

Week 5

C Programming

06 February 2019

CS 35L Lab 4

Jeremy Rotman

Announcements

- Assignment #4 is due Saturday by 11:55pm
- For Assignment #10
 - ◆ **Email me to tell me what story you are choosing**
 - ◆ [Here is the link to see what stories people have signed up for already](#)
 - Choose a story at least one week before you present
 - ◆ [Here is the link to sign up to present](#)
 - If you haven't signed up do it
 - ◆ I need article decisions from David Zhao and Angela Zhang

Questions?

Outline

→ C

→ Homework 4

C Basic Data Types

→ char

- ◆ Holds a character
- ◆ 1 byte

→ int

- ◆ Holds integer numbers
- ◆ Usually 4 bytes

→ unsigned int

- ◆ Holds positive integer numbers
- ◆ Usually 4 bytes

C Basic Data Types

→ float

- ◆ Holds floating point numbers (decimals)
- ◆ 4 bytes

→ double

- ◆ Holds floating point numbers at higher precision
- ◆ 8 bytes

→ void

→ No boolean

Pointers

Pointers

→ Variables that store memory addresses

- ◆ Generally they store the memory address at which a variable is stored

→ Declaration

- ◆ `<var_data_type> *<ptr_name>`

```
int* ptr;           // declares ptr as a pointer to an int variable
```

```
int var = 6;        // defines an int variable var
```

```
ptr = &var;         // sets the pointer equal to the memory address of var
```

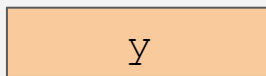
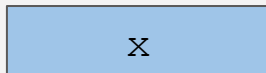
→ Dereference (access the value stored in memory) with *

```
*ptr = 15;          // sets var's value to 15
```


