CS 35L Software Construction Lab
Week 4 – C Programming

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**

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

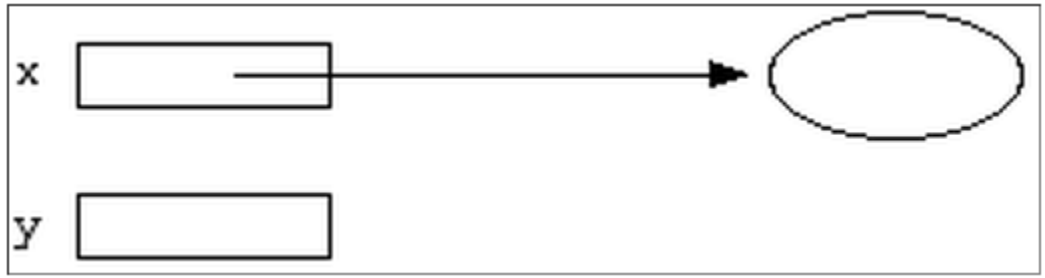
- Example:
  - double x, *ptr;
  - ptr = **&**x; // let ptr point to x
  - *****ptr = 7.8; // assign the value 7.8 to x

Pointer Example

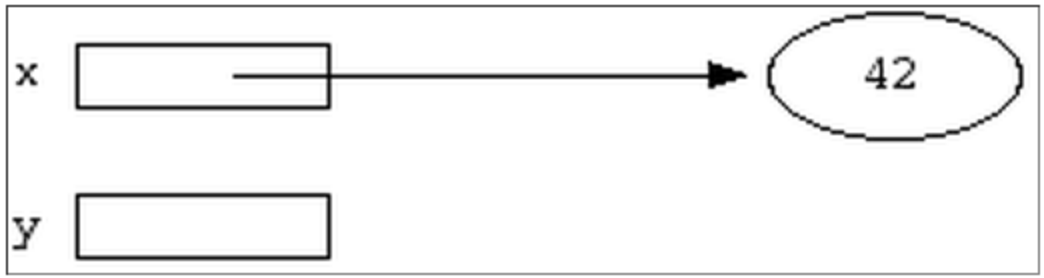
int *x;

int *y;

int var; x = &var;

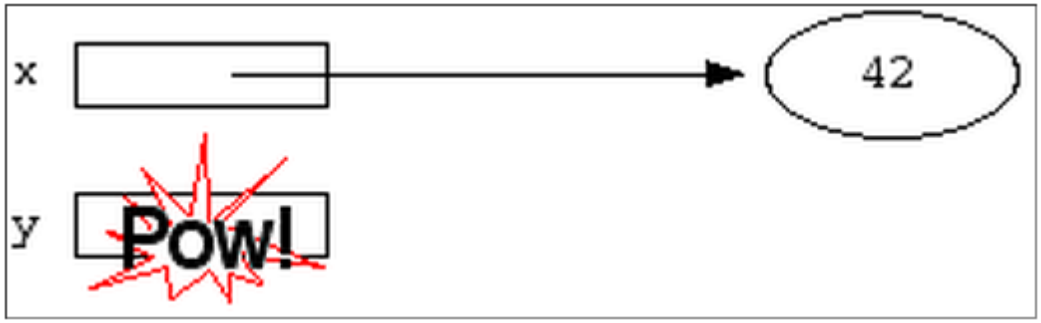


*x = 42;

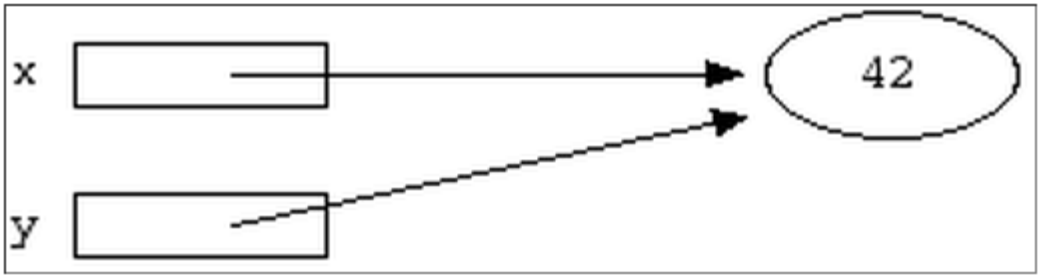


Pointer Example

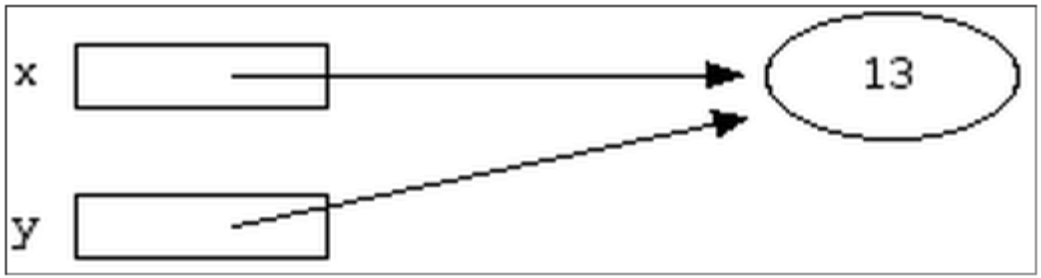
*y = 13;



y = x;

