Introduction to C/C++

Carlos E. Alvarez[1].

[1]Dep. de Matemáticas aplicadas y Ciencias de la Computación, Universidad del Rosario

Bogotá, June 2018

Universidad del Rosario | MACC Matemáticas Aplicadas y Ciencias de la Computación

CONTENTS

A FIRST CODE

File: hello_world.cpp

```
#include <stdio.h>
using namespace std;

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

Compilation:

```
$ g++ -o hello_world hello_world.cpp
```

DATA TYPES

Primitive data types:

Integer:	`int`	(typically 32 bits)
	`long`	(typically 64 bits)
Floating point:	`float`	(typically 32 bits)
	`double`	(typically 64 bits)
Boolean:	`bool`	(Implementation defined)
Character:	`char`	(8 bits)

Function `sizeof()` shows the size of a type.

OPERATORS

Assignment operator

=

OPERATORS

Assignment operator

```
=
```

Arithmetic

```
+, -, *, /,  %
+=, -=, *=, /=, ++, --
```

OPERATORS

Assignment operator

```
=
```

Arithmetic

```
+, -, *, /,  %
+=, -=, *=, /=, ++, --
```

Boolean (relation)

```
==, !=, >, >, >=, <=
```

OPERATORS

Assignment operator

```
=
```

Arithmetic

```
+, -, *, /,  %
+=, -=, *=, /=, ++, --
```

Boolean (relation)

```
==, !=, >, >, >=, <=
```

Boolean (logical)

```
!, &&, ||
```

VARIABLES

Declaration

qualifier type name = expression

VARIABLES

Declaration

qualifier type name = expression

Examples

```
char myCharacter = 'A';
const double PI = 3.14159265358979323846;
```

VARIABLES

Declaration

qualifier type name = expression

Examples

```
char myCharacter = 'A';
const double PI = 3.14159265358979323846;
```

- Local variable: Declared within a function definition
- Global variable: Declared outside any function definition

I/O

Write to the screen:

C++ style (iostream)

```
cout << expression;
```

Example:

```
int a = 1000;
cout << "A " << a << " thanks!\n";
```

I/O

Write to the screen:

C++ style (iostream)

```
cout << expression;
```

Example:

```
int a = 1000;
cout << "A " << a << " thanks!\n";
```

C style (stdio.h)

```
printf(format, expression);
```

Example:

```
int a = 1000;
printf("A %d thanks!\n", a);
```

I/O

Read from keyboard:

C++ style (iostream)

```
cin >> expression;
```

Example:

```
int a;
cin >> a;
```

I/O

Read from keyboard:

C++ style (iostream)

```
cin >> expression;
```

Example:

```
int a;
cin >> a;
```

C style (stdio.h)

```
scanf(format, expression);
```

Example:

```
int a;
scanf("%d", &a);
```

CONTROL STATEMENTS

if/else

```
if(condition 1){
   code 1;
}else if(condition 2){
   code 2;
}else{
   code 3;
}
```

CONTROL STATEMENTS

switch

```
switch(expression){
  case constant-expression 1:
    code 1;
    break;
    .
    .
  case constant-expression n:
    code n;
    break;
  dafault:
    code by default;
    break;
}
```

CONTROL STATEMENTS

while

```
while(condition){
   code;
}
```

CONTROL STATEMENTS

while

```
while(condition){
   code;
}
```

for

```
for(initial condition;end condition;increment){
   code;
}
```

FUNCTIONS

Function prototype (.hpp)

```
type name(arguments);
```

Function implementation (.cpp)

```
type name(arguments){
   code;
      .
      .
   return expression;
}
```

File I/O

Open a file for writing (stdio.h)

```
FILE *file;
file = fopen("file_name", "w");
fclose(file);
```

File I/O

Open a file for writing (stdio.h)

```
FILE *file;
file = fopen("file_name", "w");
fclose(file);
```

Example:

```
FILE *myFile;
myFile = fopen("my_file.txt", "w");
for(int i = 0; i < N; i++){
  fprintf(myFile, "%d %d %d\n", i, i+1, i+2);
}
fclose(myFile);
```

File I/O

Open a file for reading (stdio.h)

```
FILE *file;
file = fopen("file_name", "r");
fclose(file);
```

File I/O

Open a file for reading (stdio.h)

```
FILE *file;
file = fopen("file_name", "r");
fclose(file);
```

Example:

