

C/C++ PROGRAMMING REVIEW

CPSC 441 - Tutorial I

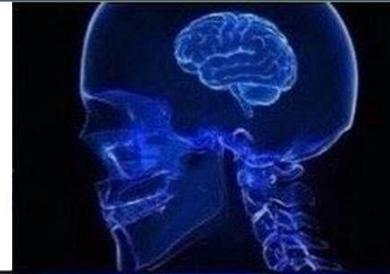
Reza Gholizadeh

Winter 2018

WHY C?

- **Speed and Control**
 - **No garbage collection**
 - **No dynamic types**

Python



Java



C / C++



Machine Code



} Your
TA!

HELLO WORLD!

C

```
1. #include <stdio.h>
2. int main(int argc, char *argv[]){
3.     printf("Hello World!\n");
4.     return 0;
5. }
```

C++

```
1. #include <iostream>
2. using namespace std;
3. int main(int argc, char *argv[]){
4.     cout <<"Hello World!\n";
5.     return 0;
6. }
```

COMPILING

C

```
$ gcc Hello.c  
$ a.out  
Hello World!
```

```
$ gcc Hello.c -o Hello  
$ ./Hello  
Hello World!
```

C++

```
$ g++ Hello.cpp -o Hello  
$ ./Hello  
Hello World!
```

COMPILER **OPTIONS**

Some useful command line options:

- **[-o file]**: specifies the output file for object or executable
- **[-Wall]**: show all warnings (highly recommended)
- **[-l libnam]**: Links the library libname
e.g., -lsocket
- If you get errors saying the library cannot be found, make sure the path is correctly set, and you do have the libraries you need.

IDE OR NO IDE?

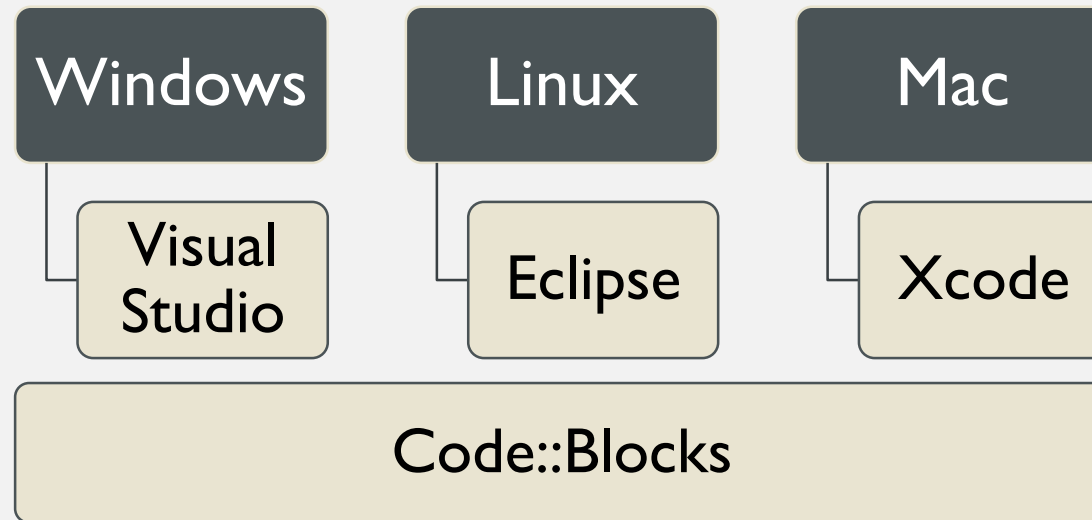


IDE for writing,
compiling, running



IDE for writing
gcc for compiling and
running

IDE RECOMMENDATION



C - BASICS

Data Types

Main Function

Arrays

Structures

Pointers

Strings

BASIC DATA TYPES

Name	Description	Size (Bytes)	Range
char	Character or small integer.	Typically 1	signed: -128 to 127 unsigned: 0 to 255
short int (short)	Short Integer.	≥ 2	signed: -32768 to 32767 unsigned: 0 to 65535
int	Integer. Most efficient.	≥ 2 ; Typically 4	signed: -2147483648 to 2147483647 unsigned: 0 to 4294967295
long int (long)	Long integer.	≥ 4	signed: -2147483648 to 2147483647 unsigned: 0 to 4294967295
long long int	long long integer.	≥ 8	signed: -9.2e18 to 9.2e18 unsigned: -1.8e19 to 1.8e19
float	Floating point number.	4	+/- 3.4e +/- 38 (~7 digits)
double	Double precision floating point number.	8	+/- 1.8e +/- 308 (~15 digits)
long double	Long double precision floating point number.	≥ 8 ; Typically 16	+/- 1.2e +/- 4932 (~34 digits)

MAIN FUNCTION

- `main(int argc, char *argv[])`
 - The main function of the program
 - The program starts at the beginning of the `main()` and ends at the end of it.

