lecture01: Intro, Pointers, and Memory

Sean Massung

Largely based on slides by Cinda Heeren
CS 225 UIUC

10th June, 2013
Welcome to CS 225!

- **Me:** Sean Massung (massung1)
  - Grad student in CS, undergrad here as well
  - Interested in information retrieval, natural language processing (and CS education!)

- **Course Staff:**
  - Tom Bogue (tbogue2)
  - Jason Cho (hcho33)
  - Joe Ciurej (ciurej2)
  - Chase Geigle (geigle1)

- **You...**
A good mixture of class years

Enrollment by Year, CS 225 Summer 2013

- Junior
- Sophomore
- Senior
- Grad
- Freshman
Not so much by degrees

Enrollment by Degree, CS 225 Summer 2013

- Bachelor
- None
- Doctor
- Master
Majors are diverse
What do you expect to get out of this class?
Please read the syllabus!

...and visit the home page.

We’ll go over any course infrastructure questions *tomorrow*
Week one is the busiest week

- Linux tutorial tomorrow night, Tuesday, June 11th at 7pm
- hw0 is out, due Wednesday, June 12th at 11:59pm in SVN
  - What’s SVN? How do I turn it in? Everything is explained in hw0 and the syllabus.
  - Make note: hw0 **must** be typed!
- Labs meet tomorrow (9am or 11am)
- mp1 released Wednesday (due Monday, June 17th)
- *Please* remember to read the syllabus before tomorrow’s lecture
Anything but “Hello World”

```cpp
/** @file main.cpp */
#include <iostream>
using namespace std;

int main(int argc, char** args)
{
    int n = 47;
    cout << "My favorite number is " << n << endl;
    return 0;
}
```

- Compiles with `g++ main.cpp -o myprog`
- Execute with `.\myprog`
- What happens?
Variables

- Each variable you declare has its own special place in memory

```c
int num = 225;
char grade = 'A';
double gpa = -6.8;
```

<table>
<thead>
<tr>
<th>addr</th>
<th>name</th>
<th>value</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x700</td>
<td>num</td>
<td>225</td>
<td>int</td>
</tr>
<tr>
<td>0x704</td>
<td>grade</td>
<td>A</td>
<td>char</td>
</tr>
<tr>
<td>0x705</td>
<td>gpa</td>
<td>-6.8</td>
<td>double</td>
</tr>
</tbody>
</table>

- Pointer types refer to another variable’s location in memory

```c
int* pnum = &num;
double* pgpa = &gpa;
```

<table>
<thead>
<tr>
<th>addr</th>
<th>name</th>
<th>value</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x70d</td>
<td>pnum</td>
<td>0x700</td>
<td>int*</td>
</tr>
<tr>
<td>0x711</td>
<td>pgpa</td>
<td>0x705</td>
<td>double*</td>
</tr>
</tbody>
</table>
How to use pointers?

- **Dereference** a pointer with the * operator to access the pointee
  
  ```
  *pgpa; // -6.8
  ```

- **&**, or the **address-of** operator, shows where its argument resides in memory
  
  ```
  &grade; // 0x704
  ```

- Change a pointee value:
  
  ```
  *pgpa = 5.0;
  
  cout << gpa << endl; // prints "5.0"
  ```

<table>
<thead>
<tr>
<th>addr</th>
<th>name</th>
<th>value</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x700</td>
<td>num</td>
<td>225</td>
<td>int</td>
</tr>
<tr>
<td>0x704</td>
<td>grade</td>
<td>A</td>
<td>char</td>
</tr>
<tr>
<td>0x705</td>
<td>gpa</td>
<td>-6.8 5.0</td>
<td>double</td>
</tr>
<tr>
<td>0x70d</td>
<td>pnum</td>
<td>0x700</td>
<td>int*</td>
</tr>
<tr>
<td>0x711</td>
<td>pgpa</td>
<td>0x705</td>
<td>double*</td>
</tr>
</tbody>
</table>
Stack memory vs heap memory

- **Stack memory**
  - ...is what we’ve used so far
  - Scratch space for the current function
  - When variables on the stack “go out of scope”, they automatically deallocate
  - Relatively small size compared to the heap

- **Heap memory**
  - We’ll explain how to use it on the next slide
  - Memory is allocated with *new* and *delete*
  - Note the *delete* function — memory on the heap must be explicitly freed by the user or else...
  - Memory leaks!

- Both are stored in RAM, just in separate areas
// 1
int x = 9;
int* ptr;
ptr = &x;
int** pptr = &ptr;
int* optr = ptr;

// 2
*optr = 47;

// 3
ptr = new int;
*ptr = 100;
int* fav = new int(7);

Stack Memory

<table>
<thead>
<tr>
<th>addr</th>
<th>name</th>
<th>value</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x716</td>
<td>x</td>
<td>9 47</td>
<td>int</td>
</tr>
<tr>
<td>0x71a</td>
<td>ptr</td>
<td>0x716 0xc00</td>
<td>int*</td>
</tr>
<tr>
<td>0x71e</td>
<td>pptr</td>
<td>0x71a</td>
<td>int**</td>
</tr>
<tr>
<td>0x721</td>
<td>optr</td>
<td>0x716</td>
<td>int*</td>
</tr>
<tr>
<td>0x725</td>
<td>fav</td>
<td>0xc04</td>
<td>int*</td>
</tr>
</tbody>
</table>

Heap Memory

<table>
<thead>
<tr>
<th>addr</th>
<th>name</th>
<th>value</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xc00</td>
<td>-</td>
<td>100</td>
<td>int</td>
</tr>
<tr>
<td>0xc04</td>
<td>-</td>
<td>7</td>
<td>int</td>
</tr>
</tbody>
</table>
Graphical pointer representation

```c
// 1
int x = 9;
int* ptr;
ptr = &x;
int** pptr = &ptr;
int* optr = ptr;
```
// 1
int x = 9;
int* ptr;
ptr = &x;
int** pptr = &ptr;
int* optr = ptr;

// 2
*optr = 47;
Graphical pointer representation

// 1
int x = 9;
int* ptr;
ptr = &x;
int** pptr = &ptr;
int* optr = ptr;

// 2
*optr = 47;

// 3
ptr = new int;
*ptr = 100;
int* fav = new int(7);
How does this make you feel?

Can you draw what is happening?

// 1
double* radius = new double(6.2);
double half = *radius / 2.0;
double* phalf = &half;
cout << "sum: " << *phalf + *radius << endl;

// 2
int* thing;
*thing = 77;

// 3
int* thing = NULL;
*thing = 47;
cout << "Can you see me?" << endl;
How about this?

Compiler errors:

```cpp
int* ptr = &47;    // NO! Can’t take address of rvalue
int* ptr = 34;    // STAHP! type mismatch (int* and int)
```

Runtime errors:

```cpp
int* myval;
int* otherval = new int(1);
int val = *myval + *otherval;    // myval is uninitialized!
```

Pointers can be tricky, even for experienced programmers

- A paper describes finding 56 pointer related bugs in a Linux file system implementation (see Defective Error/Pointer Interactions in the Linux Kernel)

Make sure you can draw a picture when dealing with pointers! (drawing a picture is actually general advice you should carry with you throughout the rest of this class and your future ones)