```
FILE *myFile;
myFile = fopen("my_file.txt", "r");
int num;
while(true){
  fscanf(myFile, "%d", &num);
  if(feof(myFile)) break;
  printf("%d ", num);
}
fclose(myFile);
```

CONTENTS

LIBRARIES

Import a library

```
System header C:      #include <library.h>
System header C++:    #include <library>
User defined header:  #include "library.hpp"
```

LIBRARIES

Import a library

```
System header C:      #include <library.h>
System header C++:    #include <library>
User defined header:  #include "library.hpp"
```

<.>: Searches the shared library on a standard list of system directories.

LIBRARIES

Import a library

```
System header C:      #include <library.h>
System header C++:    #include <library>
User defined header:  #include "library.hpp"
```

<.>: Searches the shared library on a standard list of system directories.

".": Searches the static library in the same directory first, then on the standard directories.

LIBRARIES

Import a library

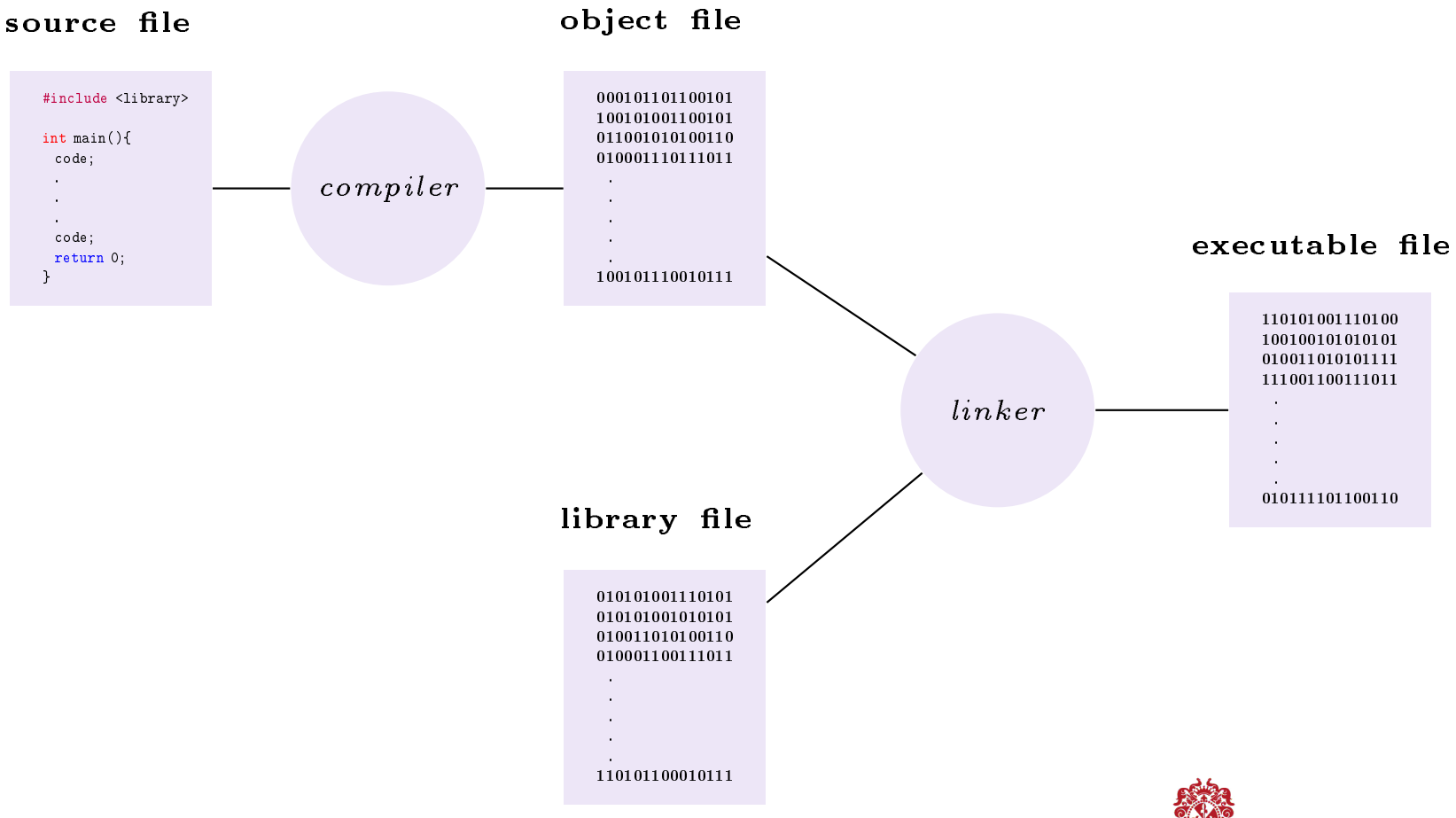
```
System header C:      #include <library.h>
System header C++:    #include <library>
User defined header:  #include "library.hpp"
```

<.>: Searches the shared library on a standard list of system directories.

".": Searches the static library in the same directory first, then on the standard directories.

List standard directories: `cpp -v /dev/null`

COMPILATION



COMPILATION

GNU C++ compiler: g++

Executable file

