CSE 303
Concepts and Tools for Software Development

Magdalena Balazinska
Winter 2010
Lecture 8 – Program structure, expressions, dangling pointers, printf/scanf

Where We Are

- Last time
  - Memory model for a process and the stack
  - Simple programs and introduction to pointers
- Today
  - Structure of a program, variable scope & storage
  - Passing arguments to functions
  - Left vs right expressions
  - Dangling pointers and NULL value
  - Formatted input and output

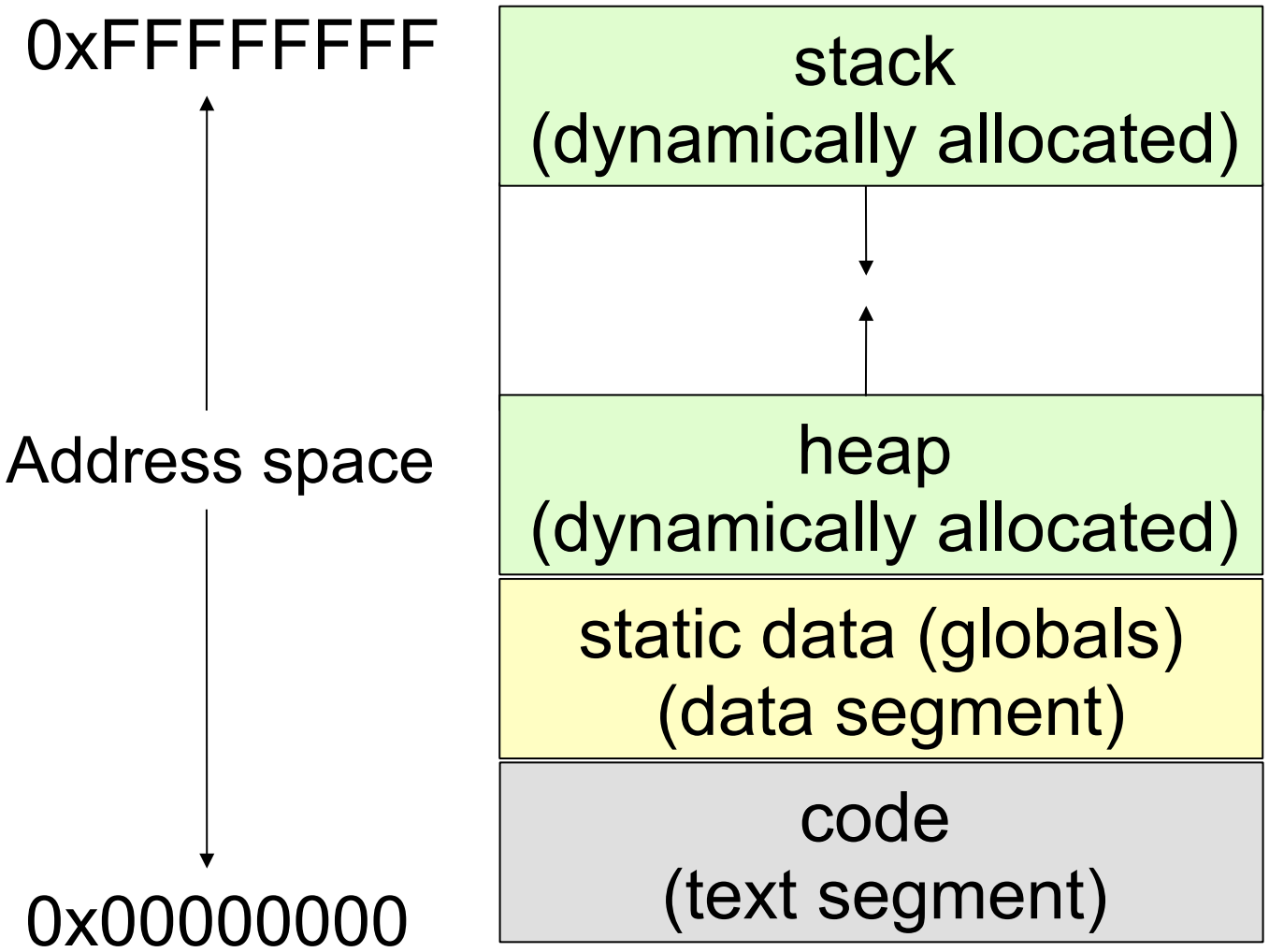
Structure of a C Program

```
// First include all header files (more later)
#include <stdio.h>

//Declare global variables (try to avoid them)
int global_int;

// Function must be defined before it is used
// Use function prototypes if needed
void my_function(int a, int b) { ... }
...
int main() { ... }
```

