Data Structures

CSCI 2270-202: REC 02

Sanskar Katiyar

Office Hours

Office Hours at ECAE 128 (Aerospace Lobby)

Tuesday: 12:15 pm – 2:15 pm

Friday: 1:30 pm – 3:30 pm

Did you locate the Office Hours Calendar?

{TA, CM, Instructor}s, {CA}s have ***separate calendars*** (Check both)

Before We Start

Is your development environment set up?

If not, come to office hours; we will figure it out

Have you started Assignment 1?

Q1 Approach?

Textbook?

Assignment 1: FAQs

Coderunner Issues

- Very picky about output format (Don’t print extra lines, commas)
- File Opening Issues: Use **`ifstream`** (Yoshi’s answer on Piazza **@36**)
- **`argv[i]`**: Using argument(s) for filename
- **`stof(), stoi()`**
- **`return 0`**;

Assignment 1: FAQs

Reading CSV file

`getline()`: supports reading with delimiters

```
123.45,Sample,25
22,Test,9
.
.
```

```
string s; float r;

ifstream fp("filename.txt");

getline(fp, s, ',');  // s = "123.45"

r = stof(s); // Why??
```

Recitation Outline

1. Memory: logical representation
2. Address-of Operator (*a.k.a* reference)
3. Pass-by-{value, reference}
4. Pointers
5. Structures
6. Exercise

Memory

Logical Representation

Memory: Logical Representation

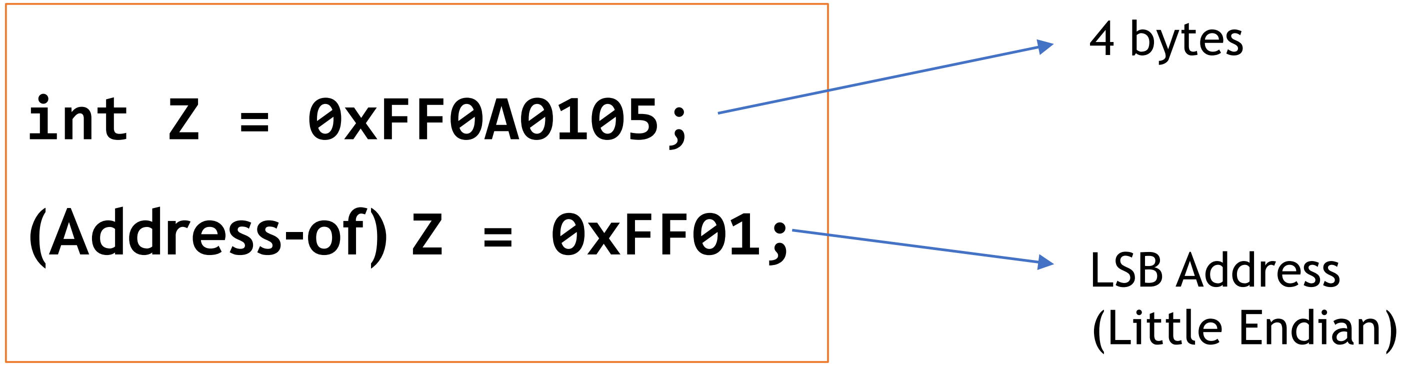
Little Endian

Address	Value	Variable
…		
0xFF08		
0xFF07		
0xFF06		
0xFF05		
0xFF04	0xFF	Z
0xFF03	0x0A	
0xFF02	0x01	
0xFF01	0x05	
…		

Computers understand binary (base = 2)

Hexadecimal number system (base = 16)

Easier to represent: $(1010)_2$ -> $(A)_{16}$



```
int Z = 0xFF0A0105;
(Address-of) Z = 0xFF01;
```

Memory: Logical Representation

Different data-types have different sizes

Thus, occupy more/less space in memory

Table for reference: ***Not universal!***

- Microprocessor architecture
- Compiler, etc.

