UCLA Samueli Computer Science

CS31 Week 7 Discussion

Fall 2021, Section 1C and 2A

Mingyu Derek Ma mdma@ucla.edu

Thanks Muhao Chen and Rosa Garza for their shared content

https://derek.ma/cs31 for slides and other discussion materials

Reminder

- Project 5, Monday Nov 15, 11pm
- Mid-term 2 Nov 16

Project 4 Feedback

- Comment your program logic, especially for complicated functions
- Need to provide concrete test cases, rather than high-level design thoughts about test cases
- Need to have test cases for all functions
- Need to have brief reason for your test cases

Project 5 Suggestions

- Variable-length array is not allowed
  - g++ extension of variable-length arrays won’t compile under g31
- All arrays must have bounds known at the compile time

Midterm 2

- Cover material up to C strings
  - No pointers, no structs, no classes

Pointer

- Pointer: a variable that holds the address of another variable in the memory
- Memory like boxes that can be used to save information
- Each box has an address
- Example: for a 32-bit machine

Addresses:	Value
0x7ffeefbff550	20
0x7ffeefbff554	
0x7ffeefbff558	2
0x7ffeefbff55c	10
0x7ffeefbff560	address 0x…550

Pointer

- Declare a pointer
  - <data_type> * <pointer_name> [=<initialization>];
  - <data_type>: what type of value is pointed by the pointer

  - Examples:
  - int *ptr;
  - double *p, *q;
  - double *p, *q, r;

Pointer

- Point a pointer to a regular variable
  - &<variable_name>
  - Example
    - int a;
    - int *ptr = &a;
- Get the value at the address indicated by the pointer
  - *<pointer_name>
  - Example
    - int b = *ptr;
- * and & are two memory operations

* operator (dereference)

- Using * before an already-initialized pointer to dereference, i.e. to get the value stored at this address

```
int a = 5, *p;
p = &a;
cout << p << endl;
cout << *p << endl;
```

```
0x16fdff35c
5
```

* operator (dereference)

```
double x, y;    // normal double variables
double *p;      // a pointer to a double variable
x = 5.5;
y = -10.0;
p = &x;         // assign x's memory address to p
cout << "p: " << p << endl;
cout << "*p: " << *p << endl;
p = &y;         // assign y's memory address to p
cout << "p: " << p << endl;
cout << "*p: " << *p << endl;
```

```
p: 0x16fdff358
*p: 5.5
p: 0x16fdff350
*p: -10
```

& operator (reference)

- Used before a variable to get the address of a variable
- Inverted operator of *

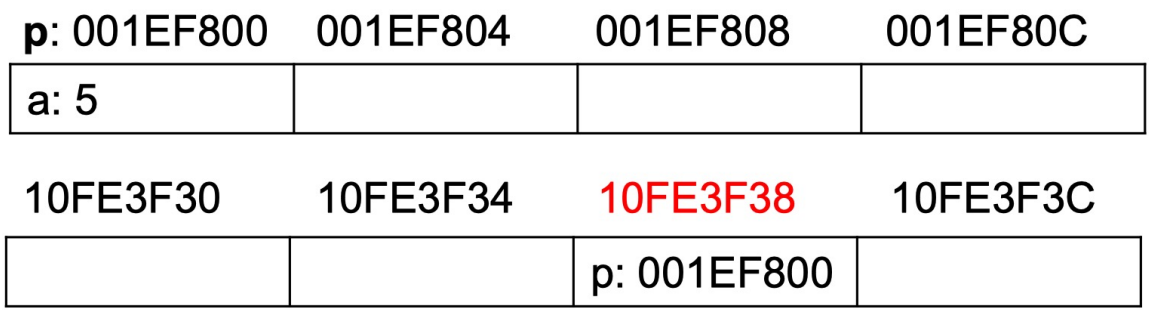
```
int a = 5, *p;
p = &a;
cout << p << endl;
cout << *p << endl;
cout << a << endl;
cout << &a << endl;
cout << *&a << endl;    // same as a
cout << *&*&a << endl; // same as a
```

```
0x16fdff35c
5
5
0x16fdff35c
5
5
```

Does a pointer have an address?

