CS35L Software Construction Laboratory

Lab 6: Nandan Parikh

Week 4; Lecture 1

QUESTIONS

PRESENTATIONS

- Make sure to give your preferences on this link
  - Doc
- 7 minutes long + 3 minutes for questions (max)
- Topic must be according to guidelines

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

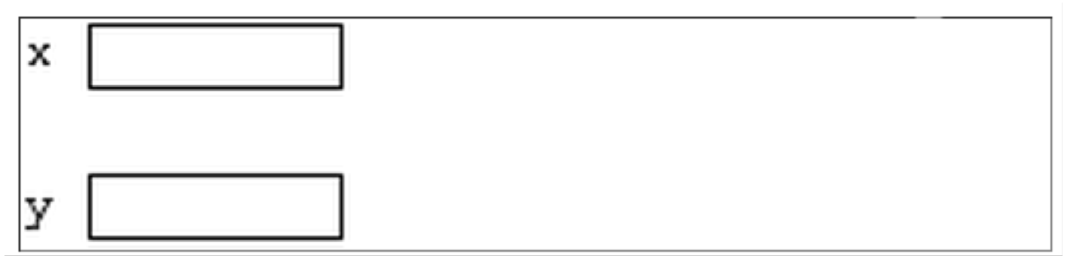
- Accessing the value that the pointer points to

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

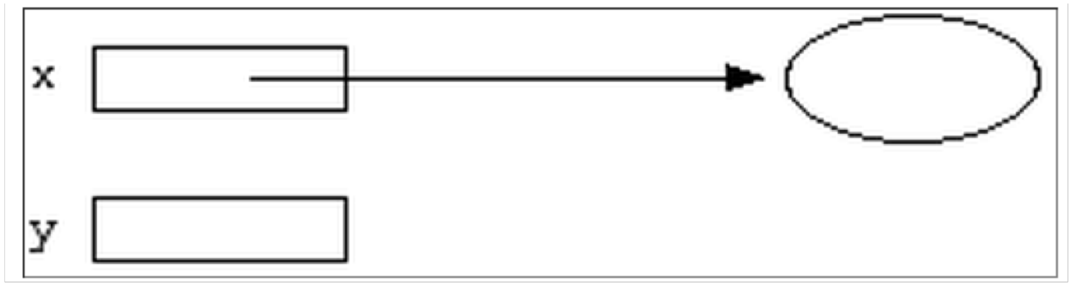
Pointer Example

int *x;

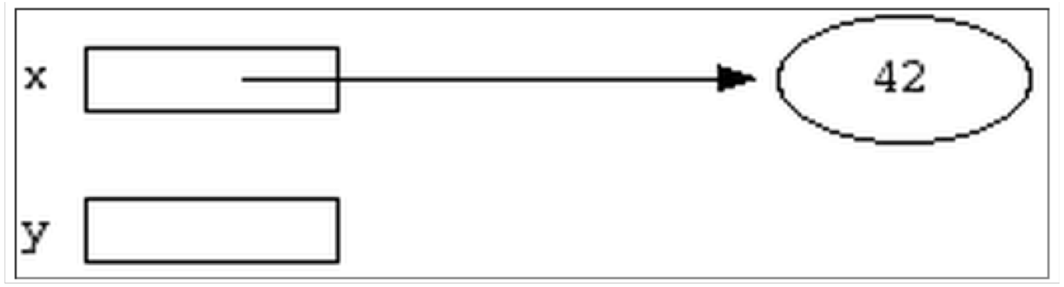
int *y;



int var;   x = &var;

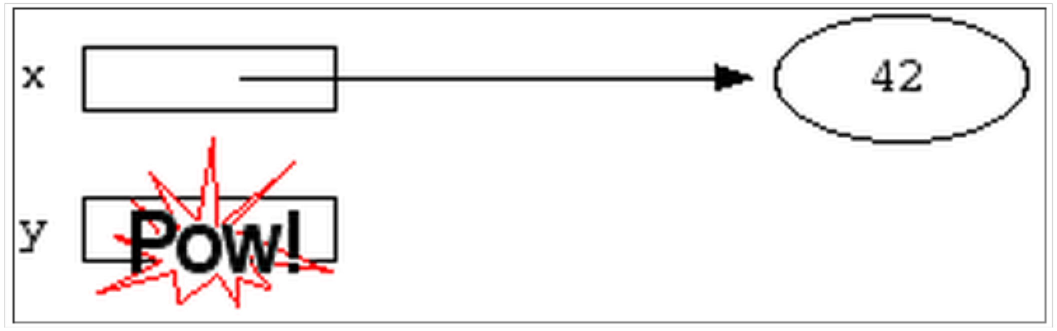


*x = 42;

