CS 35L

Software Construction Laboratory

Lecture 4.1

29th January, 2019

Logistics

- Hardware requirement for Week 8
  - Seeed Studio BeagleBone Green Wireless Development Board
- Presentations for Assignment 10
  - Fill your details min the link below
  - Do not fill a slot without a Presentation Topic
  - https://docs.google.com/spreadsheets/d/1o6r6CKCaB2du3klPflHiquymhBvbn7oP0wkHHMz_q1E/edit?usp=sharing
- Assignment 3 is due on Feb 2nd

Review - Previous Lab

- Python
- Build Process
  - Configure
  - Make
  - Make Install

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

Format Specifiers in C

- Defines the type of data to be printed on standard output –
- printf("%d", 4); //%d is integer

Data Types Revised

Data Type	Range	Bytes	Format Specifiers
char	-128 to 127	1	%c
unsigned char	0 to 255	1	%c
short signed int	-32,768 to 32,767	2	%d
short unsigned int	0 to 65,535	2	%u
signed int	-32768 to 32767	4	%d
unsigned int	0 to 65535	4	%u
long signed int	-2147483648 to 2147483647	4	%ld
long unsigned int	0 to 4294967295	4	%lu
float	-3.4e38 to 3.4e38	4	%f
double	-1.7e308 to 1.7e308	8	%lf
long double	-1.7e4932 to 1.7e4932	8	%Lf
Note: The sizes in this figure are for 32 bit compiler			

A simple C Program

```
#include <stdio.h>

int main() {
        printf("Hello World\n");
        return 0;
}
```

Loops

- int i; //initialize outside the loop unlike C++
- for(i=0;i<10;i++){
- }

- int i=0; //initialize outside the loop unlike C++
- while(i<10){
- i++; }

Functions

- Function Name
- Return Type
- Arguments/Parameters
- Function Body

```
int func(int a){
        //function body
return 0;
}
```

