FIU Engineering & Computing
School of Computing & Information Sciences

COP 4338 - Programming III

Lecture 1 - Introduction to C

Miguel Alonso Jr, Ph.D.

School of Computing and Information Sciences, FIU

Table of contents

Why C programming?

Why C programming?

The Good

- fast (compiled language -> close to hardware)
- portable (compile and run on most hardware)
- the language is small (unlike Java or C++)
- mature (really old; lot's of resources)
- many tools avaialable (like CLion)
- direct access to memory
- access to low-level system features

The Bad

- the language is small (but there are many APIs)
- it's easy to get into trouble, e.g. with direct memory access & pointers
- the code — compile — test (crash) — debug cycle
- you must manage memory yourself
- sometimes code is more verbose than in high-level scripting languages like Python, R, etc

Similarities with Java

C vs Java

Java	C
object-oriented	function-oriented
strongly-typed	can be overridden
polymorphism (+, ==)	very limited (int/float)
classes for name space	single name space, file oriented
macros are external, rarely used	macros common (preprocessor)
layered I/O model	byte-stream I/O

C vs Java

Java	C
automatic mem. management	function calls
no pointers	pointers (mem. addr.) common
by-reference, by-value	by-value parameters
exceptions, exception handling	if (f() < 0) error OS signals
macros are external, rarely used	macros common (preprocessor)
concurrency (threads)	library functions

C vs Java

Java	C
length of array	on your own
string as type	just bytes (char []), with 0 end
lot's of common libs	OS-defined

About the Course

About the Course

Instructor
Miguel Alonso Jr.

Schedule
Tu/Th 3:45PM - 4:50PM, ECS 254A

Office Hours
Tu/Th 2:00PM - 3:30PM, ECS 254A

Phone
305-348-4848

Email
malonsoj@cs.fiu.edu

Check the syllabus for Grading, Topics, and Schedule.

Setting up the environment

Environment Setup

- We'll be using a *NIX environment. (Sorry windows users)
- Virtual Machine: Docker
- Need to intall git and docker first

Then clone:

`https://github.com/drmaj/C-Docker-Setup`

and follow the instructions.

git

git is a distributed version control tool. We'll be using it this term to clone repositories and submit code.

`https://git-scm.com/book/en/v2/Getting-Started-Installing-Git`

Docker

Docker is a computer program that performs operating-system-level virtualization, also known as ”containerization”. It provides a way to run applications securely isolated in a container, packaged with all its dependencies and libraries.

`https://docs.docker.com/install/`

My first C program: Adding two numbers

Adding two numbers

```
x = 5;
y = 9;
```

Adding two numbers

```
x = 5;
y = 9;
z = x + y;
```

Adding two numbers

```
int x, y, z;
x = 5;
y = 9;
z = x + y;
```

Adding two numbers

```
int x, y, z;
x = 5;
y = 9;
z = x + y;
printf("The sum of %d and %d is %d",x, y, z);
```

Adding two numbers

```
void main(){
    int x, y, z;
    x = 5;
    y = 9;
    z = x + y;
    printf("The sum of %d and %d is %d",x, y, z);
}
```

Adding two numbers

```c
#include<stdio.h>

int main(){
    int x, y, z;
    x = 5;
    y = 9;
    z = x + y;
    printf("The sum of %d and %d is %d",x, y, z);
    return 0;
}
```

Manual Compilation

Compiling on the command line

- Must be in the main folder
- To compile

```
gcc -o add add.c
```

- To run

```
./add
```

Basic C

C statements

- Examples
  - `x = y + 3; /*Assignment*/`
  - `printf("hello, world!"); /*Function call*/`
  - `int x; /*Variable Declaration*/`
- Each statement ends with a semicolon

Variables

- variables store values, declared before usage
- Example
  - `a = 4 + 3; /*a is a variable*/`
- Types
  - `int a; /*Integers*/`
  - `char a; /*Single characters*/`
  - `float a; /*Decimals*/`
  - `double a; /*Double precision decimals*/`

int

- 4 byts (compiler dependent)
  - total of 2^{32} values
  - range: -2^{31} to 2^{31} - 1
- types of integers
  - `short int a; /* 2 bytes */`
  - `long int a; /* 8 bytes */`
  - `unsigned int a; /* only positive numbers */`
- What is the range of an unsigned integer?

