C Programming

Homework 5

Basic Data Types

- **int**
  - Holds integer numbers
  - Usually 4 bytes
- **float**
  - Holds floating point numbers
  - Usually 4 bytes
- **double**
  - Holds higher-precision floating point numbers
  - Usually 8 bytes (double the size of a float)
- **char**
  - Holds a byte of data, characters
- **void**

Pretty much like C++ basic data types, but NO **bool** before C99

Pointers

- Variables that store memory addresses

Declaration

- <variable_type> *<name>;
  - int *ptr;   //declare ptr as a pointer to int
  - int var = 77;   // define an int variable
  - ptr = &var;   // let ptr point to the variable var

Dereferencing Pointers

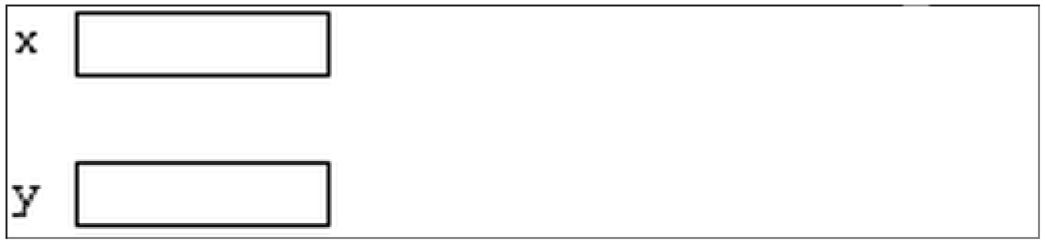
- Accessing the value that the pointer points to

- Example:
  - double x, *ptr;
  - ptr = **&**x; // let ptr point to x
  - *****ptr = 7.8; // assign the value 7.8 to x

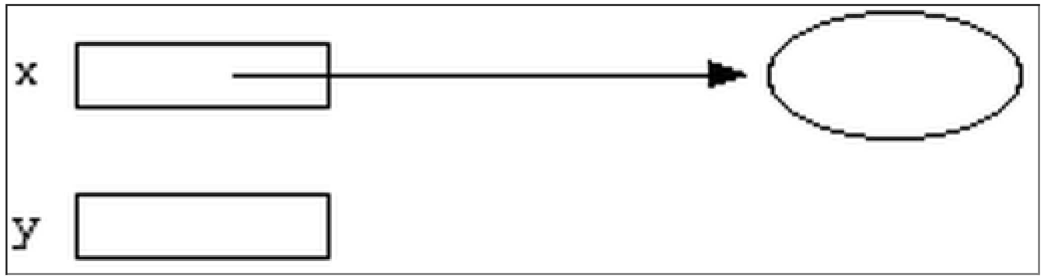
Pointer Example

int *x;

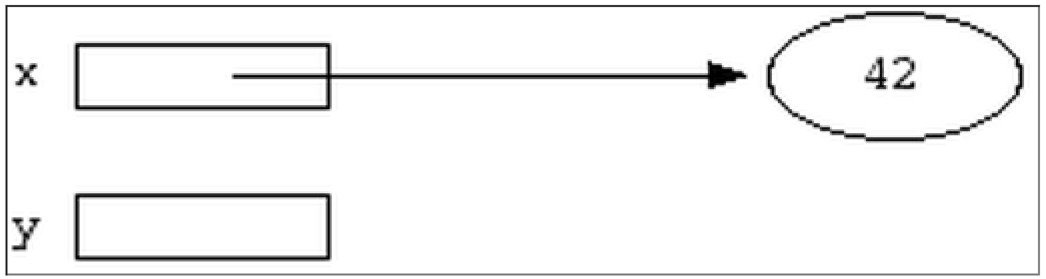
int *y;



int var;   x = &var;

