Topic 3: C Basics

CSE 30: Computer Organization and Systems Programming
Summer Session II 2011

Dr. Ali Irturk
Dept. of Computer Science and Engineering
University of California, San Diego
Has there been an update to ANSI C?

- Yes! It’s called the “C99” or “C9x” std
  - You need “gcc -std=c99” to compile

- References
  - [http://home.tiscalinet.ch/t_wolf/tw/c/c9x_changes.html](http://home.tiscalinet.ch/t_wolf/tw/c/c9x_changes.html)

- Highlights
  - Declarations anywhere, like Java (#15)
  - Java-like // comments (to end of line) (#10)
  - Variable-length non-global arrays (#33)
  - `<inttypes.h>`: explicit integer types (#38)
  - `<stdbool.h>` for boolean logic def’s (#35)
  - `restrict` keyword for optimizations (#30)
C vs. Java™ Overview (1/2)

Java
- Object-oriented (OOP)
- “Methods”
- Class libraries of data structures
- **Automatic** memory management

C
- No built-in object abstraction. Data separate from methods.
- “Functions”
- C libraries are lower-level
- **Manual** memory management
- **Pointers**
<table>
<thead>
<tr>
<th>Java</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>High memory overhead from class libraries</td>
<td>Low memory overhead</td>
</tr>
<tr>
<td>Relatively Slow</td>
<td>Relatively Fast</td>
</tr>
<tr>
<td>Arrays initialize to zero</td>
<td>Arrays initialize to garbage</td>
</tr>
<tr>
<td>Syntax:</td>
<td>Syntax:</td>
</tr>
<tr>
<td>/* comment */</td>
<td>/* comment */</td>
</tr>
<tr>
<td>// comment</td>
<td>printf</td>
</tr>
<tr>
<td>System.out.println</td>
<td></td>
</tr>
</tbody>
</table>
C Syntax: General

- Header files (.h) contain function declarations - the function interface
- The .c files contain the actual code.

```
File.h

void func1(int, char *);
int func2(char *, char *);

File.c

void func1(int a, char *b)
{
    if(a > 0)
    {
        *b = 'a';
    }
}

int func2(char *a, char *b)
{
    ...
}
```

- Comment your code:
  - only /* */ works, & they can’t be nested
  - // doesn’t work in C
C Syntax: `main`

- To get the `main` function to accept arguments, use this:
  ```c
  int main (int argc, char *argv[])
  ```
- What does this mean?
  - `argc` will contain the number of strings on the command line (the executable counts as one, plus one for each argument).
    - Example: `unix% sort myFile`
  - `argv` is a pointer to an array containing the arguments as strings (more on pointers later).
C Syntax: Variable Declarations

- All variable declarations must go before they are used (at the beginning of the block).
- A variable may be initialized in its declaration.
- Examples of declarations:
  - **correct**: 
    ```c
    int a = 0, b = 10;
    ...
    ```
  - **incorrect**: 
    ```c
    for (int i = 0; i < 10; i++)
    (but OK for C99)
    ```
Common C Error

- There is a difference between assignment and equality
  
  \[
  a = b \quad \text{is assignment}
  \]
  
  \[
  a == b \quad \text{is an equality test}
  \]

- This is one of the most common errors for beginning C programmers!

  - One solution (when comparing with constant) is to put the variable on the right!
  
    If you happen to use = it won’t compile.
    
    \[
    \text{if (3 == a) \{ ...}
    \]
C Syntax: True or False?

- What evaluates to FALSE in C?
  - 0 (integer)
  - NULL (pointer: more on this later)
  - no such thing as a Boolean

- What evaluates to TRUE in C?
  - everything else…
C syntax : control flow

- Within a method / function
  - if-else
  - switch
  - while and for
  - do-while

If-else control structure

```c
if(a == 0)
    { i++; }
else if(a == 1)
    { i--; }
else if(a == 2)
    { i = 2; }
else
    { i = 3; }
```

How do we convert this into an equivalent case control structure?

For control structure

```c
for(i = 0; i < 20; i++)
    {  a[i] = b[i]; }
```

How do we convert this into an equivalent while control structure?
Address vs. Value

- Consider memory to be a single huge array:
  - Each cell of the array has an address associated with it.
  - Each cell also stores some value.

- Don’t confuse the address referring to a memory location with the value stored in that location.

```plaintext
101 102 103 104 105 ...

... 23 ... 42 ... ...
```
Pointers

- An address refers to a particular memory location. In other words, it **points** to a memory location.

- **Pointer**: A variable that contains the **address** of a variable.

![Diagram showing memory locations and pointers](image)
Pointers

How create a pointer:

- & operator: get address of a variable

```c
int *x, y;
x = &y;
y = 3;
```

How get a value pointed to?

- * “dereference operator” : get value pointed to

```c
printf(“x points to %d\n”, *x);
```
Pointers

- How change variable pointed to?
  - Use dereference * operator to left of =

\[ *x = 5; \]

\[
\begin{array}{c}
\text{x} \\
\text{y} \\
3
\end{array}
\]

\[
\begin{array}{c}
\text{x} \\
\text{y} \\
5
\end{array}
\]
C passes a parameter “by value”

- procedure/function gets a copy of the parameter, so changing the copy cannot change the original

```c
void addOne (int x) {
  x = x + 1;
}
int y = 3;
addOne(y);
  printf(“The value of y is %d”, y);
```

What will be displayed?
Pointers and Parameter Passing

- How to get a function to change a value?

```c
void addOne (int *x) {
    *x = *x + 1;
}

int y = 3;

addOne (&y);
```

- What will be displayed?
Pointers

- Normally a pointer can only point to one type (int, char, a struct, etc.).
  - void * is a type that can point to anything (generic pointer)
  - Use sparingly to help avoid program bugs!
More C Pointer Dangers

- Declaring a pointer just allocates space to hold the pointer – it does not allocate something to be pointed to!
- Local variables in C are not initialized, they may contain anything.
- What does the following code do?
  ```c
  void f()
  {
    int* x;
    *x = 5;
  }
  ```
After declaring a pointer:

```c
int *ptr;
```

`ptr` doesn’t actually point to anything yet. We can either:

- make it point to something that already exists,
  or
- allocate room in memory for something new that it will point to… (next lecture)
Pointers & Allocation

- Pointing to something that already exists:
  ```c
  int *ptr, var1, var2;
  var1 = 5;
  ptr = &var1;
  var2 = *ptr;
  ```

- var1 and var2 have room implicitly allocated for them.
void main(); {
    int *p, x=5, y; // init
    y = *(p = &x) + 10;
    int z;
    flip-sign(p);
    printf("x=%d,y=%d,p=%d\n",x,y,p);
}

flip-sign(int *n){*n = -(*n)}

How many syntax/logic errors in this C99 code?
void main() {  
  int *p, x=5, y; // init  
  y = *(p = &x) + 10;  
  int z;  
  flip-sign(p);  
  printf("x=%d,y=%d,p=%d\n",x,y,*p);  
}  
flip-sign(int *n){*n = -(*n);}  

How many syntax/logic errors? I get 5.  
(signed printing of pointer is logical error)
Conclusion

- All declarations go at the beginning of each function except if you use C99.
- Only 0 and NULL evaluate to FALSE.
- All data is in memory. Each memory location has an address to use to refer to it and a value stored in it.
- A pointer is a C version of the address.
  * “follows” a pointer to its value
  & gets the address of a value