C++ Type	Size (in bytes)
int	4
char	1
float	4
long	8
double	8

Address-of (&) Operator

a.k.a References

Address-of (&) Operator

Little Endian

Address	Value	Variable
…		
0xFF08		
0xFF07		
0xFF06		
0xFF05		
0xFF04	0xFF	Z
0xFF03	0x0A	
0xFF02	0x01	
0xFF01	0x05	
…		

`int Z = 0xFF0A0105;`

(Address-of) `Z is 0xFF01;` (Little Endian)

&

&Z = 0xFF01;

One memory location is enough to determine the entire content, since type is known

Address-of (&) Operator

a.k.a
Reference

Code

```
#include <iostream>
using namespace std ;
int main ()
{
    int a = 10 ;
    cout << a << endl;
    cout << &a << endl;
    return 0 ;
}
```

Output

```
10
0x7ffccbbcd804
```

Pass-by-{value, reference}

Function arguments

Pass-by-value

Code

```
...
void add_val(int num)
{
    num = num + 2 ;
    cout << num << endl;
}
int main ()
{
    int a = 10;
    add_val(a);
    cout << a;
}
```

Output

```
12
10
```

Pass-by-value

```
...
void add_val(int num)
{
    num = num + 2 ;
    cout << num << endl;
}
int main ()
{
    int a = 10;
    add_val(a);
    cout << a;
}
```

***Recall*: Function Scope**

On function call, VALUE of *a* is copied over to ***num***

num is local to ***add_val***'s scope

Any changes made to the arguments are **local** to the function

Helpful to think as: **`add_val(10)`**

Pass-by-reference

Code

```
...
void add_ref(int &num)
{
    num = num + 2 ;
    cout << num << endl;
}
int main ()
{
    int a = 10;
    add_ref(a);
    cout << a;
}
```

Output

```
12
12
```

Pass-by-reference

```
...
void add_ref(int &num)
{
    num = num + 2 ;
    cout << num << endl;
}
int main ()
{
    int a = 10;
    add_ref(a);
    cout << a;
}
```

