COMsW 1003-1

Introduction to Computer Programming in C

Lecture 9

Spring 2011

Instructor: Michele Merler

http://www1.cs.columbia.edu/~mmerler/comsw1003-1.html
Are Computers Smarter than Humans?

IBM's Watson on 'Jeopardy': Computer takes big lead over humans in Round 2

February 15, 2011 | 9:20 pm

On Tuesday night’s "Jeopardy" episode, Watson, the IBM supercomputer, steamrolled to a commanding lead over his human competitors.

Today

• Homework 1 Correction

• Debugging (from Lecture 8)

• C Preprocessor
Conditional Assignment

- Another way of embedding `if - else` in a single statement
- Uses the `? :` operators

\[
\text{variable} = ( \text{condition} ) \ ? \ \text{val1} : \ \text{val2} ;
\]

- If condition is `true`, we assign `val1` to variable
- If condition is `false`, we assign `val2` to variable

\[
\begin{align*}
\text{int} \ x &= 7, \ y; \\
y &= ( \ x > 5 \ ) \ ? \ x : 5; \\
y &= 7
\end{align*}
\]
The comma operator

• In C statements can also be separated by `,` not only `;`

```c
int x = 2;
int y;

x++, y = x/3, y += 2;
```

Be careful with declarations!

```c
int x = 2, char c = 'm';  // Different types, NO
int x = 2, y;             // Same type, OK
```
The comma operator

Special case, the \texttt{for} loop statement

Example: the palindrome word checking. Check if a word is the same when read right to left

\begin{verbatim}
int i, flag = 1;

char word[100] = “radar”;

for( i=0 , j=strlen(word)-1 ; i < strlen(word)/2 ; i++ , j-- ) {
    if( word[i] != word[j] ) {
        flag = 0;
        break;
    }
}
\end{verbatim}
The comma operator

Special case, the **for** loop statement

Example: the palindrome word checking

```c
for( i=0, j=strlen(word)-1 ; i < strlen(word)/2 ; i++ , j-- ) {
    if( word[i] != word[j] ) {
        flag = 0;
        break;
    }
}
```

**Initial conditions**

**Change conditions**
Advanced Types - Const

`const` defines a variable whose value cannot be changed

```c
const double PI = 3.14;
double r = 5, circ;
circ = 2 * PI * r;
PI = 7;
```
**Advanced Types - Const**

`const` defines a variable whose value cannot be changed.

```c
const double PI = 3.14;

double r = 5, circ;

circ = 2 * PI * r;

PI = 7;  // Once it’s initialized, a const variable cannot change value
```
C Preprocessor
C Preprocessor

Preprocessor is a facility to handle
  – Header files
  – Macros

Independent from C itself, it’s basically a text editor that modifies your code before compiling

Preprocessor statements begin with `#` and do **not** end with `;`
C Preprocessor

myFile.c (program)

Compiler

myFile (executable)

0100101010021
0101001010000
11110011...

...010010100001
1110001110101
1110001110101
C Preprocessor

myFile.c (program)

myFile.c (preprocessor code)

myFile (executable)

Preprocessor

Compiler
View Preprocessor Code

**gcc** has a special option that allows to run only the preprocessor

```
gcc -E myFile.c
```

We can send output to a file using the UNIX > operator

```
gcc -E myFile.c > outFile.txt
```

Saves gcc’s output to outFile.txt
Header files

• Header files are fundamentally libraries
• Their extension is .h
• They contain function definitions, variables declarations, macros
• In order to use them, the preprocessor uses the following code

```
#include <nameOfHeader.h>
#include "nameOfHeader.h"
```

For standard C libraries
For user defined headers

• So far, we have used predefined C header files, but we can create our own! (more on this in upcoming Lectures)
Header files

```
#include <stdio.h>
...
```

Preprocessor
• A **macro** is a piece of code *c* which has been given a name *n*

• Every time we use that *n* in our program, it gets replaced with *c*

• The preprocessor allows you to declare them with **#define**

• Two types:
  – Object-like macros
  – Function-like macros
Object like macros

• Constants, usually defined on top of programs

```c
#define name text_to_substitute

#define SIZE 10

#define FOR_ALL for( i=0; i< SIZE; i++ )
```
Object like macros

#define SIZE 10

/* main function */
int main(){

    int arr[SIZE];

    return(0);
}

From now on, every time we write SIZE inside our program it is going to be replaced by 10
Object like macros

- Some compilers do not allow you to declare arrays with a variable as size

```c
int size1 = 10;
int arr1[size1]; /* should always cause error */

const int size2=10;
int arr2[size2]; /* causes errors in many compilers */

#define SIZE 10
int arr3[SIZE]; /* OK in any C compiler */
```
Function-like macros

- Macros that can take parameters like functions

  ```c
  #define SQR(x) ((x) * (x))
  #define MAX(x,y) ((x) > (y) ? (x) : (y))
  ```

- Parameters MUST be included in parentheses in the macro name, without spaces
- It is a good habit to include parameters in parentheses also in the text to be substituted
Conditional Compilation

• Allows to use or not certain parts of a program based on definitions of macros

```c
#ifdef var
  if var is defined, consider the following code
#endif

#ifndef var
  if var is not defined, consider the following code
#endif

#else

#endif

define

#endif

#undef var
  (opposite of #define)
```
#define DEBUG

\[\ldots\]\n
#ifdef DEBUG

printf(“The value of x is \%d\n”, x);

#endif

If DEBUG was defined earlier in the program, then the statement printf(...) is considered, otherwise the preprocessor does not copy it to the file to be compiled.