char

- Example
  - `var = 'x';`
- 1 byte
  - total of 2^8 values
- ASCII representation

Dec	Hex	Name	Char	Ctrl-char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	0	Null	NUL	CTRL-@	32	20	Space	64	40	@	96	60	`
1	1	Start of heading	SOH	CTRL-A	33	21	!	65	41	A	97	61	a
2	2	Start of text	STX	CTRL-B	34	22	"	66	42	B	98	62	b
3	3	End of text	ETX	CTRL-C	35	23	#	67	43	C	99	63	c
4	4	End of xmit	EOT	CTRL-D	36	24	$	68	44	D	100	64	d
5	5	Enquiry	ENQ	CTRL-E	37	25	%	69	45	E	101	65	e
6	6	Acknowledge	ACK	CTRL-F	38	26	&	70	46	F	102	66	f
7	7	Bell	BEL	CTRL-G	39	27	'	71	47	G	103	67	g
8	8	Backspace	BS	CTRL-H	40	28	(	72	48	H	104	68	h
9	9	Horizontal tab	HT	CTRL-I	41	29	)	73	49	I	105	69	i
10	0A	Line feed	LF	CTRL-J	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	VT	CTRL-K	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	FF	CTRL-L	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage feed	CR	CTRL-M	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	SO	CTRL-N	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	SI	CTRL-O	47	2F	/	79	4F	O	111	6F	o
16	10	Data line escape	DLE	CTRL-P	48	30	0	80	50	P	112	70	p
17	11	Device control 1	DC1	CTRL-Q	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	DC2	CTRL-R	50	32	2	82	52	R	114	72	r
19	13	Device control 3	DC3	CTRL-S	51	33	3	83	53	S	115	73	s
20	14	Device control 4	DC4	CTRL-T	52	34	4	84	54	T	116	74	t
21	15	Neg acknowledge	NAK	CTRL-U	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	SYN	CTRL-V	54	36	6	86	56	V	118	76	v
23	17	End of xmit block	ETB	CTRL-W	55	37	7	87	57	W	119	77	w
24	18	Cancel	CAN	CTRL-X	56	38	8	88	58	X	120	78	x
25	19	End of medium	EM	CTRL-Y	57	39	9	89	59	Y	121	79	y
26	1A	Substitute	SUB	CTRL-Z	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	ESC	CTRL-[	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	FS	CTRL-\	60	3C	<	92	5C	\	124	7C	|
29	1D	Group separator	GS	CTRL-]	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	RS	CTRL-^	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	US	CTRL-_	63	3F	?	95	5F	_	127	7F	DEL

float / double

- Floating point decimal
- Example
  - `float a;`
  - `a = 2.54;`
- 4 bytes
  - IEEE format
  - $-3.4e^{38}$ to $3.4e^{38}$
- double
  - twice as much memory as float
  - twice the precision
  - usually 8 bytes

sizeof

```c
#include<stdio.h> /* Header file */

int main(){ /* The main entrypoint function */
    int x;
    x = 0;
    printf("x is %d bytes", sizeof(x));
    return 0;
}
```

Casting

- Can change the variable type after declaration

```
int x;
float y;
x = 3;
y = (float) x; /* Explicit casting */
y = x; /* Implicit casting */
```

printf

- Example

```
printf("Hello!\n");
printf("The sum of %d and %d is %d", x, y, z);
```

- Output

```
Hello!
The sum of 5 and 10 is 15
```

- Placeholders
  - %d int
  - %f float
  - %c char

scanf

Practice: Compile and run the following program:

```c
#include<stdio.h> /*Header file*/

int main() /* The main entrypoint function */
{
    int x, y, z; /*Variable Declaration*/
    printf("Enter x:");
    scanf("%d", &x); /* Wait for input */
    printf("Enter y:");
    scanf ("%d", &y); /* Wait for input */
    z = x + y;
    printf ("The sum is %d", z);
    return 0;
}
```

CMake and Make

CMakeLists.txt

```
cmake_minimum_required(VERSION 3.10)
project (add)
add_executable(add add.cpp)
```

Then run..

```
cmake .
make
```

TODO

- Read Chapter 1 of the textbook
- Setup your development environment
- Run the scanf program
- Submit the c file, along with the CMakeLists.txt file in a zip file on Canvas (Assignment 0).