`g++ -o` *executable* *main_src.cpp* → *executable*

Object file

`g++ -c` *source.cpp* → *source.o*

COMPILATION

Compiling:

Object file

`g++ -c` *source1*`.cpp` → *source1*`.o`

`g++ -c` *source2*`.cpp` → *source2*`.o`

COMPILATION

Compiling:

Object file

`g++ -c` *source1*`.cpp` → *source1*`.o`

`g++ -c` *source2*`.cpp` → *source2*`.o`

Linking:

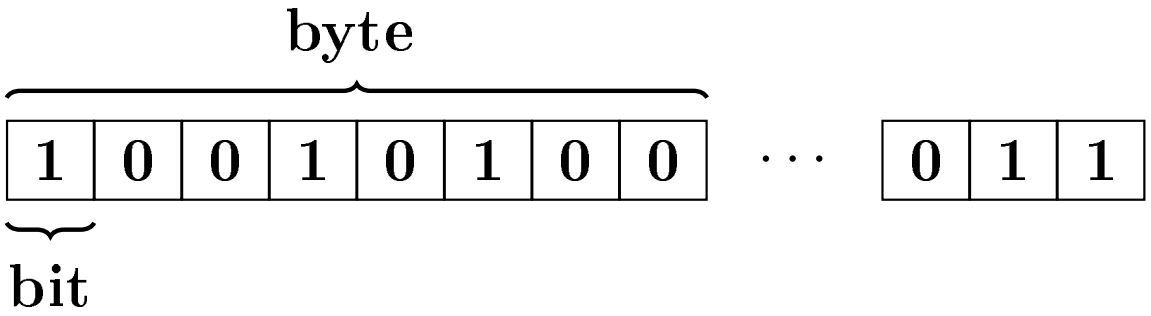
Executable file

`g++ -o` *executable* *main_src*`.cpp` *source1*`.o` *source2*`.o`

→ *executable*

CONTENTS

BITS AND BYTES



- Bit: Holds a 0 or a 1
- Byte: 8 bits
- Word: Holds an int . Usually 4 or 8 bytes

BINARY AND HEX BASES

Binary

$$10010 = 1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = 18$$

BINARY AND HEX BASES

Binary

$$10010 = 1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = 18$$

Hexadecimal

"Digits": `0,1,2,3,4,5,6,7,8,9,a,b,c,d,e` *and* `f`

$$\texttt{1c3a} = 1 \times 16^3 + 12 \times 16^2 + 3 \times 16^1 + 10 \times 16^0 = 7226$$

BINARY AND HEX BASES

Binary

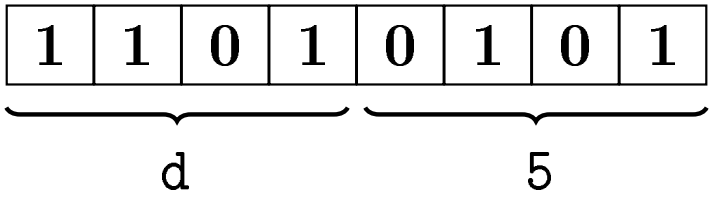
$10010 = 1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = 18$

Hexadecimal

"Digits": `0,1,2,3,4,5,6,7,8,9,a,b,c,d,e` *and* `f`

`1c3a` $= 1 \times 16^3 + 12 \times 16^2 + 3 \times 16^1 + 10 \times 16^0 = 7226$

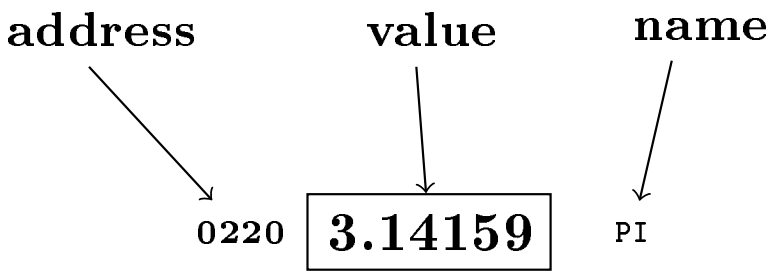
Binary to Hex:



Memory addresses are represented in Hex!

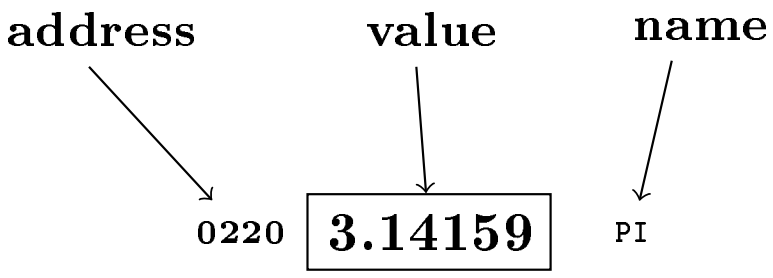
MEMORY ASSIGNMENT

