COMsW 1003-1

Introduction to Computer Programming in C

Lecture 8

Spring 2011

Instructor: Michele Merler

http://www1.cs.columbia.edu/~mmerler/comsw1003-1.html
Announcements

Homework 1 correction out this afternoon

Homework 2 is out

– Due Monday, February 28th

– Start early (especially Exercise 2)!
Today

• Functions

• Recursion

• Debugging (if time)
Infinite Loops

- Loops where the condition is always TRUE

- Will stop only with:
  - `break`
  - Modification of the condition variables

```c
while ( 1 ){

    /* body modifies x */

    if( x != 0 ) {
        break;
    }

}
```
Infinite Loops

• Loops where the condition is always TRUE

• Will stop only with:
  • `break`
  • modification of the condition variables

```c
while ( 1 ){
    /* body modifies x */
    if( x!= 0 ) {
        break;
    }
}
```

`while ( 1 is true )`

Always!
Operators - Logical

• A variable with value 0 is false, a variable with value !=0 is true

```c
int x = 3, y = 0, z, k, t, q = -3;

z = x && y;  // z = 0;  x is true but y is false
k = x || y;   // k = 1;  x is true

q = !q;       // q = true
```

```c
```
Infinite Loops

• Loops where the condition is always TRUE

• Will stop only with:
  • break
  • modification of the condition variables

```c
int cond = 7;

while (cond){
  /* body */
  if( x[3][5] != 7 ){
    cond = 0;
  }
}
```

Until we set `cond` to 0!
Functions Example

- Simple calculator
- Program that computes one basic arithmetic operation between 2 numbers
Functions - Recursion

• What if a function calls itself? Recursion
Functions - Recursion

• A recursive function must have two properties:
  – **Ending point** (i.e. a terminating condition)
  – **Simplify the problem** (every call is to a simpler input)
Example: Fibonacci sequence

In mathematics, famous numbers following the sequence

\[0 \ 1 \ 1 \ 2 \ 3 \ 5 \ 8 \ 13 \ 21 \ 34 \ 55 \ 89 \ ...\]

Given \(F_0 = 0\), \(F_1 = 1\) can be computed with recurrence

\[F_n = F_{n-1} + F_{n-2}\]

Code to compute the first 100 Fibonacci numbers:

```c
int i = 0;
int fib[100];

fib[0] = 0;
fib[1] = 1;

for( i = 2; i < 100 ; i++ ) {
    fib[i] = fib[i-1] + fib[i-2];
}
```
Functions - Recursion

• What if a function calls itself? Recursion
• What is the value of the number at position num in the Fibonacci sequence?

```c
/* Fibonacci value of a given position in the sequence */
int fib ( int num ) {

    switch(num) {
        case 0:
            return(0);

        case 1:
            return(1);

        default: /* Including recursive calls */
            return(fib(num - 1) + fib(num - 2));
    }
}
```

Why are there no breaks?
Functions - Recursion

- What if a function calls itself? Recursion
- What is the value of the number at position num in the Fibonacci sequence?

```c
/* Fibonacci value of a given position in the sequence */
int fib ( int num ) {

    switch(num) {
        case 0:
            return(0);
        case 1:
            return(1);
        default: /* Including recursive calls */
            return(fib(num - 1) + fib(num - 2));
    }
}
```
Debugging
Debugging

• Debugging consists basically in finding and correcting **run-time errors** in your program

• Multiple ways of doing it
  • Manual runs (for small programs)
  • Insert `printf()` in key lines

• There also exist INTERACTIVE debugging tools

• We will now see a basic one for UNIX: `gdb`
1. In order to use gdb on a program, we must use the \texttt{--g} option when compiling it

\begin{verbatim}
gcc  -g  program.c   -Wall -o  nameOfExecutable
\end{verbatim}

2. Then, we can use the gdb command to start the interactive debugging environment

\begin{verbatim}
gdb nameOfExecutable
\end{verbatim}
gdb commands

• **run**: run executable (program) currently watched.
  
  (gdb) run

• **kill**: kill current execution of program
  
  (gdb) kill

• **list**: show program source code
  
  (gdb) list 2, 8 : shows lines 2 to 8 from source program

• **print**: print value of a variable or expression at the current point
  
  (gdb) print buf
gdb commands

• **break** : insert breakpoint in program. Debugging run will stop at the breakpoint
  
  (gdb) break  nameSource.c  :  lineNumber
  (gdb) break  test.c:  12

• **next** : step to the next line (execute current line)
  
  (gdb) next

• **continue** : continue with execution until next breakpoint or end of program
  
  (gdb) continue

• **Quit** : exit gdb
  
  (gdb) quit
Graphical GDB

- gdb can be run from Emacs
- Press \texttt{M-x} (in Windows \texttt{Esc - x})
- Insert \texttt{gdb}
- Insert \texttt{executableName}
- Visual debugger
Can enable breakpoints with a click.