Variables

Bryce Boe

2012/09/05

CS32, Summer 2012 B

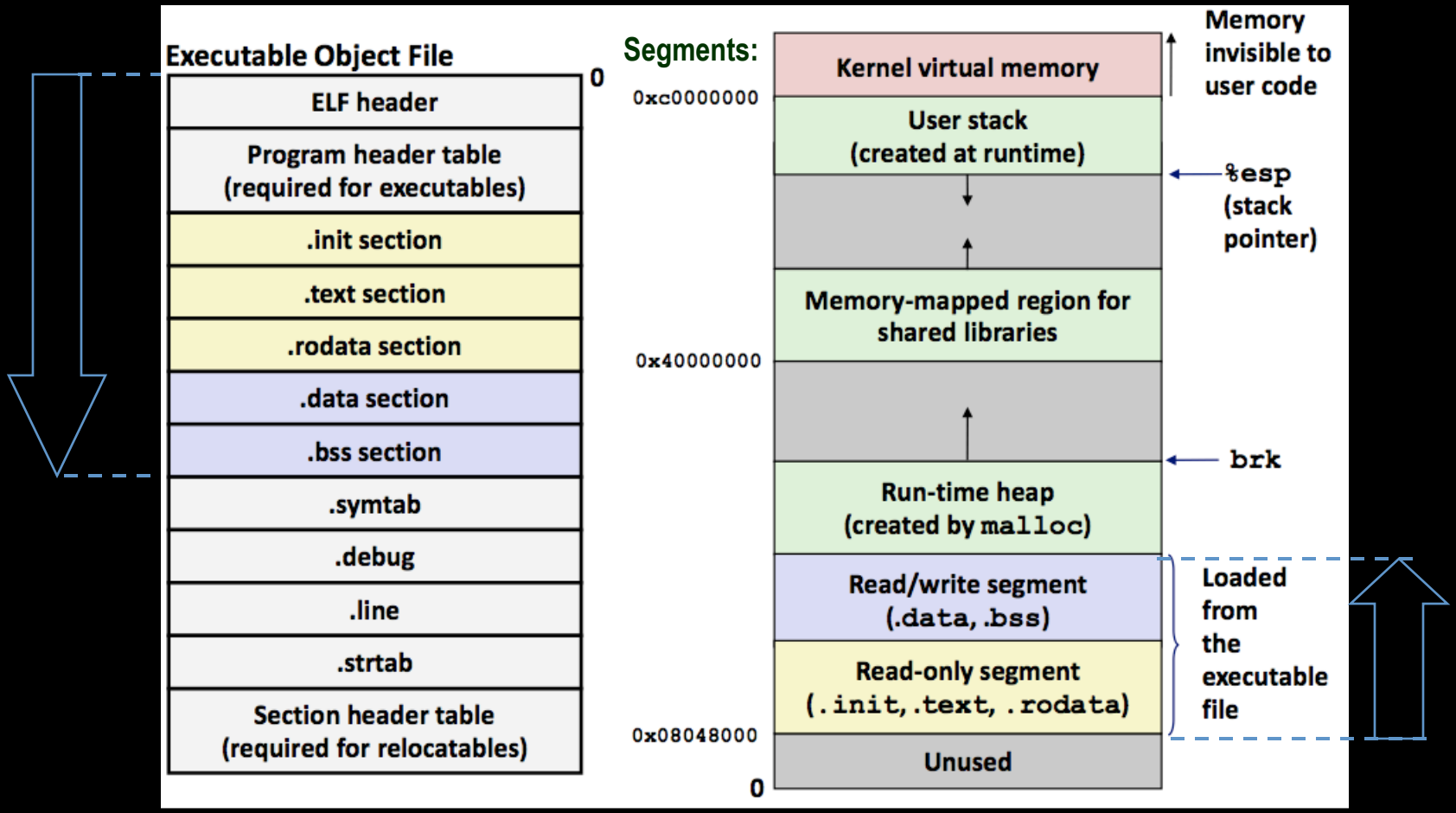
Overview

- Review of Variable Segment Locations
- Variable types and storage

Review

- The program's code is stored in the **text** segment (it is read-only)
- The value(s) of initialized global and static variables are stored in the **data** segment
- The value(s) of uninitialized global and static variables are stored in the **bss** segment
- The value(s) of local variables are stored on the **stack**
- The value(s) of dynamically allocated variables are stored on the **heap**

Sections of an executable file



Where is all the data stored?

```
int a1[] = {5, 6, 7, 8, 9};
char msg[] = “hello world”;
int main {
    static int call_count = 0;
    int i;
    return 0;
}
```