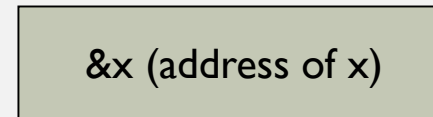
```
1. int main(int argc, char *argv[]) {  
2.     /* your code */  
3.     return 0;  
4. }
```

- `argc` - number of arguments passed to the program
- `argv` – array of strings showing command line arguments
 - Name of executable is stored in `argv[0]`
- The return value is `int`
 - **0** for **success**
 - **>0** for **some error**

POINTERS

- A **pointer** is just an address to some memory location
 - Another variable
 - Some dynamically allocated memory
 - Some function
 - NULL

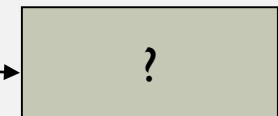
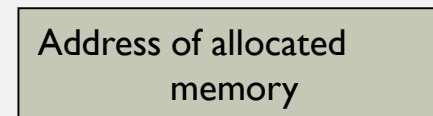
```
int *p = &x;
```



```
int x = 4;
```



```
int *p = malloc (sizeof int);
```



allocated
memory

ARRAYS

- **type** name[size];
 - **type**: type of data in the array
 - **name**: a pointer to the first element (and the name!)
 - **size**: size of the array

Watch out: C does not care about array bounds!

```
1. int numbers[];  
2. char characters[6];  
3. numbers[-1] = 5;  
4. characters[6] = 'a';
```

- **numbers** – an array of integers without defining its length
- **characters** – an array of chars with length of 6

No “index out of bound”
error while compiling and
running line 3 and 4.

You program will crash at
some point!

STRUCTURES

- `struct` structure_tag{
 member_def;
 ...
 member_def;
}
- User defined data type logically grouping different data types
- Similar to C++/Java classes without methods and encapsulation

```
1. struct Books{  
2.     char title[50];  
3.     char author[50];  
4.     char subject[50];  
5.     int book_id;  
6. }
```

STRINGS

- `char name[length];`
 - A string is an array of chars terminated with “\0” – “**hello**” is actually **hello\0**

```
1. char str1[10];
2. char str2[10] = {"hello"};
3. char *strp1;
4. char *strp2 = malloc(sizeof(char)*10);
```

- `str1` – a string of 10 characters
- `str2` – an initialized string
- `strp1` – a char pointer
- `strp2` – a char pointer initialized to point to a chunk of memory

STRING LIBRARY

- `string.h` library contains some useful functions:
 - `char *strcpy(char *dest, char *source, int num) :`
copies chars from *source* to *dest* and stopped after *num* element
 - `int strlen(const char *source) :`
returns number of chars, excluding NULL
 - `char *strchr(const char *source, const char ch) :`
returns pointer to first occurrence of *ch* in *source*; NULL if none
 - `char *strstr(const char *source, const char *search) :`
return pointer to first occurrence of *search* in *source*

FORMATTING STRINGS

- `int sscanf(char *str, char *format, var1, var2, ...)`
 - Parses *str* based on the *format*. if desired, you can store the parsed strings into *var1*,... *varn* based on the *format*
 - Returns the number of successful conversions
- `int sprintf(char *buffer, char *format, var1, var2, ...)`
 - Produces a string formatted according to *format* directives and place this string into the *buffer*
 - Returns the number of successful conversions

FORMATTING CODES FOR SCANF

Code	Meaning	Variable
%c	Matches a single character	char
%d	Matches an integer in decimal	int
%f	Matches a real number	float
%s	Matches a string up to a white space	char *
%[^c]	Matches a string up to next c char	char *

For many more formatting options, refer to:
<http://www.cplusplus.com/reference/cstdio/scanf/>

FORMATTING CODES FOR PRINTF

Values normally right-justified; use negative field width to get left-justified

Code	Meaning	Variable
%nc	Char in field of n spaces	char
%nd	Integer in field of n spaces	int
%n.mf	Real number in width n.m decimals	float, double
%n.ms	First m chars from string in width n	char *
%%	Writes a single % to the stream	

For many more formatting options, refer to:

<http://www.cplusplus.com/reference/cstdio/printf/>

STANDARD LIBRARY

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

- Formatted I/O:

```
-int scanf(const char *format, ...)
```

- read from standard input and store according to *format*

```
-int printf(const char *format, ...)
```

- write to standard output according to *format*

- File I/O

```
-FILE *fopen(const char *path, const char *mode)
```

- open a file and return the file descriptor

```
-int fclose(FILE *stream)
```

- close the file; return 0 if successful, EOF if not

STANDARD LIBRARY

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

- Other I/O operations:

```
-int getchar()
```

- read the next character from stdin; returns EOF if none

```
-char *fgets(char *buffer, int size, FILE *in)
```

- read the next line from a file into *buffer*

```
-int fputs(const char *str, FILE *out)
```

- output the string to a file, stopping at ‘\0’
- returns number of characters written or EOF

REFERENCES

- *C* tutorial:
 - <https://www.tutorialspoint.com/cprogramming/>
- *C++* tutorial:
 - <https://www.tutorialspoint.com/cplusplus/>
- *C* for Java programmers:
 - http://faculty.ksu.edu.sa/jebari_chaker/papers/C_for_Java_Programmers.pdf
 - <http://www.cs.columbia.edu/~hgs/teaching/ap/slides/CforJavaProgrammers.ppt>
- ***Many other useful resources on the internet***