Tutorial 5
Pointers and Arrays

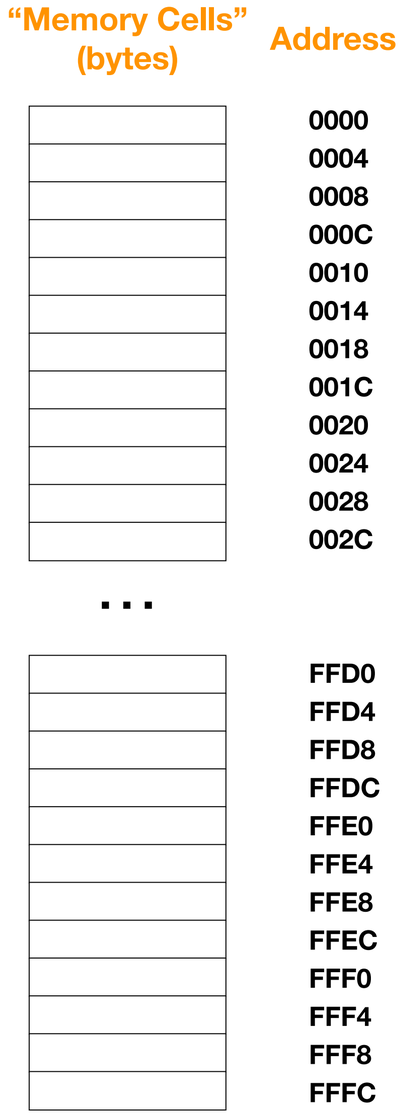
Lanruo Xia (USTF)
122030080@link.cuhk.edu.cn

Contents

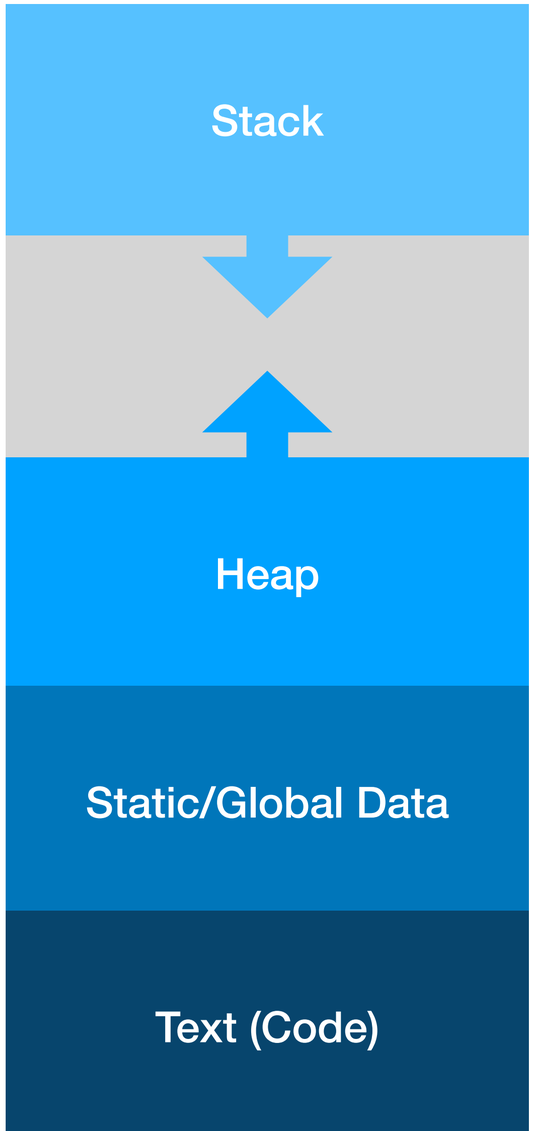
What is Memory and How It works

- Memory: an electronic component of a computer to store data and program for immediate use in computer

- Abstraction: Memory can be seen as a long continuous sequences of bytes
- Each byte has a unique address and is addressable
- We use an address to locate a piece of data stored in memory



