```c
for(i=alength - 1; i > 0; i--) {
    for(j = 0; j < i; j++) {
        if(a[j] > a[j+1]) {
            int temp = a[j];
            a[j] = a[j+1];
            a[j+1] = temp;
        }
    }
}
```

$\Theta(n^2)$
Merge sort

```c
void mergesort(Item a[],
    int start, int stop)
{
    int m = (start + stop)/2;
    if (stop <= start) return;
    mergesort(a, start, m);
    mergesort(a, m + 1, stop);
    merge(a, start, m, stop);
}
```

$\Theta(n \log n)$

---

Data structures

- Data structures
  - Arrays
  - Lists
    - Stacks
    - Queues
  - Trees
    - Binary trees
- How to implement data structures
Implementing lists

- Linked lists
  - Lists with dynamic structures
  - Head pointer to point to the head of the list
  - NIL pointer to the end of the list

```c
struct list_el {
    int val;
    struct list_el * next;
};
typedef struct list_el item;
```

Add a new node to a linked list

```c
addnode(int value, struct NODE *position) {
    struct NODE *new;
    if(new=(struct NODE*)malloc(sizeof(struct NODE))) {
        new->nodevalue = value;
        new->next = position->next;
        position->next = new;
    }
    else
        printf("Memory shortage");
}
```
Delete a node from data structure

```c
int deletenode(struct NODE *target)
{
    struct NODE *curr = list;
    while(curr->next != target)
        curr = curr->next;
    curr->next = target->next;
    deletedval = target->value;
    free(target);
    return(deletedval);
}
```

The big picture

```
+-------------------------------------+----------------+----------------+-----------------+----------------+
| Application                         | Operating      | Instruction    | Architecture    |
|                                    |                | Architecture   | I/O System      |
| Compiler                           | Firmware       |                | Datapath & Control|
|                                    |                |                | Digital Design  |
|                                    |                |                | Circuit Design  |
|                                    |                |                | Layout           |
```
Multitasking

More complicated resource contention requires locking; concept is similar to the barriers at a train track crossing.
- Semaphores == fancy locks implemented by flags

Avoid deadlock:
Various AI methodologies

- Reasoning/production systems
- Neural networks
- Genetic algorithms
- Natural language processing
- Robotics, vision
- Databases/expert systems

A large search problem
Search strategies

- **Breadth-first**
  - Look at the first row, then the second row, then the third row...

- **Depth-first**
  - Go all the way to one leaf, then backtrack and resume

- **Best-first**

- **Heuristic**
  - Have a special piece of code that “tells” you a preferred choice
  - A directed search – not always foolproof, but reduces amount of nodes searched
  - For 8-puzzle: “# of tiles out of place” – take move that minimizes this value

Learning

- **Imitation**
  - Demonstrating steps in a task

- **Supervised Training**
  - Correct responses for a series of examples
  - Training set

- **Reinforcement**
  - Agent needs to figure out responses to the training set

- **Evolutionary Techniques**
  - Evolution of agents through generations
  - Evolve toward the solution from a set of proposed solutions
An artificial neuron

An example of a processing unit

- Positive weights exciting effect
- Negative weight inhibiting effect on the input
- What is the output for input of 1,1,0?
Training an artificial network

- Backpropagation
  - Neural networks are trained not programmed

- Learns weights by supervised training

- Weights are adjusted by small increments

- Desired output 1 when inputs are different

Associative memory

- Human mind can retrieve information associated with a current topic of consideration

- Smell, music

- Hopfield networks, neural networks without input or output

- System reaches a stable state from an unstable configuration

- Stable bit pattern close to the given bit pattern

- Representing smell or memories with bit patterns
Representing an associative memory

Lab 08
Pointer Arithmetic (exercise)

- What do the following return?
  - given -> char data = 'a'; char * ptr = &data;

  1. &data
  2. ptr
  3. &ptr
  4. *ptr
  5. *(ptr+1)
  6. *(ptr+1)
  7. ++ptr
  8. ptr++
  9. *(++ptr)
  10. *(++ptr)
  11. *ptr++
  12. (**ptr)++
  13. +++(*ptr)
  14. +++(*ptr)
  15. +++*ptr

Unions

- There are like structs, however they have only one memory space.

  union structure-name {
    field-type field-name;
    field-type field-name;
    ....
  } variable-name;
Unions