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

Storage Duration and Scope

- Scope
  - Global variables can be used in any function that follows their declaration
  - Local variables can only be used in the block where they are defined
- Storage class (lifetime)
  - Global vars exist for the duration of the program
  - Local vars exist while the block where they are defined is active
  - Static local vars retain their value between invocations

Passing Arguments to Functions

- In C, arguments are always passed by value
  - Function receives a copy of the argument
  - Changes to this copy will not affect original
- What if we want to modify argument?
  - Use pointers
- Example: `scope.c`
- Note: In C++, arguments can also be passed by reference (more later)

Passing Arguments to Functions

```
void main() {

    int i = 3;
    func(i);

}
```

Activation record for func

Return address
Info for returned val
3
...

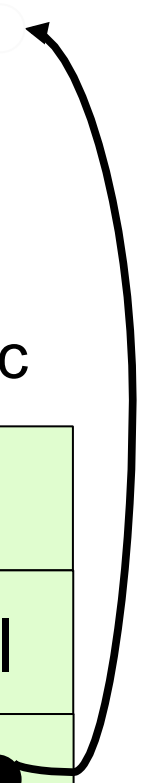
```
void main() {

    int i = 3;
    func(&i);

}
```

Activation record for func

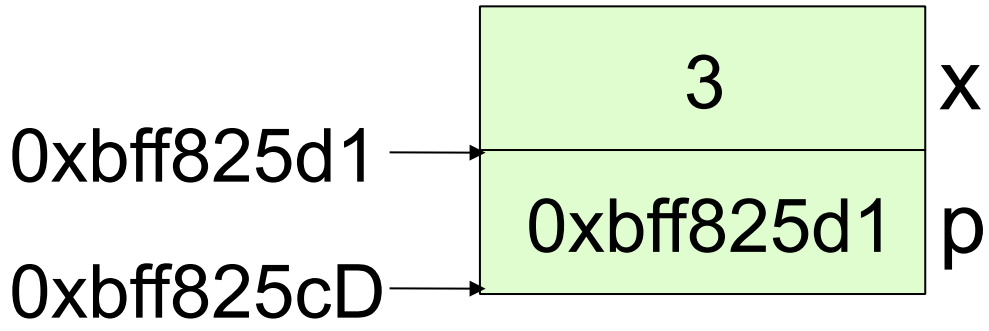
Return address
Info for returned val
0xFFFAACF4
...



Left vs right

- To “really get C”, it helps to understand the difference between the left side and the right side of an assignment
  - Law #1: Left-expressions evaluated to locations (addresses)
  - Law #2: Right expressions evaluated to values
  - Law #3: Values include addresses
- Examples

```
int x = 3;
int *p;
p = &x;
```



Left vs Right (continued)

