C Tutorial for EDA093 (part 1)

Amir Keramatian

About the Crash Course

Introduction to C concepts required for EDA093, especially Lab1

Recommended reading material:

- •*The C Programming Language*, Kernighan, Richie
- •*C Traps and Pitfalls*, Andrew Koenig, Addison-Wesley
- •Kernighan & Ritchie: "The C programming language", 2nd edition. Prentice-Hall, 1988.

Outline

Variables & data types

Loops

Pointers

Functions

Recursive functions

Variables & Data types

Variable *declaration* tells the compiler two things: (I) the name of the variable. (II) The type of the variable.

- char ch; //compiler sets aside 1 byte for ch
- ch = 'a'// fills the space corresponding to ch with the binary representation of 'a'
- int x = 5; //compiler sets aside 4 bytes for x and fills it with the binary representation of 5=0b101
- Other data types in C are: float, double, long, void, …

Changing Variable Values

```
int x = 5;
double y = 0.5;
double z = y+x; //implicit conversion of x
x+=5; // x=x+5;
x--;
--y;
```

Formatted Output

The printf(“”, …) function lets you print formatted data to output:

- printf(“Hello world\n”);
- printf(“x is %d”, x);
- printf(“x+y is %f”, x+y);

Format specifiers:

- %d: integer
- %f: float
- %c: character
- %s: string

Formatted Input

The scanf("", …) function lets you receive formatted data from standard input:

- int x;
- float y;
- printf("enter an integer and a float\n");
- Scanf("%d %f\n", &x, &y);

Mind the ampersand(&): it is needed to instruct the compiler to store the input value into the respective variables.

Looping in C

“A loop is a sequence of instructions continually repeated until a certain condition is reached”

A few structures in C that allow looping:

- For loop
- While loop
- Do-While loop

Looping in C: For loop

```
for (<initialization>; <test>; <update>) {
// code block
}
```

initialization takes place before the loop starts

before each iteration *test* is evaluated: if true iterate; otherwise, stop

after each iteration, *update* is executed

Example: loop

Code:

A simple C program that:

- gets integer N from the user
- computes the sum: 1+2+...+N

Compilation:

- Using gcc: gcc filename.cc -o output.out

Pointers

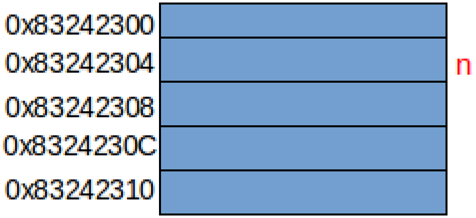
In C programming, pointers are data types that contain memory addresses of other variables.

int n = 2;

int* nPtr = &n; // ==> <nPtr = 0x83242304>

***nPtr // de-referencing the pointer**

printf(“%d”, *nPtr) ?



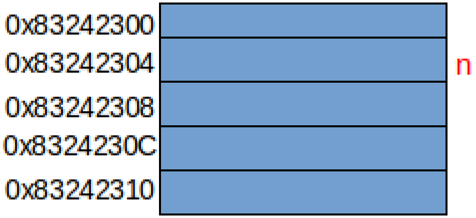
Pointers

```
int n = 2;

int* nPtr =  &n; // ==> <nPtr = 0x83242304>

printf(“%d”, *nPtr) // prints 2
```

How to get the address of the memory location in which n is stored?



0x83242300
0x83242304
0x83242308
0x8324230C
0x83242310

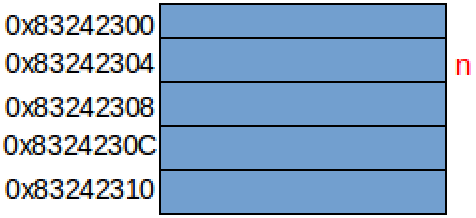
Pointers

int n = 2; int* nPtr = &n; // ==> <nPtr = 0x83242304>

printf(“%d”, *nPtr) // prints 2

Q: How to get the address of the memory location in which n is stored?

A: printf(“The address of the memory location in which n is stored is: %p”, nPtr);



Example: Pointers

- Performing operations on a variable through a pointer to the variable
- Performing operations on a variable through a pointer to the pointer to the variable!

Functions

A function has a signature:

```
<return type> name_of_the_function(type input_1, ….,
type input_N){

//code block

}
```

***Declaration* and *Definition* of a function can come together or separately**

Example: Functions

Two Functions:

- A function to test whether a given integer is even
- A function to compute the squares of its variables

Passing Arguments:

- Pass by value
- Pass by pointer

Recursive Functions

An onion is a natural recursive phenomenon!

- Peel one layer off an onion => what remains is an onion

Recursion in C programming:

- When a function calls itself
- When a bigger problem can be broken down into smaller chunks of same problem

Example:Recursive Functions

N! = Factorial(N)

= 1 * 2 * … * (N-1) * N

Q: Recursive definition of the factorial function?

Example:Recursive Functions

N! = Factorial(N)

= 1 * 2 * … * (N-1) * N

Q: Recursive definition of the factorial function?

A: N! = N * (N-1)!

1! = 1 (no recursion, base case)

Example:Recursive Functions

The Fibonacci sequence:

1, 1, 2, 3, 5, 8, 13, 21, …

Recursive Definition:

- Fib(N) = Fib(N-1) + Fib(N-2)
- Fib(1) = Fib(2) = 1 (base case)

Next Session

Arrays

Strings

Dynamic Memory

Linked-lists