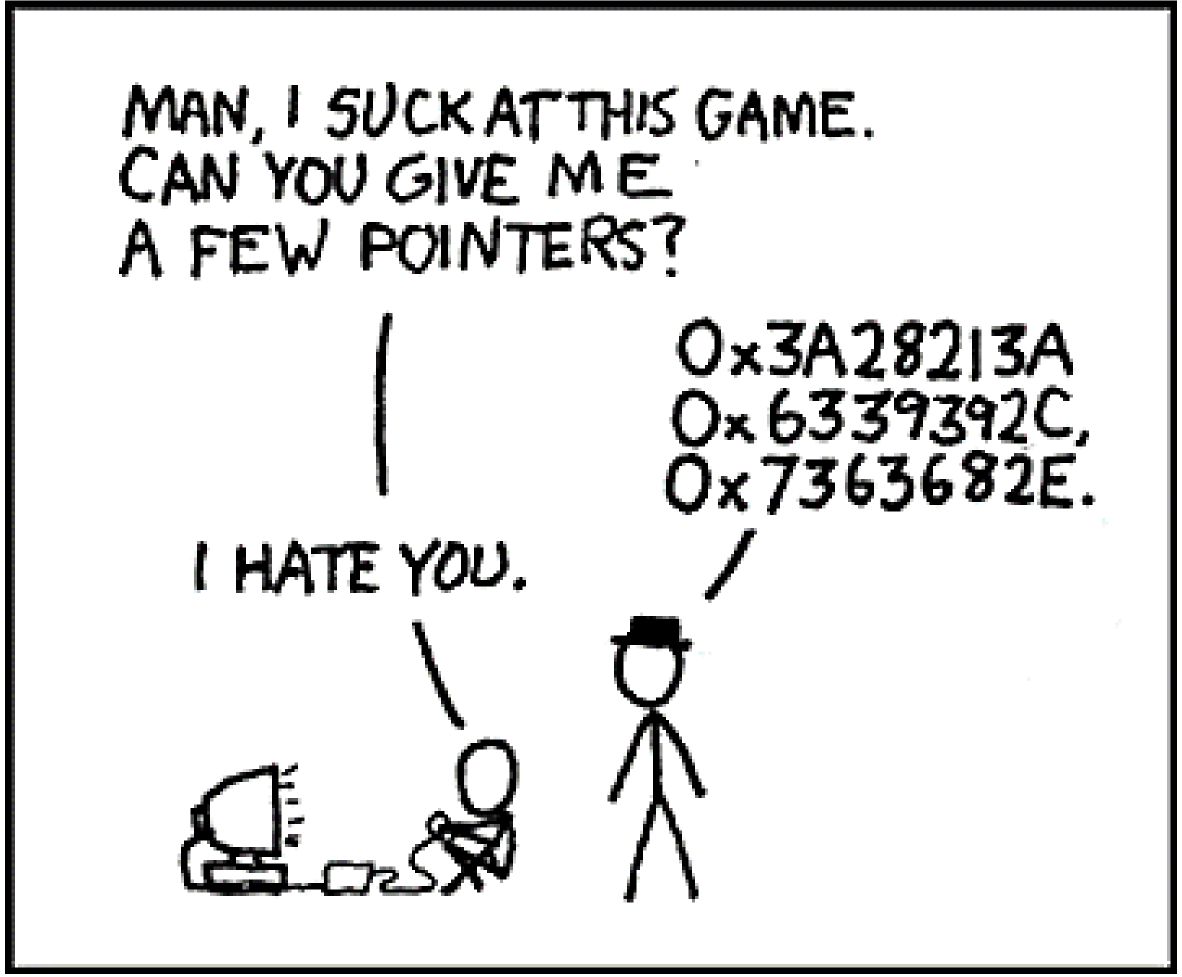
***Recall*: Address-of, Memory**

On function call, ADDRESS of ***a*** is copied to ***num***

Thus, ***num*** and ***a*** refer to the same memory address

Any changes made to the reference arguments are **persistent** even outside the function too

Source: https://xkcd.com/138/

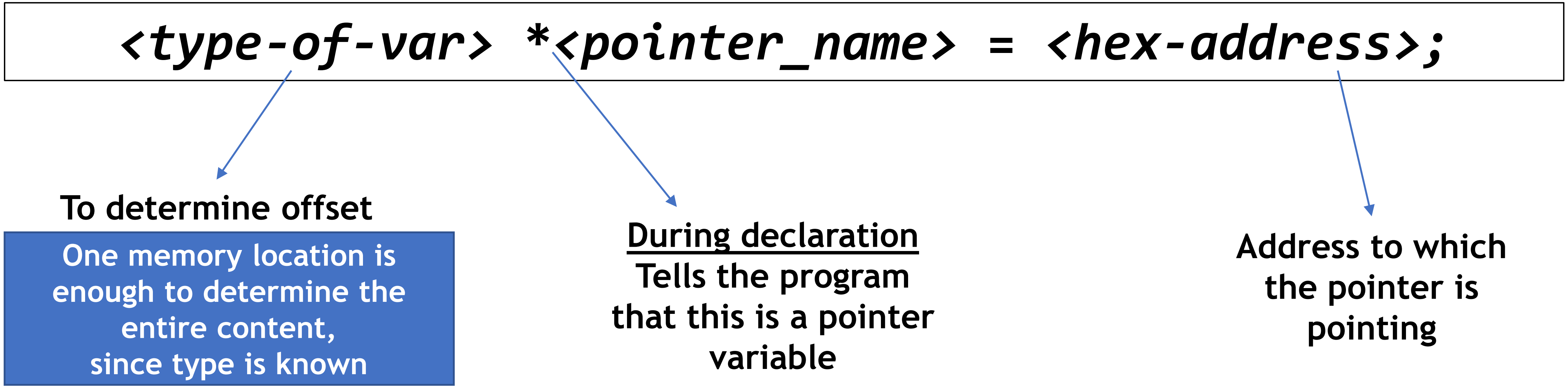


Pointers

Pointers

Special variable that store a memory address

Declaration:



```
<type-of-var> *<pointer_name> = <hex-address>;
```

To determine offset

One memory location is enough to determine the entire content, since type is known

<u>During declaration</u>
Tells the program that this is a pointer variable

Address to which the pointer is pointing

Pointers

```
int Z = 0xFF0A0105;
int *ptrZ = &Z;
```

Pointers are of fixed size. ***Reason?***

The fixed size is machine dependent, however.

Can there be a pointer to a pointer?