Where is all the data stored?

```
int a1[5];
char *msg;
int main {
    int blah[16];
    string *tmp = new string(“some message”);
return 0;
}
```

Where is all the data stored?

```
int a1[5];
int main {
Point p;
return 0;
}
```

```
class Point {
private:
    int a;
    int b;
    string *name;
};
```

How do initialized local arrays work?

```
void main2(int count) {
    if (count <= 0) return;
    int array[] = {0, 1, 2, 3, 4};
    main2(count – 1);
}
int main() {
    main2(3);
    return 0;
}
```

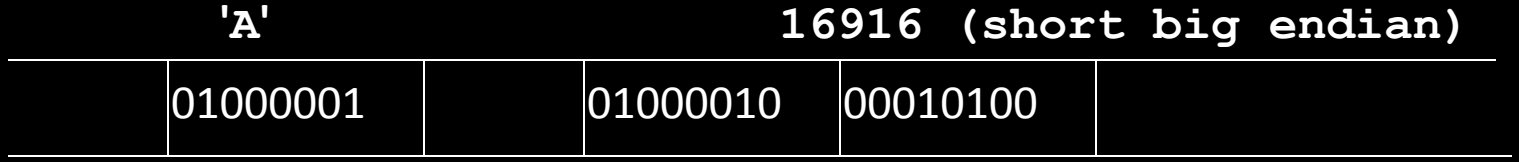
Function's Activation Record

- Stores:
  - Return value
  - Previous AR's ebp (base pointer)
  - Function parameters
  - Function local variables
- 1 activation record per function call (allows for recursion)

Why is mixing data and control on the stack not the best idea?

- Data
  - Variable values
- Control
  - Return value
  - Previous EBP
- Buffer overflow example

Variables and objects in memory



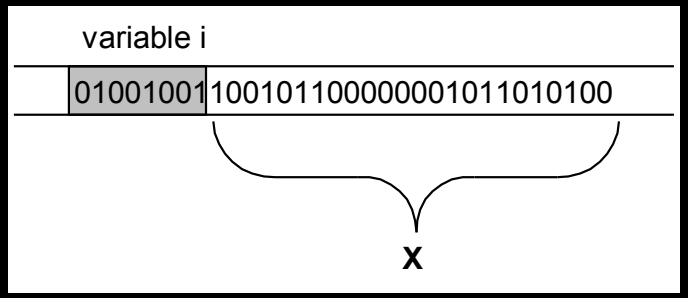
- Variables and data objects are data containers with names
- The value of the variable is the code stored in the container
- To evaluate a variable is to fetch the code from the container and interpret it properly
- To store a value in a variable is to code the value and store the code in the container
- The size of a variable is the size of its container

Variable Types and Storage

Overflow is when a data code is larger than the size of its container

- e.g., `char i;` // just 1 byte

```
int *p = (int*)&i; // legal
*p = 1673579060;
    // result if "big endian" storage:
```

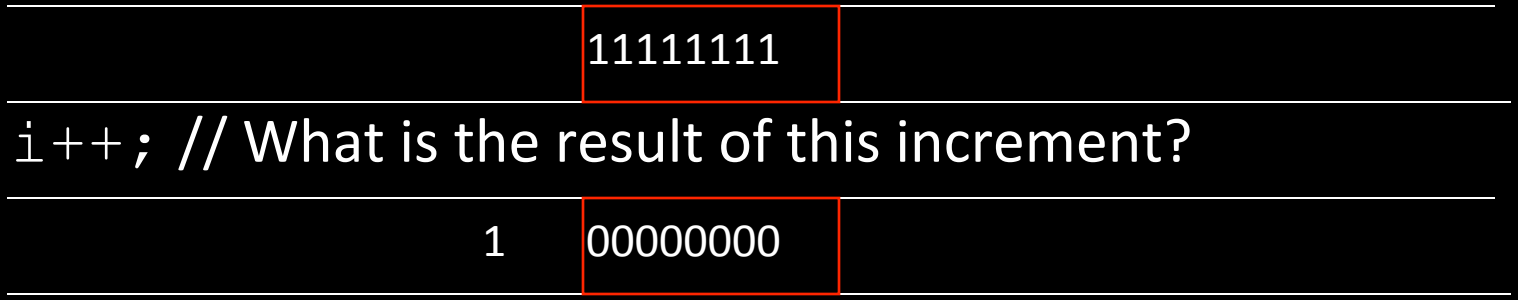


- If whole space (X) belongs to this program:
  - Seems OK if X does not contain important data for rest of the program's execution
  - Bad results or crash if important data are overwritten
- If all or part of X belongs to another process, the program is terminated by the OS for a memory access violation (i.e., segmentation fault)