Examples

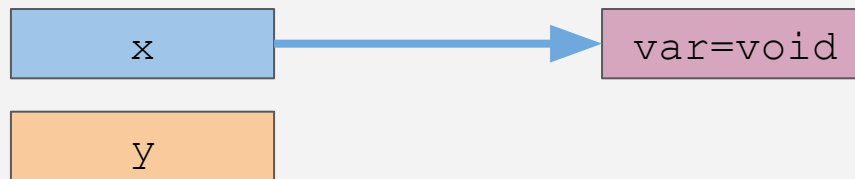
`int* x;`

`int* y;`



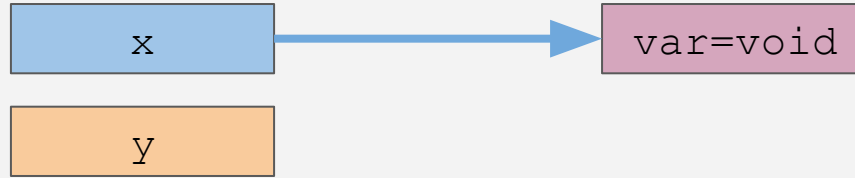
Examples

```
int var;  
x = &var;
```



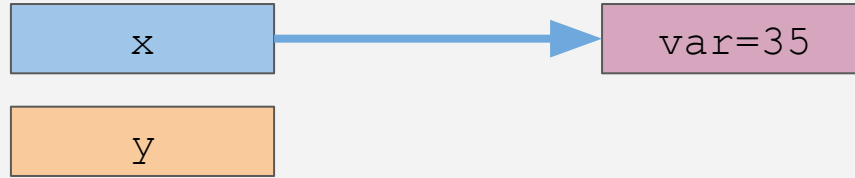
Examples

$*X = 35$



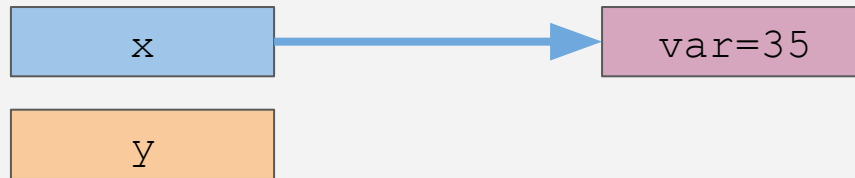
Examples

*X = 35



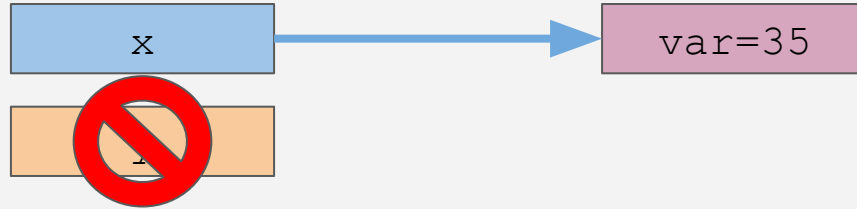
Examples

$*y = 100$



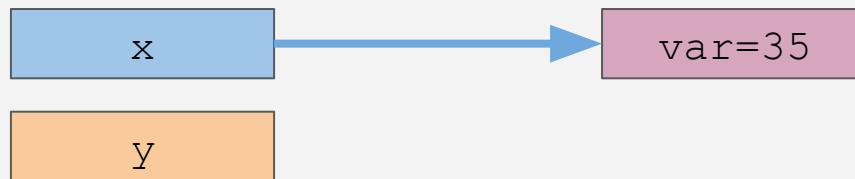
Examples

$*y = 100$



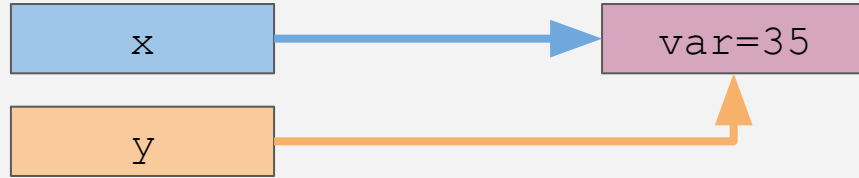
Examples

$y = x$



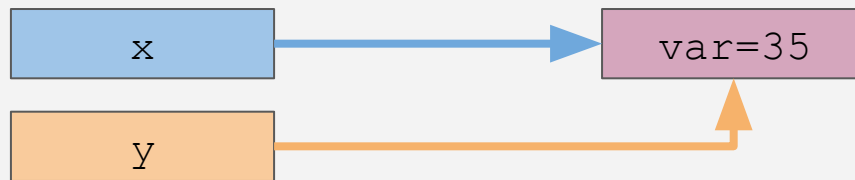
Examples

$y = x$



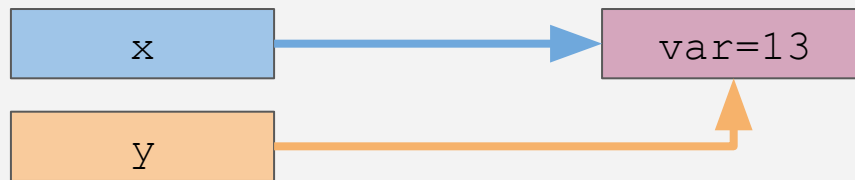
Examples

$*y = 13$



Examples

$*y = 13$



More Pointers

→ Can a pointer point to a pointer?

More Pointers

→ Can a pointer point to a pointer?

- ◆ Yes, they can
- ◆ Pointception

```
char c = 'Z'
```

```
char* cPtr = &c
```

```
char** cPtrPtr = &cPtr
```

Function Pointers

- Functors
- Allows you to pass a function into another function
- Why is this useful?