Yes, a pointer is also a variable (stored in memory)

int* *ptrZ (or ***int** ptrZ***, or ***int **ptrZ***)

Address	Value	Variable
. . .		
0xFF0E		
0xFF0D	0x00	**ptrZ**
0xFF0C	0x00	
0xFF0B	0xFF	
0xFF0A	0x01	
. . .		
0xFF05		
0xFF04	0xFF	**Z**
0xFF03	0x0A	
0xFF02	0x01	
0xFF01	0x05	
…		

Little Endian

De-referencing (*) Operator

Asterisk (*) has two **roles *w.r.t.* pointers**:

- ***Declaration***: tells program this is a pointer
- ***Otherwise***: Accesses the contents at the memory location

Let’s look at a complete example

Pointers

Code

```
...
int a = 10 ;
int *p = &a;

cout << p << ", " << *p << endl;

*p = *p + 2;

cout << a << ", " << *p << endl;
```

Output

```
0x7ffccbbcd804, 10
12, 12
```

Pass-by-pointer

Code

```
...
void add_ptr(int *num)
{
    *num = *num + 2 ;
    cout << *num << endl;
}
int main () {
    int a = 10; int *b = &a;
    add_ptr(&a); // pass address-of(a)
    add_ptr(b); // pass pointer
    cout << a;
}
```

Output

```
12
14
14
```

Pointers: Arrays

Array is **stored contiguously** in memory

Compiler allocates **`<size-of-datatype> X <number-of-elements>`**

Pointer to an array:

points to the first element (or zero index) of the array. ***Why?***

Recall:

One memory location is enough to determine the entire content since type is known

Pointers: Arrays

int (4 bytes)

Why C/C++ arrays start at 0-index?

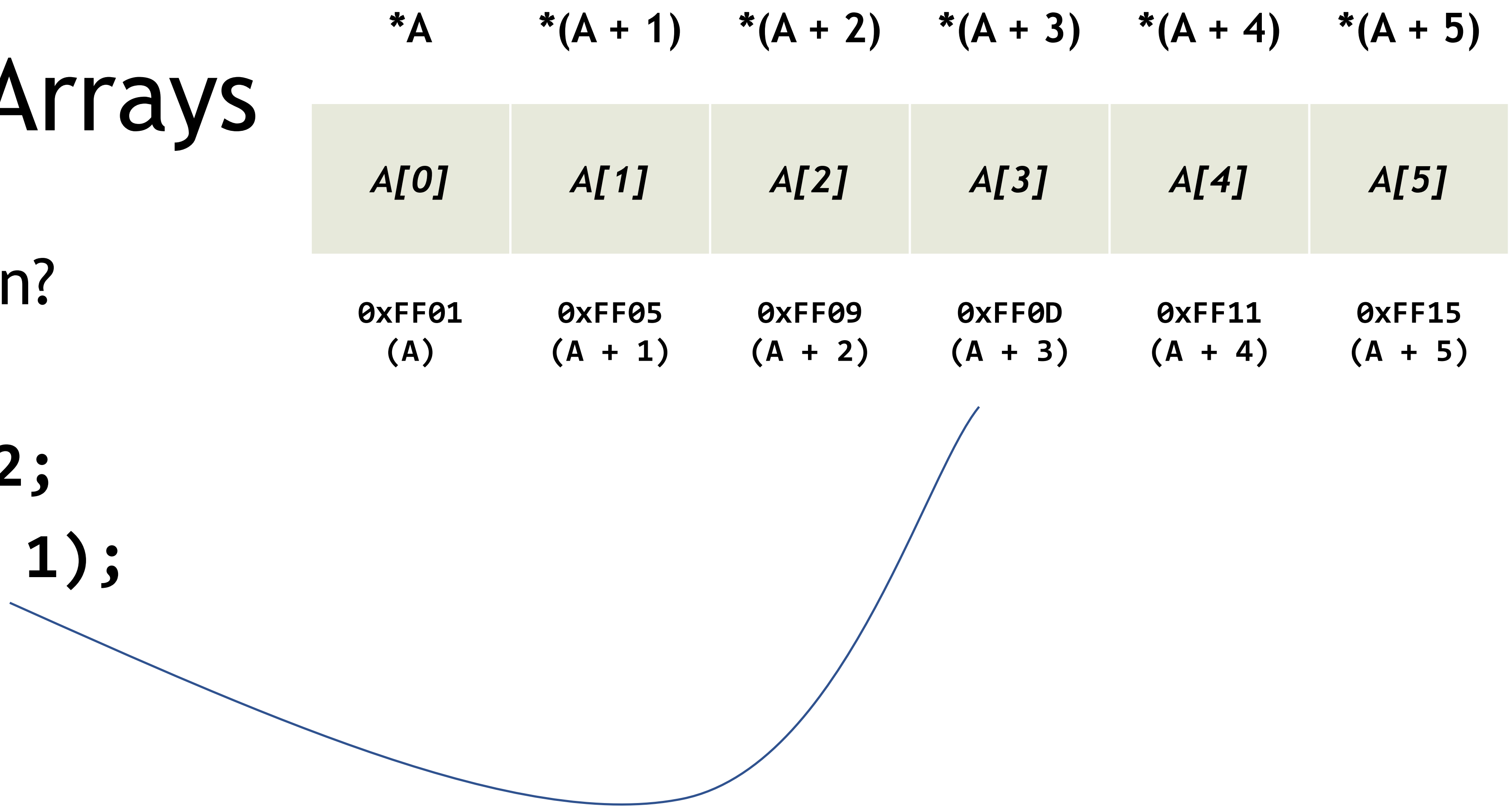
Dereferenced Pointers (Indexing)	*A	*(A + 1)	*(A + 2)	*(A + 3)	*(A + 4)	*(A + 5)
Indexing Elements	*A[0]*	*A[1]*	*A[2]*	*A[3]*	*A[4]*	*A[5]*