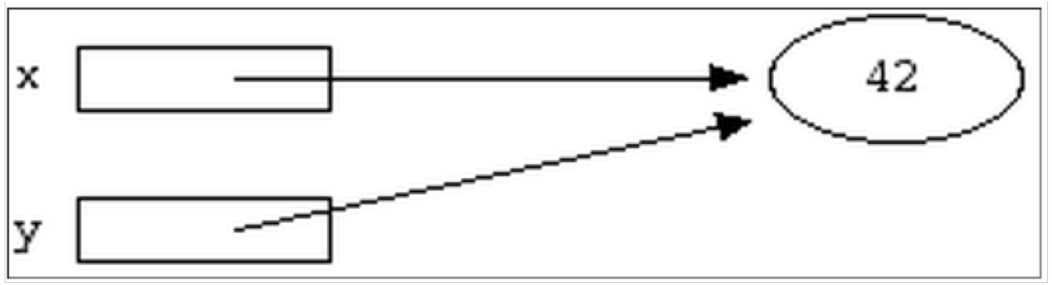


Pointer Example

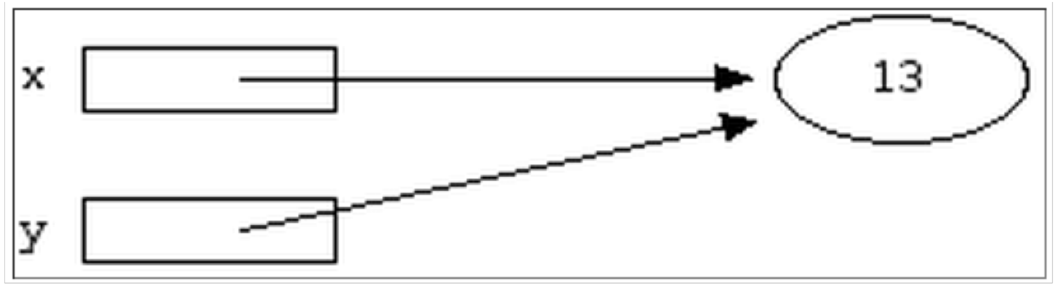
*y = 13;



y = x;

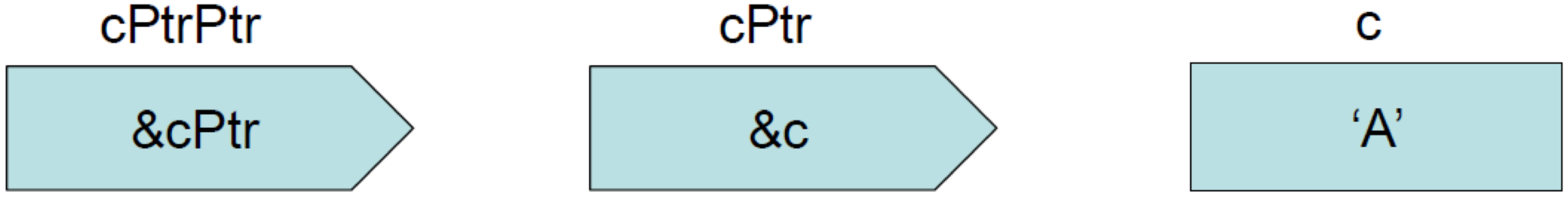


*x = 13; or
*y = 13;



Pointers to Pointers

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



Pointers to Functions

- Also known as: **function pointers**
- 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);

qsort Example

void qsort (void* base, size_t num, size_t size, int (*compar)(const void*,const void*));

Return value meaning for comparator function:

< 0	The element pointed by p1 goes before the element pointed by p2
= 0	The element pointed by p1 is equivalent to the element pointed by p2
> 0	The element pointed by p1 goes after the element pointed by p2

```
#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 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, …);
- 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[] = “John”;
    - fp = fopen(“file.txt”, “w+”)
    - fprintf(fp, **“%s has %d points.\n”, player, score**);

Parameter Passing

Pass by value

The data associated with the actual parameter is copied into a separate storage location assigned to the formal parameter.
Any modifications to the formal parameter variable inside the called function or method affect only this separate storage location and will therefore *not* be reflected in the actual parameter in the calling environment

```
int add(int a, int b) {
    return a+b;
}
void main() {
    int x=4,y=8;
    int z = add(x,y);
    printf(“%d”,x);
}
```

Parameter Passing...

- **Pass by reference**

  The formal parameter receives a pointer to the actual data in the calling environment. Any changes to the formal parameter *are* reflected in the actual parameter in the calling environment.