Function Pointers

- Functors
- Allows you to pass a function into another function
- Why is this useful?
 - ◆ It allows you to create a more generalized function
 - ◆ Conditionals can allow you to use a different function within the same function

Functors

→ Declaration

```
double (*func_ptr) (double, double);  
func_ptr = &pow;    // func_ptr now points to pow()
```

→ Usage

```
double result = (*func_ptr)(1.5, 2.0);
```

Functor Example: qsort

```
void qsort(void* base, size_t num, size_t size,  
int (*comparator) (const void*, const void*))
```

- ➔ The fourth argument of qsort is a functor to a function to use to compare the elements of base
 - ◆ This allows qsort to work on any data type provided you give it a valid comparator for the data type
 - ◆ Comparator returns
 - <0: sort the first element before the second
 - 0: means they are equal
 - >0: sort the second element before the first
- ➔ This can be passed into qsort as the function name

Structs

- C does not have classes
- But structs can package related data
 - ◆ Kind of like a “lite” version of a class

```
struct Student {  
    char name[64];  
    char UID[9];  
    int age;  
    int year;  
    double panicLevel;  
};  
Struct Student s;
```

C Structs vs. C++ Classes

C Structs	C++ Classes
-----------	-------------

C Structs vs. C++ Classes

C Structs	C++ Classes
No member functions	Member functions
No access specifiers	Access specifiers that are private by default
No defined constructors	Must at least have default constructor

Dynamic Memory

- C allows you to allocate memory at run time
 - ◆ This gets allocated on the heap
- `void* malloc(size_t size);`
 - ◆ Allocates *size* bytes and returns a pointer to the allocated memory
- `void* realloc(void* ptr, size_t size);`
 - ◆ Changes the size of the memory block pointed to by *ptr* to *size* bytes
- `void free(void* ptr);`
 - ◆ Frees the block of memory pointed to by *ptr*

Reading and Writing Characters

- **int getchar();**
 - ◆ Returns the next character from stdin
- **int putchar(int *character*);**
 - ◆ Writes *character* to the current position in stdout
- **int fprintf(FILE* *stream*, const char* *format*, ...);**
 - ◆ Print the c string *format* to *stream*
 - Stream is either a pointer to a file, or one of stdin, stdout, stderr
- **int fscanf(FILE* *stream*, const char* *format*, ...);**
 - ◆ Read the c string *format* from *stream*

Formatted I/O

- `fprintf` and `fscanf` rely on file pointers and formatted strings
- File pointers
 - ◆ Generated by opening files
 - ◆ `file* fp = fopen("file.txt", "r")`
 - `fp` is now a pointer to the opened file `file.txt` with read permission
- Formatted String
 - ◆ Allows a string to be followed by variables to be placed into a string
 - ◆ `fprintf(fp, "This class, %s, has %i students", class_name, n_students)`
 - ◆ For `fscanf`, the formatted string will allow you to place separate pieces of the string into variables

Homework 4

- Write a C function, `frobcmp`
 - ◆ Compare two objects byte-by-byte
 - In the style of `memcmp`
 - ◆ Return an integer, denoting which of the two arguments should come lexicographically before the other
 - ◆ The catch being that the two arguments are frobnicated
 - Frobnication is an obfuscation technique
 - Encoded by an XOR with `0x2A` (hexadecimal 42)
 - ◆ You will have to manually do this byte-by-byte, because we do not want the deobfuscated objects to appear in memory

Homework 4

- Use `frobcmp` to write another function, `sfrob`
- `sfrob` reads frobnicated lines from `stdin` and outputs the sorted (still frobnicated) lines to `stdout`
- You can use `<stdio.h>` functions to do IO
 - ◆ This includes the things in the earlier slides
- Use `malloc`, `realloc`, and `free` to ensure you have enough space to store all of the input
- Use `qsort` to sort the data
- Your function must work on continually growing input
 - ◆ Do not stop until you've hit an EOF

Homework 4 Hints

- Remember a pointer to a pointer is also essentially an array of arrays (`char** arr`), this can help you store strings
- Use correct casting when passing `frobcmp` to `qsort`
 - ◆ `qsort` expects the type `void*` because it's generalized
- Use `realloc` to reallocate memory for every string, and the array of strings
- Use `exit` to exit with error
- Frobnicated newlines become spaces
 - ◆ Thus each string is separated by spaces

Questions?