- There are like structs, however they have only one memory space.

```c
union structure-name {
    field-type field-name;
    field-type field-name;
    ....
} variable-name;
```

Unions II

```c
struct bin {
    char name [30];  // name of the part
    int quantity;    // how many in the bin
    double cost;    // the cost of the single part
} printer_cable_bin;  // where we put the cables
```

VS

```c
union bin {
    char name [30];  // name of the part
    int quantity;    // how many in the bin
    double cost;    // the cost of the single part
} printer_cable_bin;  // where we put the cables
```

- Make space for largest variable

<table>
<thead>
<tr>
<th>name</th>
<th>quantity</th>
<th>cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>quantity</td>
<td>cost</td>
</tr>
</tbody>
</table>
Unions III

- You can overwrite quantities, in union
  ```
  printer_cables_bin.name = "Printer Cables"
  printer_cables_bin.cost = 10;
  printf("The name of the bin is %s\n",
         printer_cables_bin.name);
  ```
  - What will the produce?
  - Answer: Unexpected result
  - You must keep track of which field you used

- So why use this?
  - Memory space saving

Example

```c
union value {
    long int i_value;
    float f_value;
} data;
int i;
float f;
main() {
    data.f_value = 5.0;
    data.i_value = 3;
    i = data.i_value; /* legal */
    f = data.f_value; /* not legal */
    data.f_value = 5.5;
    i = data.i_value; /* not legal */
    return 0;
}
```
Typedefs

- Struct allows you to create a data type/structure
- Typedefs allow the programmer to define their own variable type

Typedefs II

- Usage
  - typedef type-declaration;
  - where type-declaration is the same as variable declaration, except that a type name is used instead of a variable name
  - eg: typedef int count; //creates a new type count that is the same as an integer
  - Now you can say – count a; //equal to int a;
Typedefs III

- But you can get more complex
  - typedef int group[10];
    - You can now say group classroom, which will create a variable classroom of 10 integers
  
  ```c
  main() {
    typedef int group[10];
    group class;
    for (i=1; i<10; i++)
      class[i] = 0;
    return 0;
  }
  ```

Typedefs IV

- But you can get more complex
  - typedef struct bin bin
    - This creates a variable type bin of type struct bin, and you can now say bin printer_cables_bin, instead of struct bin printer_cables_bin
    ```c
    struct bin {
      char name [30];
      int quantity;
      int cost;
    };
    
    typedef struct bin bin;
    
    bin printer_cables_bin = {"Printer Cables", 10, 1290};
    ```
Enums

- This is designed for variables that contain only a limited set of values
- Traditionally, if you wanted to set up the days of a week, you would -

```c
typedef int week_day;
const int Sunday = 0;
const int Monday = 1;
const int Tuesday = 2;
const int Wednesday = 3;
const int Thursday = 4;
const int Friday = 5;
const int Saturday = 6;

week_day today = Tuesday;
```

Enums II

- That was cumbersome
- You can say

```c
enum week_day {Sunday, Monday,
    Tuesday,   Wednesday,
    Thursday, Friday, Saturday};

enum week_day today = Tuesday;
```

- Usage

```c
enum enum-name (tag-1, tag-2, ....}
    variable-name;
```
Enums III

- You can omit variable-name, like in struct and union
- C implements the enum type as compatible with integer, so it is legal to say
  - `today = 5; //though this may throw a warning // will make today Thursday`

Enums IV – more examples

```c
enum week_day {Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday};
enum week_day d1, d2; // makes d1 and d2 of type  // enum week_day

d1=Friday;
if (d1==d2)
...
```
Enums V – more examples

- You can use it to do switches

```c
enum week_day {Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday};
typedef enum day day;

day find_next_day(day d) {
    day next_day;
    switch(d) {
        case Sunday:
            next_day = Monday;
            break;
        case Monday:
            next_day = Tuesday;
            break;
        ...
        case Saturday:
            next_day = Sunday;
            break;
    }
    return next_day;
}
```

Casting

- Convert one type of variable to another
- (type) expression
- You already know this

```c
int a;
float b, total;
total = (float)a + b;
```
Thank you!

- You guys have been a great audience.
- I hope you found this class rewarding.
- Good luck with the rest of your Computer Science mini-careers!
  - And with the final