```
void swap(int *a, int *b) {
  int tmp = *a;
  *a = *b;
  *b = tmp;
}
void main() {
   int a = 1;
   int b = 2;
   printf("before swap a = %d\n", a);
   printf("before swap b = %d\n", b);
   swap(&a, &b);
   printf("after swap a = %d\n", a);
   printf("after swap b = %d\n", b); }
```

Task 1

- Create a function s.t. it takes three numbers ‘a’, ‘b’ and ‘c’ as arguments, computes a^b and store the results in ‘c’. It should not return any value. Call this function from main() and print the answer in main().

  Hint: pass by reference

  Hint: you may want to see the pow function [check the return type and library] (or compute the exponent yourself <- better)

Task 1 solution

```
#include <stdio.h>
#include <math.h> //library import
void exponent(int a, int b, double *c){
        *c=pow(a,b); //pow returns a pointer
}
int main(void) {
        int a=2;
        int b=2;
        double z;
        exponent(a,b,&z);
        printf("%f", z);
        return 0;
}
```

Task

Program Statement – Define a structure called student that will describe the following information.

- name (char *array)
- Uid (int)

Then create an array (of size 3) of this structure type.

struct student <array name>[3]; //access attributes using <array name>[index].attributename}

Using student, declare an array student with 3 elements and write a program to read the information about all the 3 students and print a sorted name wise list (sort by team name) containing names of students with their UIDs.

*you can hardcode the data for your convenience

Use the qsort function

Task solution

```
int compare (const void * a, const void * b ) {
   struct student *pa = (struct student*)a;
  struct student *pb = (struct student*)b;
    return strcmp(pa->name, pb->name);
}
qsort(<arrayname>,5, sizeof(struct student),compare);
```

*you can also typedef to avoid writing ‘struct’

Initializing array using malloc

```
int *arr = malloc (sizeof (int) * n); /* n is the length of the array */
int i;

for (i=0; i<n; i++)
{
  arr[i] = 0;
}
```

Task 2

/*Using structures to calculate the area of a rectangle*/

Create two structs for Rectangle and Point.

Calculate the area of the rectangle using the given coordinates (top left and bottom right)

Use the below structure:

```
typedef struct {
   Point topLeft;   /* top left point of rectangle */
   Point botRight;  /* bottom right point of rectangle */
} Rectangle;
```

Task 2 Solution

```
#include <stdio.h>
#include <string.h>
#include <math.h>

typedef struct {
   double x;
   double y;
} Point;

typedef struct {
   Point topLeft;   /* top left point of
rectangle */
   Point botRight;  /* bottom right
point of rectangle */
} Rectangle;
double computeArea(Rectangle *r);
```

```
int main()
{
   Point p;
   Rectangle r;
   printf("\nEnter top left point: ");
   scanf("%lf", &r.topLeft.x);
   scanf("%lf", &r.topLeft.y);
   printf("Enter bottom right point: ");
   scanf("%lf", &r.botRight.x);
   scanf("%lf", &r.botRight.y);
   printf("Top left x = %lf  y = %lf\n",
r.topLeft.x, r.topLeft.y);
   printf("Bottom right x = %lf  y = %lf\n",
r.botRight.x, r.botRight.y);
   printf("Area = %f", computeArea(&r));
   return 0;
}
```

```
double computeArea(Rectangle *r)
{
   double height, width, area;

   height = ((r->topLeft.y) - (r-
>botRight.y));
   width = ((r->topLeft.x) - (r-
>botRight.x));
   area = height*width;
   return (area);
}
```

Task 3

Write a C program using getchar() and putchar() which continuously takes user input and prints it on the screen. This should keep on happening till the user inputs a string containing ‘#’ and Enters.

Hint: use while(getchar() != #)

Task 3 solution

```
#include <stdio.h>
/* --  Copy input to output -- */
int main(void)
{
      int c;
      c = getchar();
      while ( c != "#" ) {
          putchar(c);
          c = getchar();
      }
      return 0;
}
```

Link to Code Discussed

- https://docs.google.com/document/d/1dbJ8SDTXfgtkIO9sbGprQmPULxqwQp2h4gwimWUYy94/edit?usp=sharing

Homework 4

- 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)**
  - Error checking (**fprintf**)
  - 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

- Start as soon as possible
- 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
- *memfrob() function for own test cases*