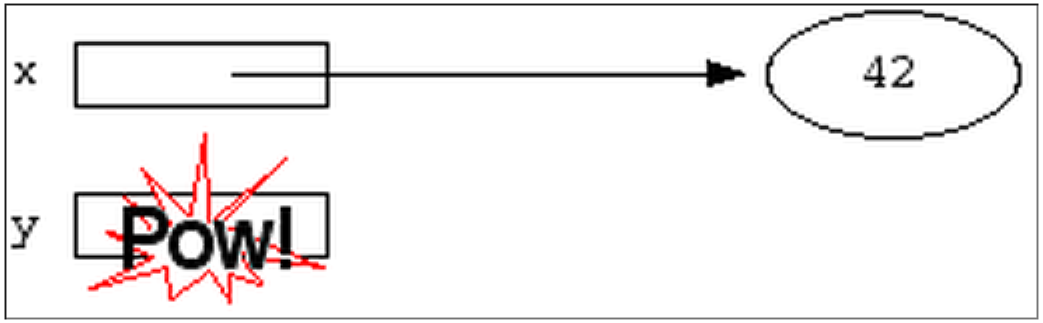


*x = 42;

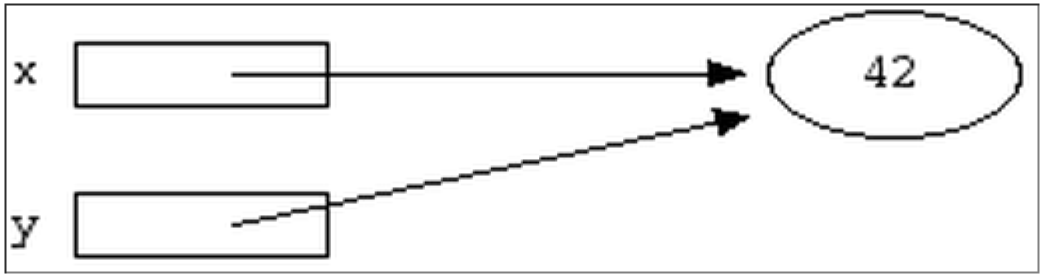


Pointer Example

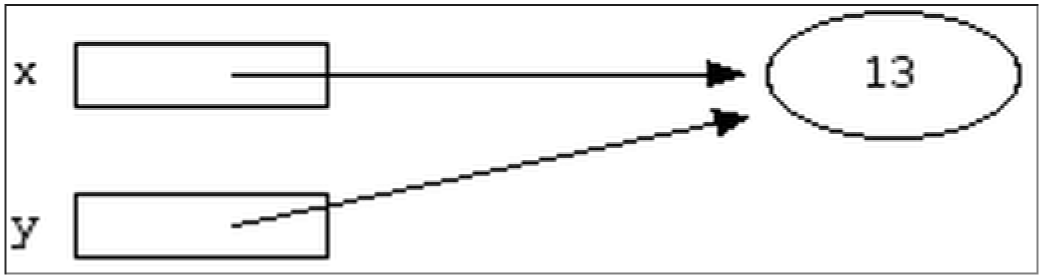
*y = 13;



y = x;

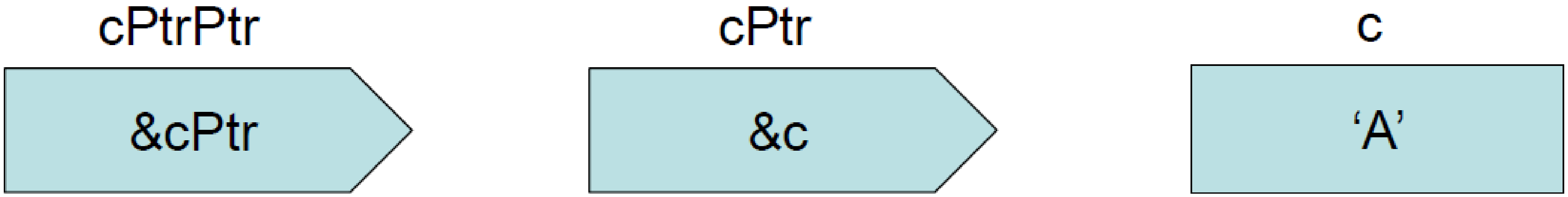


*x = 13; or
*y = 13;



Pointers to Pointers

**cPtrPtr = &cPtr   *cPtr = &c   char c = ‘A’



Pointers to Functions

- Also known as: **function pointers** or **functors**
- Goal: write a sorting function
  - Has to work for ascending and descending sorting order + other
- How?
  - Write multiple functions
  - Provide a flag as an argument to the function
  - Polymorphism and virtual functions
  - Use function pointers!!