Address (Pointers)	`0xFF01`
	`(A)`
		`0xFF05`
		`(A + 1)`
			`0xFF09`
			`(A + 2)`
				`0xFF0D`
				`(A + 3)`
					`0xFF11`
					`(A + 4)`
						`0xFF15`
						`(A + 5)`

Pointers: Arrays

What will happen?

```
int *z = A + 2;
cout << *(z + 1);
```



*A	*(A + 1)	*(A + 2)	*(A + 3)	*(A + 4)	*(A + 5)
A[0]	*A[1]*	*A[2]*	*A[3]*	*A[4]*	*A[5]*
0xFF01 (A)	0xFF05 (A + 1)	0xFF09 (A + 2)	0xFF0D (A + 3)	0xFF11 (A + 4)	0xFF15 (A + 5)

Pointers: Arrays

Need to know **how many elements in the array**

Otherwise may access memory address which is outside array

```
int main(int argc, char* argv[])
```

argv as array of pointers to type *char*
(point to first char)

```
int main(int argc, char** argv)
```

argv as pointer to type *char**
(character strings)

Structure

Structure: Overview

A **structure** is a:

1. *Composite*: can be composed of multiple members, different types
2. *User-defined*: what members, which types?
3. *Data-type*

Once defined, **behaves like any other data type**

Pass in functions, define arrays, etc.

(C-style) Only **contains data members**, no functions

Structure: Define, Declare, Initialize

```
…
struct student
{
   string name;
   string email;
   int birthyear;
   string address;
};
int main() {
student stu = {"ABC", "abc@colorado.edu", 1987, "Boulder CO"};
…
```

Structure: Dot operator

Code

```
…
student stu; //student defined


stu.address = "Boulder, CO";
stu.email = "abc@colorado.edu";
stu.birthyear = 1987;
stu.name = "ABC";


cout << stu.name << endl;
cout << stu.email << endl;
cout << stu.birthyear << endl;
cout << stu.address << endl;
```

Output

```
ABC
abc@colorado.edu
1987
Boulder CO
```

Structure: Pointer-to-struct (->)

Code

```
…
student *stu1;
stu1 = &stu; //stu exists

stu1->name = "XYZ";
cout << stu1->name << endl;
cout << stu1->email << endl;
cout << stu1->birthyear << endl;
cout << stu1->address << endl;
```

Output

```
XYZ
abc@colorado.edu
1987
Boulder CO
```

Exercise

Exercise: Overview

Download ***Lab2.zip*** (on Moodle)

Complete the **TODOs** in *swap.cpp*, *main.cpp*

Compile: `g++ main.cpp swap.cpp –std=c++11`

Exercise: Review

Dereferenced Pointers (Indexing)	*A	*(A + 1)	*(A + 2)	*(A + 3)	*(A + 4)	*(A + 5)
Indexing Elements	*A[0]*	*A[1]*	*A[2]*	*A[3]*	*A[4]*	*A[5]*
Address (Pointers)	0xFF01 (A)	0xFF05 (A + 1)	0xFF09 (A + 2)	0xFF0D (A + 3)	0xFF11 (A + 4)	0xFF15 (A + 5)