Introduction to Computer Science and Programming in C

Session 7: September 23, 2008
Columbia University
Announcements

- Homework 1 due.
- Homework 2 out this afternoon. Due 10/7
Review

- Conditionals:
  - if (<condition>)
  - switch (variable)

- Loops:
  - while (<condition>)
  - for (<init>; <condition>; <count>)
Today

- Functions
- Variable Scope
- Recursion
Functions

- We’ve seen functions already
  - int main()
  - printf()
- But we can write our own functions too!
Function Syntax

- `<return type>` `myFunction(<arguments>)`
  
  `/* instructions to execute function */`
  
- `int multiply(int x, int, y)`
  
  `{ return x*y; }

Note: functions can return type `void`, which essentially means they return nothing, like `printf()`
Abstraction Revisited

- Functions allow us to abstract away algorithms
- Once a function is written and tested, you can forget about how it works.
- We can describe the logic at a higher level.
Example

/* if we have three arrays A[10], B[10], C[10] */
/* Check if they are equal */
int check = 1;
for (i=0; i<10; i++) {
    check = check && (A[i]==B[i]);
}
for (i=0; i<10; i++) {
    check = check && (B[i]==C[i]);
}
Example

```c
int arrayEqual(int X[], int Y[], int N)
{
    int i, check=1;
    for (i=0; i<N; i++) {
        check = check && (X[i]==Y[i]);
    }
}

/* main code */
int check;
check = arrayEqual(A,B,10) && arrayEqual(B,C,10);
```
Scope

- float half(float x)
  {
    return x/2;
  }

- int main()
  {
    float x=10.0, y;
    y = half(x);
    /*what is x? ...*/
    x is 10

- **scope** - area of program where variable is valid.

- **global variable** - variable is valid everywhere

- **local variable** - only valid within **block**

- **block** – area of code enclosed by {}

- We have only been using **local** variables so far in class
Global Variables

- Declared outside of any functions

```c
#include <stdio.h>

int count; /* a global variable */

int someFunction()
{
    ...
}

int main()
{
    ...
}
```
Scope Example

/* From PCP Figure 9-1 */
int global;
int main()
{
  int local;
  global = 1;
  local = 2;
  {
    int very_local;
    very_local = global + local;
  }
}

12
Scope Weirdness

- `int count = 0;`

  ```
  if (myCondition(count)) {
    int count;
    count++; /* which count is this incrementing? */
  }
  ```

- Be careful when naming variables
Scope Weirdness 2

- /* did not declare count */
  if (myCondition(count)) {
    int count;
    count++;
  }

  printf("count is %d\n", count); /* won’t compile*/

- Make sure you declare in the correct block.
Rule of thumb

- Try to avoid global variables

- Share information between parts of your program via function arguments and return values

- Keeps code easier to manage
Recursion

- What happens when a function calls itself?

```c
int paradox(int x)
{
    return paradox(-x);
}
```

- **Recursion** is when a function calls itself.

- We can implement loops with recursion.
Recursion

void loop(int i)
{
    if (i>1) {
        <do something>
        loop(i-1);
    } else
    return;
}

Towers of Hanoi

- Classic Computer Science example for recursion
- Three pegs, N discs of different sizes:

Rules for Hanoi

- Start with all discs on one peg in order of size
- Only move one disc at a time
- Only move the top disc of any peg
- No disc may be placed on top of a smaller disc