```
const double PI = 3.14159;
```



MEMORY ASSIGNMENT

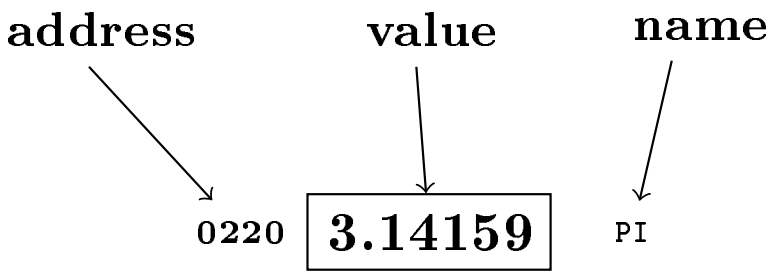
```
const double PI = 3.14159;
```



Stack: Closely managed by the processor. Limited size. Fast access. Stores local variables and releases them when out of scope

MEMORY ASSIGNMENT

```
const double PI = 3.14159;
```



Stack: Closely managed by the processor. Limited size. Fast access. Stores local variables and releases them when out of scope

Heap: Managed by the user. Size limited by the physical memory. Global in scope

POINTERS

xkcd.com/138/

POINTERS

xkcd.com/138/

- Store variable addresses

POINTERS

xkcd.com/138/

- Store variable addresses
- Refer to large structures in a compact way

POINTERS

xkcd.com/138/

- Store variable addresses
- Refer to large structures in a compact way
- Used when reserving memory during execution

POINTERS

xkcd.com/138/

- Store variable addresses
- Refer to large structures in a compact way
- Used when reserving memory during execution
- Store relationships among data

POINTERS

Declaration

qualifier type ***pointer_name* = &*name***

POINTERS

Declaration

*qualifier type *pointer_name = &name*

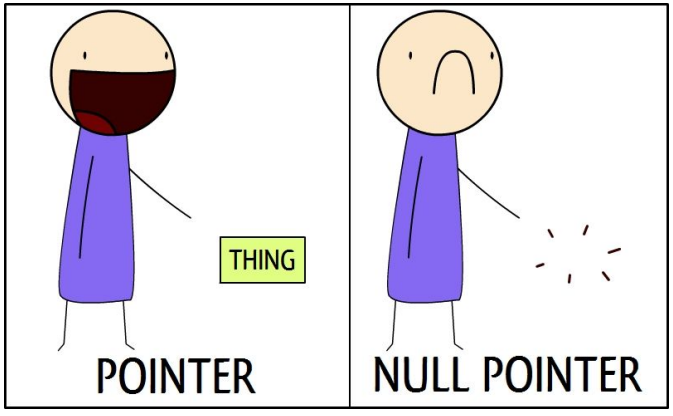
Example

