CSE 303
Concepts and Tools for Software Development

Magdalena Balazinska
Winter 2010
Lecture 7 – Introduction to C
Welcome to C

- Going from Java to C is like going from an automatic transmission to a stick shift
  - Lower level: much more is left for you to do
  - Unsafe: you can set your computer on fire
  - C standard library is much smaller
  - Similar syntax can both help and confuse
  - Not object oriented: paradigm shift

- We will also learn C++ later this quarter
  - Both better and worse than C
Our Plan for Learning C

- Learn non-object oriented programming
- Gain a deep understanding of
  - Memory management
  - Pointers
  - Program execution
  - We will “look under the covers”
- Acquire good debugging skills
- Acquire software development techniques
- And also learn the C syntax
Our Plan for Today

- Introduction to memory management
- Simple C programs
- A first look at pointers
Address Space of a Unix Process

Address space is just array of 8-bit bytes

Typical total size is: $2^{32}$ or $2^{64}$

We will assume that integer is 4 bytes

A pointer is just an index into this array

0xFFFFF.FFFFFFF

Address space

0x00000000

stack (dynamically allocated)

heap (dynamically allocated)

static data (globals) (data segment)

code (text segment)
More about the Address Space

- **An address** refers to a position in this array
- Trying to read an unused part of the array may cause a “**segmentation fault**” (crash)
- **Code**: instructions of program (read-only)
- **Static data** contains global variables
- **Stack**: local variables and code address
  - Grows and shrinks as program executes
- **Heap**: data (Objects returned by Java's new)
  - Must manage manually
Hello World

#include <stdio.h>

/*
 * First C program
 */
int main() {

    printf("Hello World\n");

    return 0;
}

Testing Hello World

- To compile the program, `hello.c`
  
  ```
gcc -g -Wall -o hi hello.c
  ```

- To execute the program:
  
  ```
./hi
  ```
gcc -g -Wall -o hi hello.c

Meaning:
gcc: Gnu C Compiler
-g: include debugging information
-Wall: show all warnings
-o hi: specifies program name

If you do not specify a name
gcc -g -Wall hello.c

The executable will be called: a.out
Quick Hello World Explanation

- `#include <stdio.h>`
  - Directive to the C preprocessor (more later)
  - Finds file `stdio.h`, includes its entire content
  - `stdio.h` is a header file
  - `stdio.h` describes `printf`

- `main` is a function
  - Every C program begins executing at the function `main`

- `\n` is an escape sequence. Means newline.
C Functions

- A lot like Java methods but...
  - They are not part of a class
  - They are not associated with an object
  - No “this”
Address Space of a Unix Process

Address space is just array of 8-bit bytes

Typical total size is: $2^{32}$

We will assume that integer is 4 bytes

A pointer is just an index into this array
About the Stack

- The call-stack (or just stack) has one “part” or “frame” (also called activation record) for each function call that has not yet returned.

- It holds
  - Room for local variables
  - The return address (index into code for what to execute after the function is done)

- Hello World is not interesting to discuss the stack, so let's try a different example...
### Activation Record

<table>
<thead>
<tr>
<th>Return address</th>
<th>Note: each item on the stack can be many bytes in size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Info where to write returned val</td>
<td></td>
</tr>
<tr>
<td>Argument 1</td>
<td></td>
</tr>
<tr>
<td>Argument 2</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>Local variable 1</td>
<td></td>
</tr>
<tr>
<td>Local variable 2</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

Local variables can appear **in any order** and may not be contiguous.
```c
#include <stdio.h>

1 int main() {
2    int integer1;
3    int integer2;
4    int sum;
5    integer1 = 10;
6    integer2 = 20;
7    sum = integer1 + integer2;
8    printf("\nSum is %d", sum);
9    return 0;
}
```
#include <stdio.h>

1 int main() {
2    int integer1;
3    int integer2;
4    int sum;
5    integer1 = 10;
6    integer2 = 20;
7    sum = integer1 + integer2;
8    printf("\nSum is %d", sum);
9    return 0;
}
```c
#include <stdio.h>

int main() {
    int integer1;
    int integer2;
    int sum;
    integer1 = 10;
    integer2 = 20;
    sum = integer1 + integer2;
    printf("\nSum is %d", sum);
    return 0;
}
```

Stack after line 6

<table>
<thead>
<tr>
<th>Content of Stack</th>
<th>Stack after line 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer1</td>
<td>10</td>
</tr>
<tr>
<td>integer2</td>
<td>20</td>
</tr>
<tr>
<td>sum</td>
<td>XXX</td>
</tr>
</tbody>
</table>
```c
#include <stdio.h>

int main() {
    int integer1;
    int integer2;
    int sum;
    integer1 = 10;
    integer2 = 20;
    sum = integer1 + integer2;
    printf("\nSum is %d", sum);
    return 0;
}
```

Stack after line 7:

- `integer1`: 10
- `integer2`: 20
- `sum`: 30
#include <stdio.h>

int main() {
    int integer1;
    int integer2;
    int sum;
    integer1 = 10;
    integer2 = 20;
    sum = integer1 + integer2;
    printf("\nSum is %d", sum);
    return 0;
}
Introduction to Pointers

- Address of something is index into address-space array: \&integer1;
- Declaring a pointer to an integer
  ```c
  int *mypointer;
  ```
- Assigning an address to a pointer
  ```c
  mypointer = &integer1;
  ```
- Accessing data pointed to by pointer
  ```c
  *mypointer
  ```
Example with Pointers

```c
#include <stdio.h>

int main() {
    int integer1;
    int *mypointer;
    integer1 = 10;
    mypointer = &integer1;
    printf("Value is %d", integer1);
    printf("Value is %d", *mypointer);
    return 0;
}
```
```c
#include <stdio.h>

int main() {
    int integer1;
    int *mypointer;
    integer1 = 10;
    mypointer = &integer1;
    printf("\nValue is %d", integer1);
    printf("\nValue is %d", *mypointer);
    return 0;
}
```
Example with Pointers

```c
#include <stdio.h>

int main() {
    int integer1;
    int *mypointer;
    integer1 = 10;
    mypointer = &integer1;
    printf("\nValue is %d", integer1);
    printf("\nValue is %d", *mypointer);
    return 0;
}
```
Example with Pointers

#include <stdio.h>

int main() {
    int integer1;
    int *mypointer;
    integer1 = 10;
    mypointer = &integer1;
    printf("\nValue is %d", integer1);
    printf("\nValue is %d", *mypointer);
    printf("\nAddress is %p", mypointer);
    return 0;
}

Stack after line 5

integer1 10
mypointer 0xF4

CSE 303 - Winter 2010
```c
#include <stdio.h>

int main() {
    int integer1;
    int *mypointer;
    integer1 = 10;
    mypointer = &integer1;
    printf("\nValue is %d", integer1);
    printf("\nValue is %d", *mypointer);
    printf("\nAddress is %p", mypointer);
    return 0;
}
```
Readings

- Programming in C
  - Note: skim sections that look familiar to you! The book assumes NO programming background
  - Chapter 1: Introduction (you need to know that you may encounter different versions of C)
  - Chapter 2: Fundamentals
    - We will get back to compiling and linking later
  - Chapter 3: Compiling and Running
  - Chapter 11: Pointers (only pages 235-240)