Pointers to Functions

- User can pass in a function to the sort function
- Declaration
  - double (*func_ptr) (double, double);
  - func_ptr = [&]pow; // func_ptr points to pow()
- Usage
  - // Call the function referenced by func_ptr

    double result = (*func_ptr)(1.5, 2.0);
  - // The same function call

    result = func_ptr(1.5, 2.0);

qsort Example

```
#include <stdio.h>                    /* printf */
#include <stdlib.h>                   /* qsort */

int compare (const void * a, const void * b)
 {
          return ( *(int*)a - *(int*)b );
}

int main ()
{
          int values[] = { 40, 10, 100, 90, 20, 25 };
          qsort (values, 6, sizeof(int), compare);
          int n;
          for (n = 0; n < 6; n++)
                    printf ("%d ",values[n]);
          return 0;
}
```

Return Value	Meaning
<0	The element pointed to by a goes before element pointed to by b
0	The element pointed to by a is equivalent to the element pointed to by b
>0	The element pointed to by b goes before the element pointed to by a

Structs

- No classes in C
- Used to package related data (variables of different types) together
- Single name is convenient

```
struct Student {
        char name[64];
        char UID[10];
        int age;
        int year;
};
struct Student s;
```

```
typedef struct {
        char name[64];
        char UID[10];
        int age;
        int year;
  } Student;
  Student s;
```

C structs vs. C++ classes

- C structs cannot have member functions
- There's no such thing as access specifiers in C
- C structs don't have constructors defined for them

- C++ classes can have member functions
- C++ class members have access specifiers and are **private** by default
- C++ classes must have at least a default constructor

Dynamic Memory

- Memory that is allocated at runtime
- 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/Writing Characters

- **int getchar();**
  - Returns the next character from stdin
- **int putchar(int character);**
  - Writes a character to the current position in stdout

Formatted I/O

- int fprintf(FILE * fp, const char * format, …);
  - FILE *fp can be either:
    - A file pointer
    - stdin, stdout, or stderr
  - The format string
    - int score = 120; char player[] = “Mary”;
    - printf(**“%s has %d points.\n”, player, score**);

Compiling a C program

- gcc –o FooBarBinary -g foobar.c
  - gcc is the name of the compiler
  - The –o option indicates the name of the binary/program to be generated
  - The –g option includes symbol and source-line info for debugging
  - foobar.c is the source code to be compiled

Homework 5

- Write a C program called *sfrob.c*
  - Input: records/words separated by spaces
    - Each byte in input is frobnicated (XOR’d w/ 42)
  - Output: frobnicated words in sorted ASCII order

- One way: unfrobnicate → sort → frobnicate
  - `printf` 'ler nem' | ./sfrob
  - Read the records: l e r n e m
  - frobnicate(ler) = fox, frobnicate(nem) = dog (use `memfrob`)
  - dog < fox => Output: nem ler
- Problem: memfrob does transformation in place
  - Memory will temporarily include unfrobnicated data
  - Need to make sure no decoded data is written to memory

Homework 5

- Read stdin byte-by-byte **(getchar)**
  - Consists of records that are space-delimited
- Each byte is frobnicated (XOR'd with 42)
  - Sort records **without decoding** (**qsort, frobcmp**)
  - Output frobnicated result to stdout **(putchar)**
- Error checking and reporting (**fprintf**)
- Dynamic memory allocation (**malloc, realloc, free**)

Homework Hints

- Start as soon as possible
- Use ***gdb***
- Use *exit*, not *return* when exiting with error
- 1-D vs. 2-D array
- Test your code with od –ta