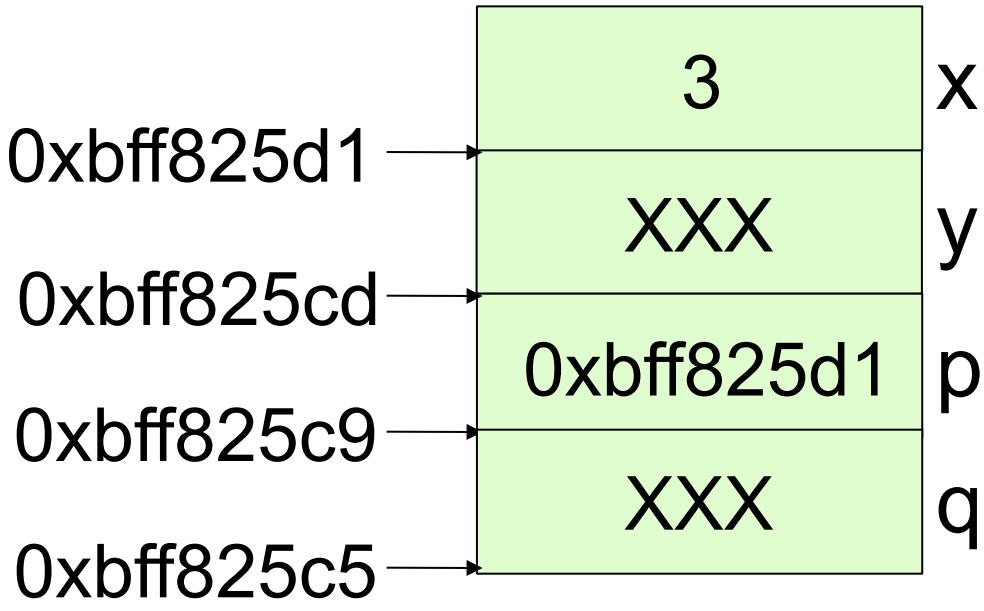
- Key difference is the “rule” for variables
  - As left-expression, a variable is a location and we are done
  - As right-expression, a variable gets evaluated to the content of its location and then we are done
- Note: this is true in Java as well

Examples Left vs Right

- Examples

```
int x = 3;
int y;
int *p;
int *q;
p = &x;
```

```
q = p;
q = &y;
*q = *p;
```

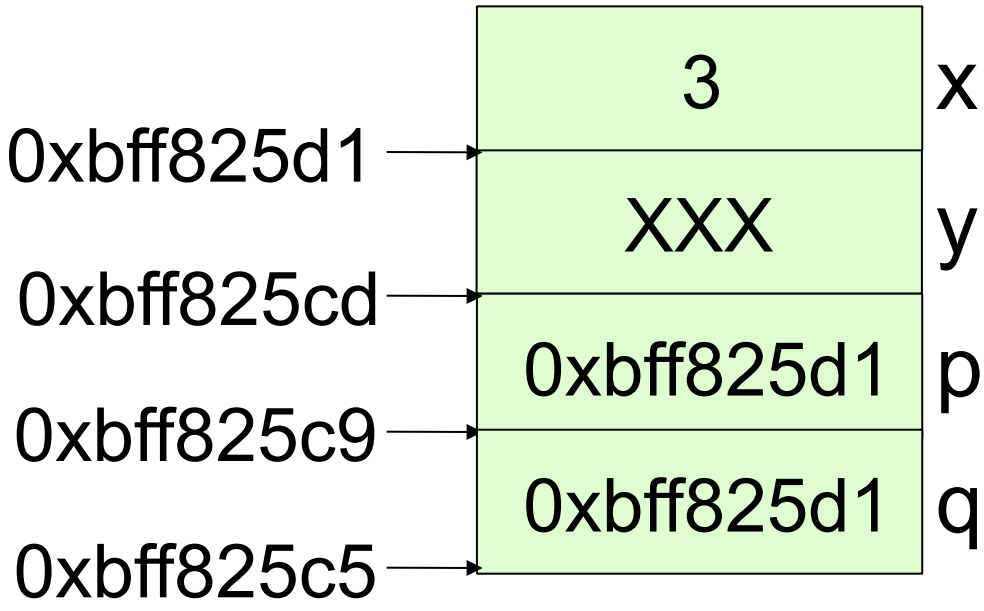


Examples Left vs Right

- Examples

```
int x = 3;
int y;
int *p;
int *q;
p = &x;
q = p;
```

```
q = &y;
*q = *p;
```