- Pointer is also a kind of variable, and stored in the memory

```
int a = 5, *p;
p = &a;
cout << &a << endl;
cout << &p << endl;
cout << *&p << endl;
cout << **&p << endl;
```



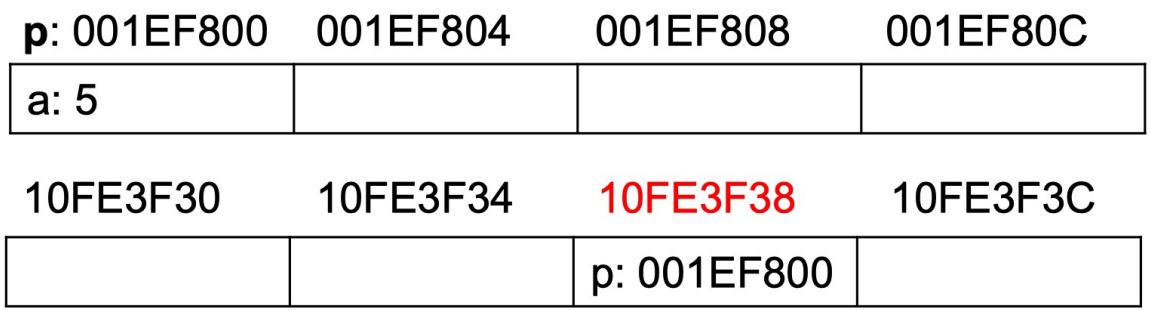
Does a pointer have an address?

- Pointer is also a kind of variable, and stored in the memory

```
int a = 5, *p;
p = &a;
cout << &a << endl;
cout << &p << endl;
cout << *&p << endl;
cout << **&p << endl;
```

Output:

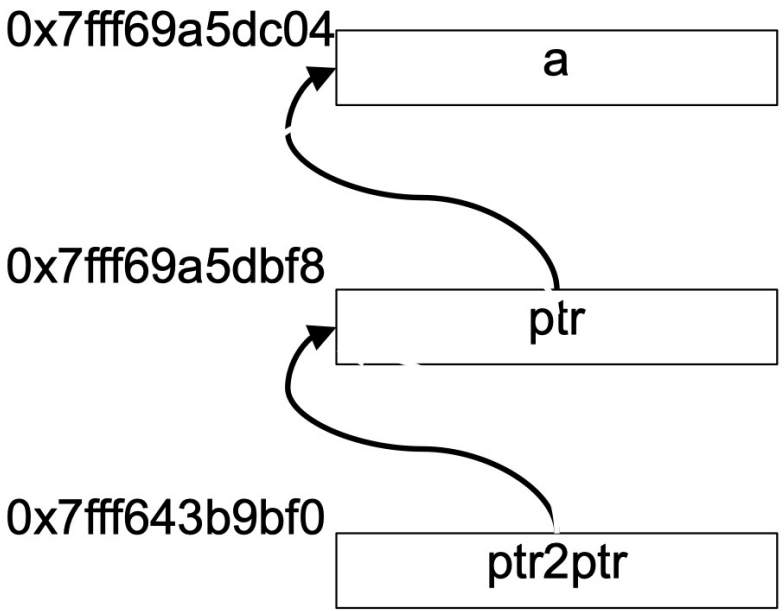
```
001EF800 <- a's address
10FE3F38 <- p's address
001EF800 <- value at p's address
5 <- value at the *&p address
```



Pointers of pointers

- Pointer is a type of variable, so a pointer can point to another pointer

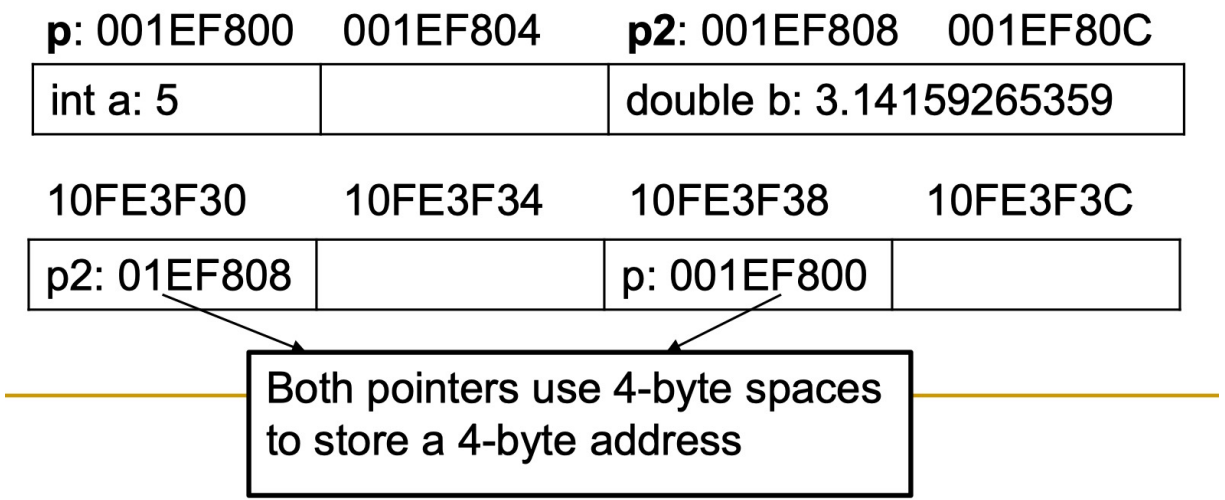
```
int a = 5;
int *ptr = &a;
int **ptr2ptr = &ptr;
```