More about overflow

- Previous slide showed example of "right overflow" – result truncated (also warning)

- Compilers handle "left overflow" by truncating too (usually without any warning)
  - Easily happens: `unsigned char i = 255;`

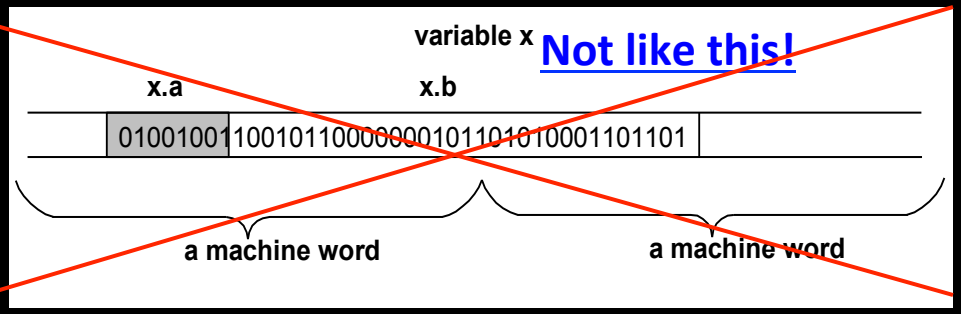


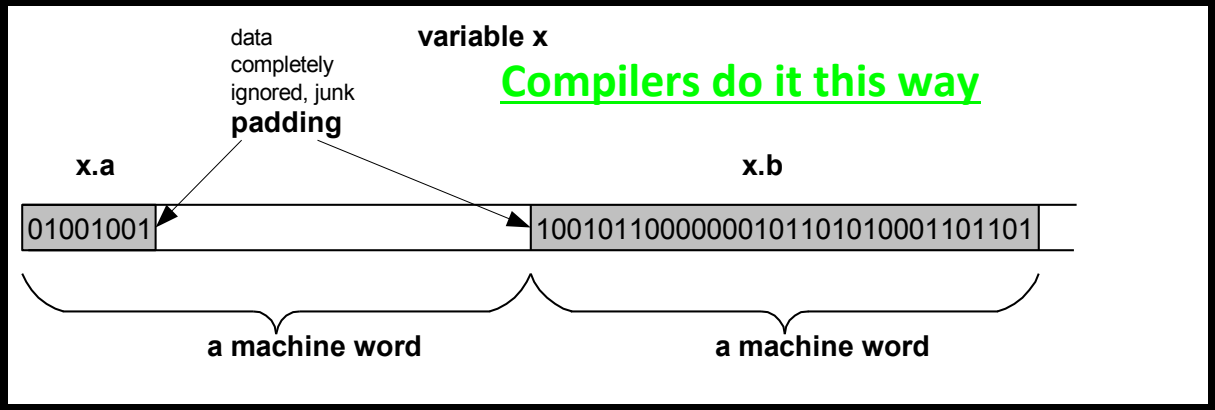
Placement & padding – word

- Compiler places data at word boundaries
  - e.g., word = 4 bytes
- Imagine:

```
struct {
   char a;
   int b;
} x;
```