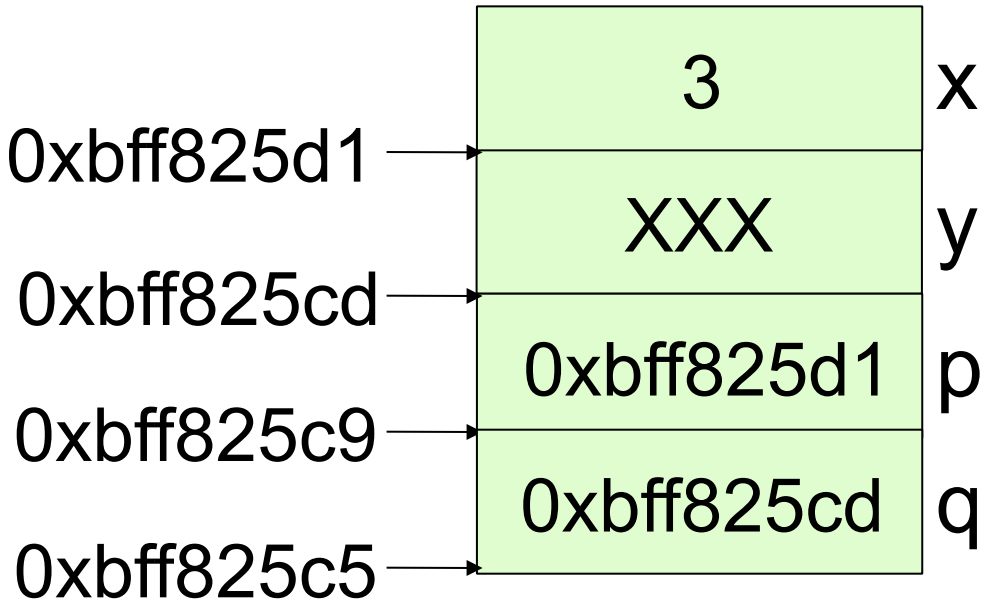


Examples Left vs Right

- Examples

```
int x = 3;
int y;
int *p;
int *q;
p = &x;
q = p;
q = &y;
```

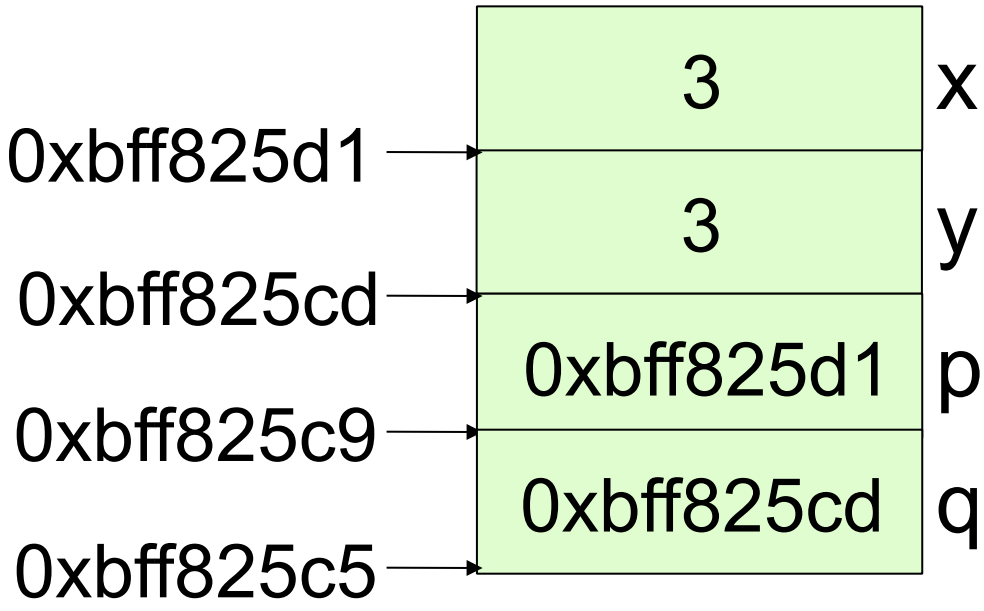
```
*q = *p;
```



Examples Left vs Right

- Examples

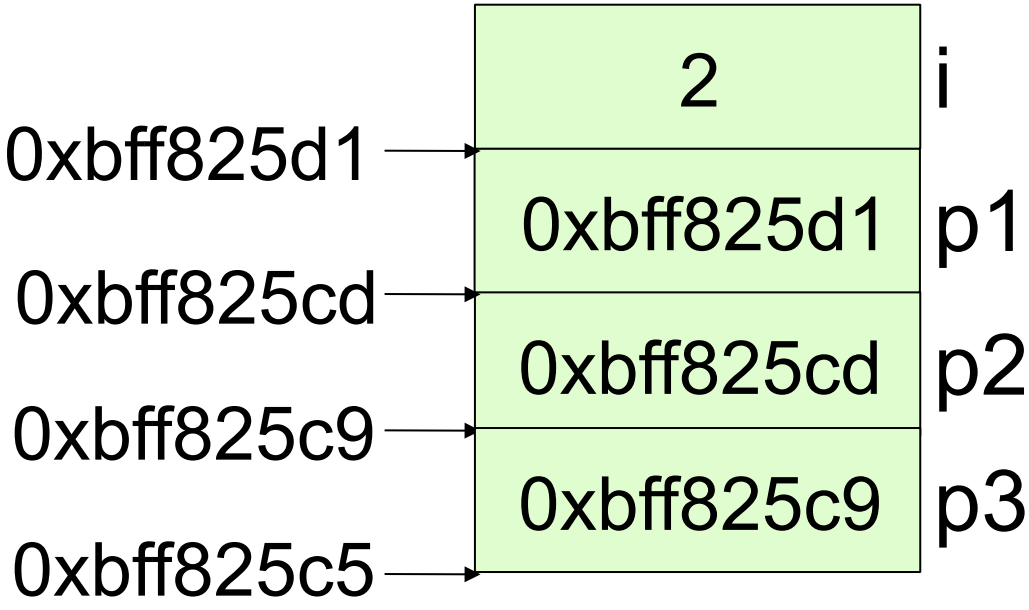
```
int x = 3;
int y;
int *p;
int *q;
p = &x;
q = p;
q = &y;
*q = *p;
```



Pointers to pointers

```
int i=2;
int *p1;
p1 = &i;
int **p2;
p2 = &p1;
int ***p3;
p3 = &p2;
**p2 = 5;
***p3 = 10;
```

Both change the value of i



Additional examples in `pointer-to-pointer.c`

NULL Value

- The value of a pointer is an address
- A pointer can also hold the value 0 or NULL
- A pointer with the value NULL points to nothing
- NULL is a symbolic constant defined in `stddef.h` (included by `stdio.h`)
- Example: `null-pointer.c`

A Note About Boolean Type

- In C, any integer type may be used to represent a boolean value
  - Anything but 0 (or NULL) is true
  - 0 and NULL are false
- C99 introduces an "extended integer" type named bool and boolean values true and false (you must include `stdbool.h`)
- Example: `bool.c`

Dangling Pointers

- Pointer initialized to address of piece of data
- Storage for data is reclaimed because
  - Lifetime of variable ends
  - Or explicitly deallocated (when using the heap)
- The pointer is left "dangling"
  - Points to undefined location
- If you're lucky... result will be KABOOM!!
- Frequently, causes subtle and silent bugs!
- Example: `dangling.c`

Formatted Input and Output

- What we already know
  - Input and output is performed with streams
  - Streams are just sequences of bytes
  - stdin connected to keyboard
  - stdout and stderr connected to screen
- Formatted output: printf
- Formatted input: scanf

Formatted Input and Output

- `printf("format string", v1, v2, ...);`
- `scanf("format string", v1, v2, ...);`
- Basic formats
  - %d: int
  - %f: float, double
  - %c: char
  - %s: char* (strings)
  - %e: scientific notation
- Examples: `format.c`
- Also take a look at `fileIO.c` (needed for hw3)

Readings

- Programming in C
  - Skim Chapters 4, 5, 6, and 8
  - Chapter 11 Pointers and Functions (pp 254-259)
  - Chapter 16 Formatted I/O (pp 348-359)