Size of a pointer

- 4 bytes or 8 bytes
  - Depends on your system environment (32-bit system or 64-bit system)
- Same size regardless of what type of variable it points to

```
int *p=&a;
double *p2=&b;
```

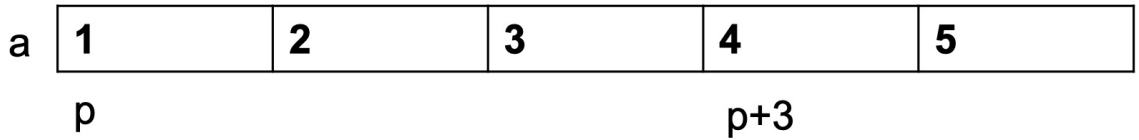


“Move” a pointer

- Perform arithmetic operations on a pointer to point to other address

```
int a[5] = {1,2,3,4,5};
int *p = a; // or p=&a[0];
cout << *p << endl;
cout << *(p+3) << endl;
p++;
cout << *p << endl;
```

```
1
4
2
```



“Move” a pointer

- */& operator has higher priority than regular arithmetic operations (* / % + -)
- Priority of ++ is higher than * (+ << * << ++)
- More info: [C++ built-in operators, precedence, and associativity | Microsoft Docs]

```
int a[2] = {0, 100};
int *p = &a[0];
cout << *(p + 1) << endl; // give us 100
cout << *p + 1 << endl;   // give us 1
```

Example: invert a C-string using pointers

```
char s[]="abcdefg";
char t[100];
char *p = &s[strlen(s) - 1]; // point p to the last char of s
char *q = &t[0];             // point q to the first char of t
while (p >= &s[0]){ // while p doesn't go before the first char of s
    *q = *p; // get the content pointed by p to that of q
    p--; // p moves left
    q++; // q moves right
}
*q = '\0';
cout << t << endl;
```

gfedcba

s	a	b	c	d	e	f	g	\0

t	g							

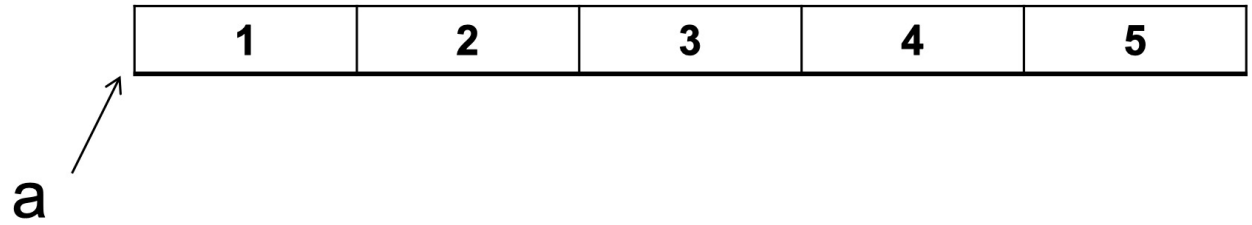
Null pointer

- A null pointer indicates the pointer does not point to anything (point to 0)

```
int *p;
p = 0;          // style 1
p = NULL;       // style 2
p = nullptr; // style 3
```

Array is a kind of pointer

- Array is a fixed pointer that points to the first element of the array
  - int a[] = {1,2,3,4,5};
  - a is the same as &a[0]
  - *(a+1) is equivalent to a[1]



Thank You