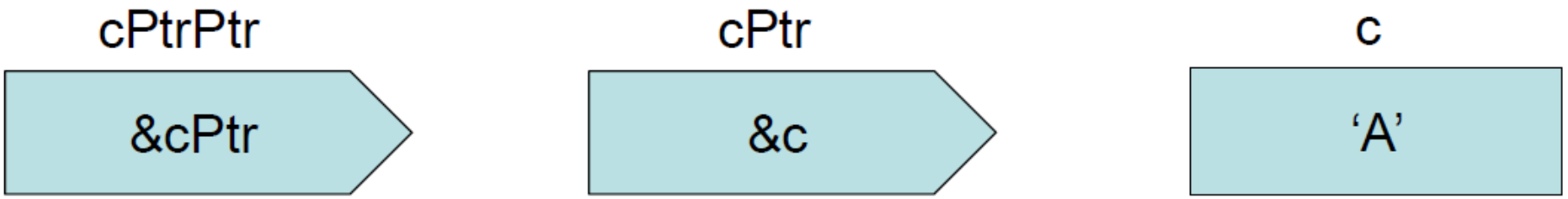


*x = 13; or
*y = 13;



Pointers to Pointers

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



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
  - 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()
  - func_ptr = pow; // an alternative way
- Usage
  - // Call the function referenced by func_ptr

    double result = (*func_ptr)(1.5, 2.0);
  - // an alternative way

    double result = func_ptr(1.5, 2.0);

qsort Example

```
#include <stdio.h>
#include <stdlib.h>

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;
}
```

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/structs 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
  - Allocates space in the heap during the execution of the program.
  - does not initialize the allocated memory . It carries garbage value.
  - returns null pointer if it cannot allocate requested amount of memory.

void* calloc (size_t num, size_t size);

- Allocates a block of memory for an array of num elements, each of them size bytes long
- initializes all its bits to zero.

void *realloc (void *ptr, size_t size);

- Changes the size of the memory block pointed to by *ptr* to *size* bytes
- expanding or contracting the existing area pointed to by ptr, if possible.
- OR allocating a new memory block of size new_size bytes, copying memory area with size equal the lesser of the new and the old sizes, and freeing the old block.

void free (void *ptr);

- Frees the block of memory pointed to by *ptr*
- If a null pointer is passed as argument, no action occurs.

```c
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    int *pa = malloc(10 * sizeof *pa); // allocate an array of 10 int
    if(pa) {
        printf("%zu bytes allocated. Storing ints: ", 10*sizeof(int));
        for(int n = 0; n < 10; ++n)
            printf("%d ", pa[n] = n);
    }

    int *pb = realloc(pa, 1000000 * sizeof *pb); // reallocate array to a larger size
    if(pb) {
        printf("\n%zu bytes allocated, first 10 ints are: ", 1000000*sizeof(int));
        for(int n = 0; n < 10; ++n)
            printf("%d ", pb[n]); // show the array
        free(pb);
    } else { // if realloc failed, the original pointer needs to be freed
        free(pa);
    }
}
```

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, …);
- int fscanf(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”;
    - fp = fopen(“file.txt”, “w+”)
    - fprintf(fp, **“%s has %d points.\n”, player, score**);

Homework 5

- Write a C program called *sfrob*
  - Reads stdin byte-by-byte **(getchar)**
    - Consists of records that are newline-delimited
  - Each byte is frobnicated (XOR with dec 42)
    - Sort records without decoding (**qsort, frobcmp**)
    - Output result in frobnicated encoding to stdout **(putchar)**
  - Dynamic memory allocation (**malloc, realloc, free**)

Example

- Input: printf 'sybjre obl’
  - $ printf 'sybjre obl**\n**' | ./sfrob
- Read the records: sybjre, obl
- Compare records using *frobcmp* function
- Use *frobcmp* as compare function in *qsort*
- Output: obl

  sybjre

Homework Hints

- Array of pointers to char arrays to store strings (char ** arr)
- Use the right cast while passing frobcmp to qsort
  - cast from void ** to char ** and then dereference because frobcmp takes a char *
- Use realloc to reallocate memory for every string and the array of strings itself, dynamically
- Use *exit*, not *return* when exiting with error