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);}
```

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 Student {
        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

Reading/Writing Characters

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

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

Points to Remember

- Normal variable stores the value whereas pointer variable stores the address of the variable
- The content of the C pointer always be a whole number i.e. address
- C pointer is always initialized to null, i.e. int *p = null
- The value of null pointer is 0
- & symbol is used to get the address of the variable
- * symbol is used to get the value of the variable that the pointer is pointing to
- If a pointer in C is assigned to NULL, it means it is pointing to nothing
- Two pointers can be subtracted to know how many elements are available between these two pointers
- But pointer addition, multiplication, division are not allowed
- The size of any pointer is 4 byte (for 32 bit compiler).

Dereferencing Pointers

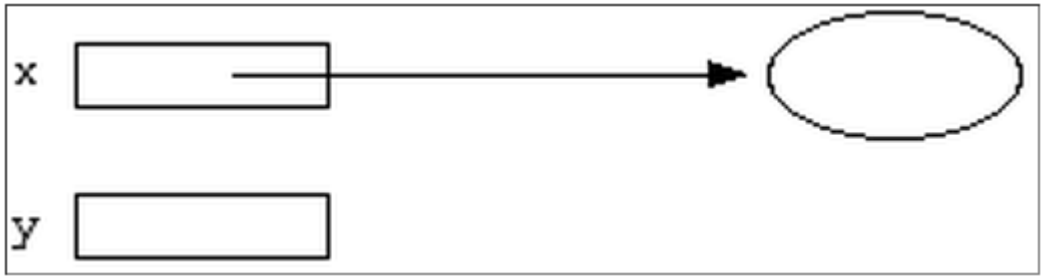
- Accessing the value that the pointer points to
- Example:
  - double x, *ptr;
  - ptr = &x; // let ptr point to x
  - *ptr = 7.8; // assign the value 7.8 to x

Pointer Example

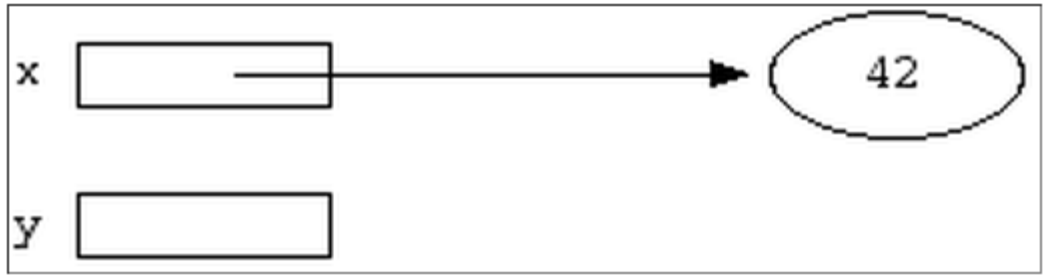
int *x;

int *y;

int var; x = &var;

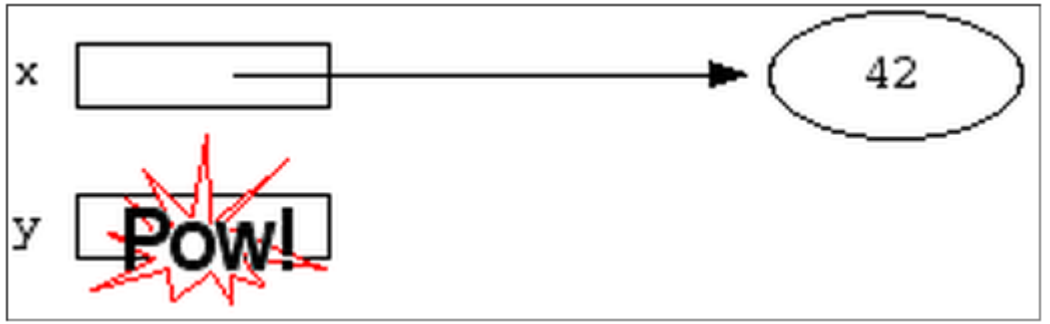


*x = 42;

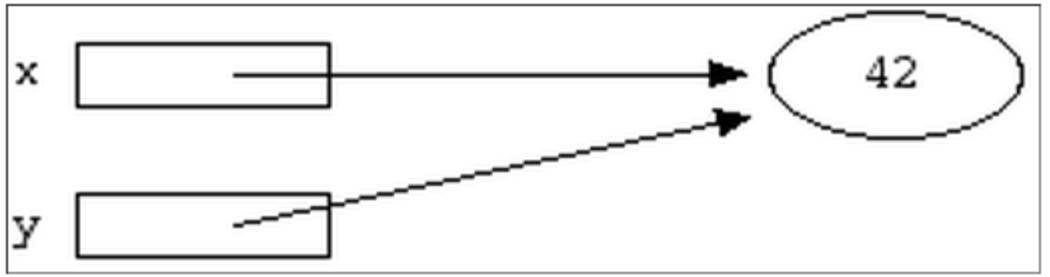


Pointer Example

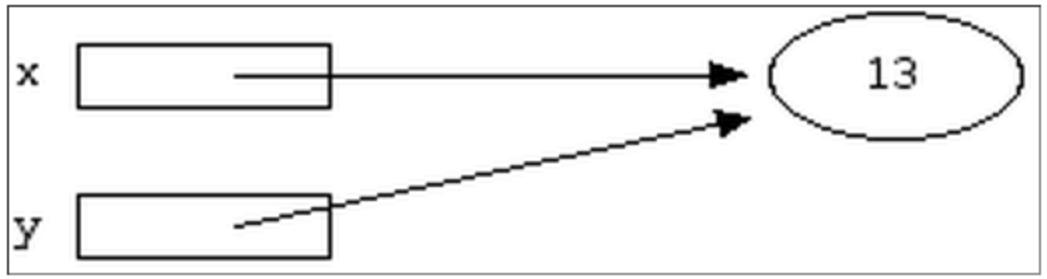
*y = 13;



y = x;

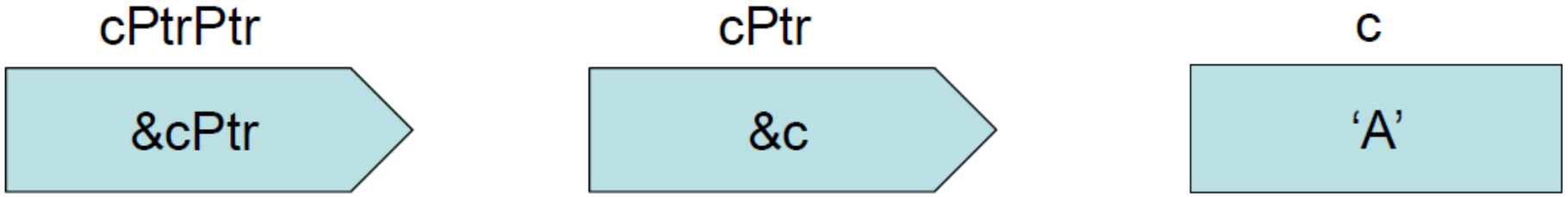


*x = 13; or
*y = 13;



Double Pointer

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



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 ab 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)

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

Assignment 3 - Homework

- randline.py script
  - Input: a file and a number n
  - Output: n random lines from file
  - Get familiar with language + understand what code does
  - Answer some questions about script (Q3, Q4)
- Implement shuf utility in python

Running randline.py

- Run it
  - ./randline.py –n 3 filename (need execute permission)
  - python randline.py –n 3 filename (no execute permission)
- randline.py has 3 command-line arguments:
  - filename: file to choose lines from
    - argument to script
  - n: specifies the number of lines to write
    - option
  - 3: number of lines
    - option argument to n
- Output: 3 random lines from the input file

Shuf.py

- Support the options for shuf
  - --echo (-e)
  - --head-count (-n)
  - --repeat (-r)
  - --help
- Support all type of arguments
  - File names and – for stdin
  - Any number of non-option arguments
- Error handling

Assignment 3 - Homework

- shuf.py – this should end up working almost exactly like the utility ‘shuf’
  - Check $ man shuf for extensive documentation
- Use randline.py as a starting point!
  - Modify to accomplish logical task of shuf
- shuf C source code :
  - Present in coreutils
  - This will give you an idea of the logic behind the operation that shuf executes
- Python argparse module instead of optparse:
  - How to add your own options to the parser
  - -e -n --repeat --echo etc

Assignment 3 - Homework

- shuf.py – this should end up working almost exactly like the utility ‘shuf’
  - Check $ man shuf for extensive documentation
- Use randline.py as a starting point!
  - Modify to accomplish logical task of shuf
- shuf C source code :
  - Present in coreutils
  - This will give you an idea of the logic behind the operation that shuf executes
- Python argparse module instead of optparse:
  - How to add your own options to the parser
  - -e -n --repeat --echo etc

Assignment 3 - Homework Hints

- If you are unsure of how something should be output, run a test using existing shuf utility!
  - Create your own test inputs
- The shuf option --repeat is Boolean
  - Which action should you use?
- Q4: Python 3 vs. Python 2
  - Look up “automatic tuple unpacking”
- Python 3 is installed in /usr/local/cs/bin
  - export PATH=/usr/local/cs/bin:$PATH

Questions?