```
int myVariable = 5;
int *myPointer;
myPointer = &myVariable;
```

*: Value pointed to.

&: Address of.

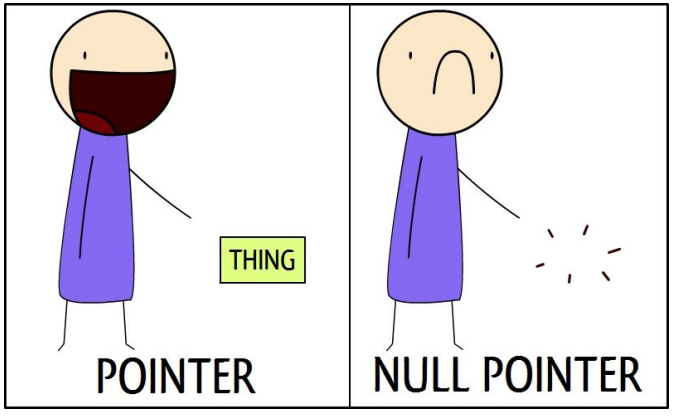
POINTERS



somethingofthatilk.com

`NULL` pointer:

POINTERS

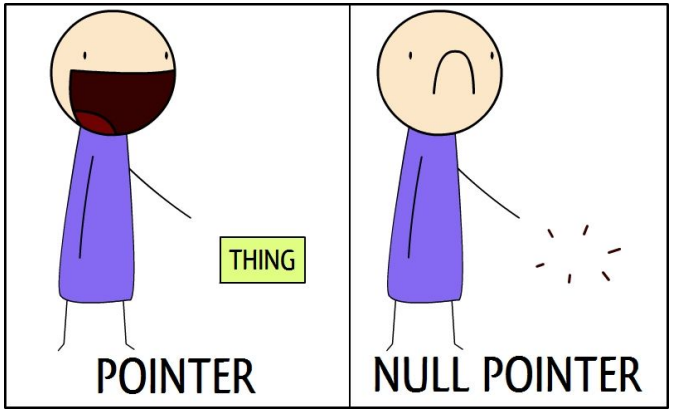


somethingofthatilk.com

`NULL` pointer:

- Indicates the pointer does not refer to a valid memory address

POINTERS



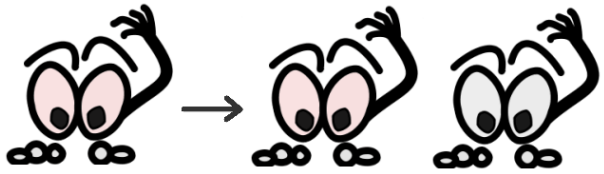
somethingofthatilk.com

`NULL` pointer:

- Indicates the pointer does not refer to a valid memory address
- Illegal to use * on a null pointer

PASS BY REFERENCE OR BY VALUE

Pass by reference



Pass by value

Image: https://www.elsieisy.com/where-do-you-live/

PASS BY REFERENCE OR BY VALUE

Pass by reference

```
void foo(int &var){
  var++;
}
```

PASS BY REFERENCE OR BY VALUE

Pass by reference

```
void foo(int &var){
  var++;
}
```

Pass by value

```
void foo(int var){
  var++;
}
```

ARRAYS

- Ordered set of fixed No. of elements

ARRAYS

- Ordered set of fixed No. of elements
- Contiguous memory locations

ARRAYS

- Ordered set of fixed No. of elements
- Contiguous memory locations
- Passed by reference

ARRAYS

- Ordered set of fixed No. of elements
- Contiguous memory locations
- Passed by reference

Declaration

type name[*size*];

Example

```
float myArray[5] = {1.1,3.2,-1.5,7.3,-9.0};
```

ARRAYS

Get element

```
float a = myArray[2];
```

ARRAYS

Get element

```
float a = myArray[2];
```

Set element

```
float b = -3.3;
myArray[0] = b;
```

MEMORY ALLOCATION

Memory can be dynamically allocated from the heap.

MEMORY ALLOCATION

Memory can be dynamically allocated from the heap.

Memory allocation / deallocation

```
type *name = new type[size];
delete[] name;
```

MEMORY ALLOCATION

Dynamically allocated memory can be accessed using an index.

Example

```
int size = 50;
double *myDynarr = new double [size];
for(int i = 0; i < size; i++){
  myDynarr[i] = 0.;
}
delete[] myDynarr;
```

BIBLIOGRAPHY

1. *Programming Abstractions in C++*, E.S. Roberts, Pearson (2014)
2. *Effective Modern C++*, S. Meyers, O'Reilly (2015)