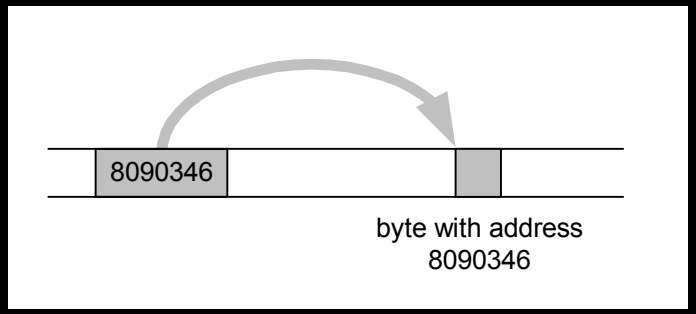
- Classes too

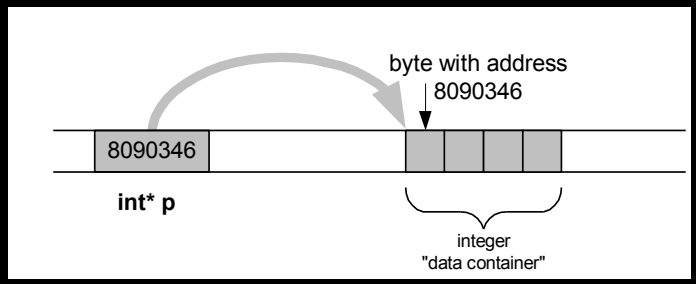


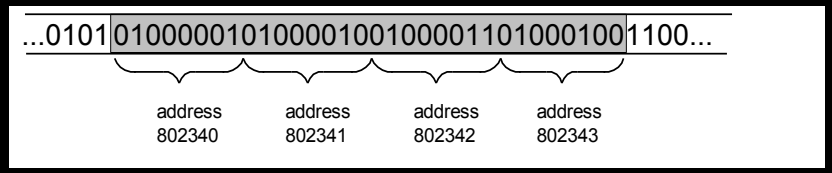


Pointers are data containers too

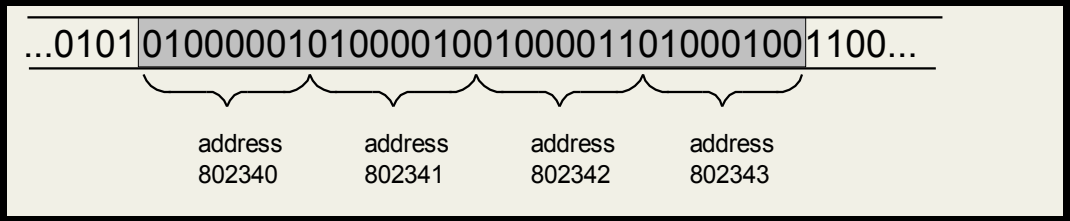
- As its *value* is a memory address, we say it "points" to a place in memory
- It points at just 1 byte, so it must "know" what data type starts at that address
  - How many bytes?
  - How to interpret the bits?
- Question: What is stored in the 4 bytes at addresses `802340..802343` in the diagram at right?
  - Continued next slide



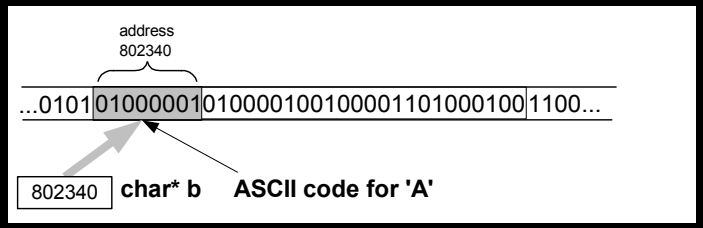


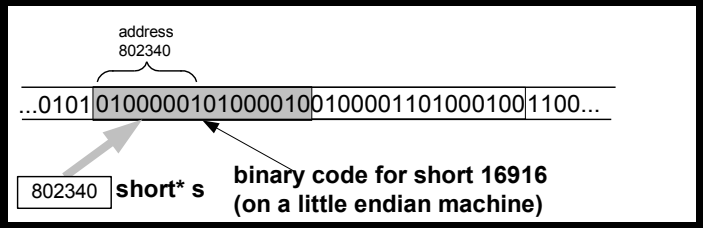


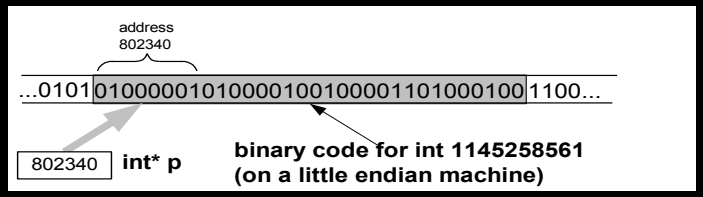
What is

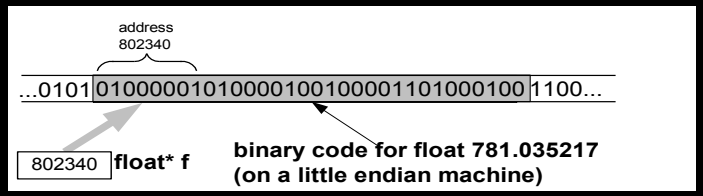


- Could be four chars: ‘A’, ‘B’, ‘C’, ‘D’
- Or it could be two shorts: 16961, 17475
  - All numerical values shown here are for a "little endian" machine (more about endian next slide)
- Maybe it’s a long or an int: 1145258561
- It could be a floating point number too: 781.035217









Beware: two different byte orders

- Matters to actual value of anything but chars
- Say: `short int x = 1;`
- On a big endian machine it looks like this:

  - Some Macs, JVM, TCP/IP "Network Byte Order"
- On a little endian machine it looks like this:

  - Intel, most communication hardware