A Program's Memory



- When you execute a program, the program is loaded into memory
- The memory allocated to a program is divided into several segments
- You will learn more about them in later lectures/tutorials

Variable Stored in Memory

```
int a = 20241015
```

For example, integer variable a is stored in 0x006DFE81

The size of an integer variable is 4 bytes which is 32 bits in memory

20241015 in decimal = 00000001001101001101101001110111 in binary

Memory Address	Value of variable a stored in memory
0x006DFE81	0 0 0 0 0 0 0 1
0x006DFE82	0 0 1 1 0 1 0 0
0x006DFE83	1 1 0 1 1 0 1 0
0x006DFE84	0 1 1 1 0 1 1 1

1 bit

4 bytes (32 bits) memory to store an integer variable

Variable Stored in Memory — Size?

- The memory space required to store value of a is 4 bytes (32 bits), since it is an integer variable

```
int a = 20241015;

sizeof(a); // output:4
```

- Variables of different types may take up different amount of space in memory

A character in C/C++ takes up 1 byte of memory space.
A double uses 8 bytes

```
char b = 'B';

sizeof(b); // output:1
```

- sizeof(x) return the number of bytes to store a variable x

```
double c = 3.14;

sizeof(c); // output:8
```

How to access and manipulate memory in C/C++

- We need a variable to store the memory address
- C/C++ provides the variable for us: we can use pointers!
- A pointer is a variable that stores the memory address of another variable
- The value of a pointer is basically an address of memory (an integer)

```
int a = 20241015
```

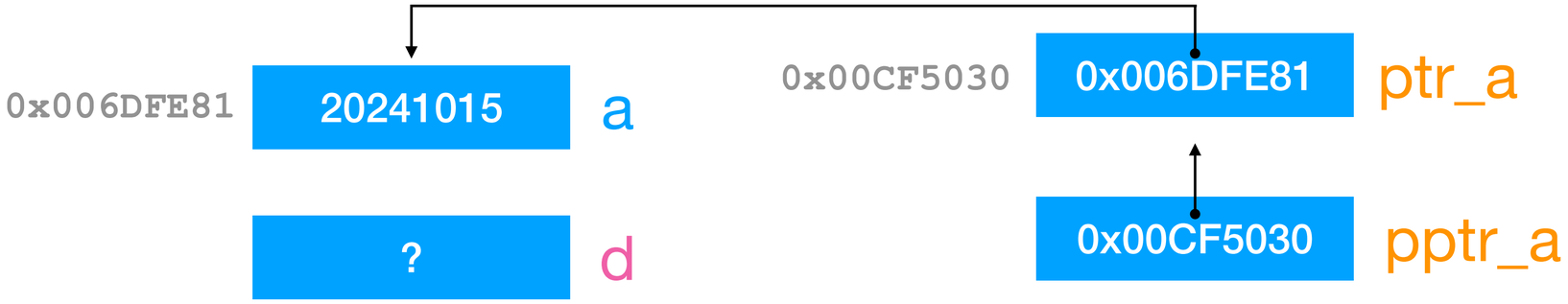
We want a pointer that points to the memory of a, so that there is a way for us to access a in memory

The value of the pointer should be 0x006DFE81 in this example

Pointer Declaration and Operators

- Declaration: `type * var`
- Pointer Operators:
- **&** operator: return address of an variable **(address-of)**
- * operator (used before a pointer variable): return the actual value of the variable that a pointer points to **(dereferencing)**
- **Declare a pointer to another pointer?**

```
int a = 20241015;

int * ptr_a; // declare a pointer to integer

ptr_a = &a; // assign address of a to the pointer

// Now, ptr_a = 0x006DFE81

int d = *ptr_a; // dereference ptr_a

// Now, d = 20241015

int ** pptr_a = &ptr_a;
//the value of pptr_a is 0x00CF5030 (the address of
ptr_a)
```



Reference in C++

- A reference variable is a reference to an existing variable. It is an alias for that variable.

- Reference can be used in passing arguments to a function. It can avoid creating a new copy of structures which is a waste of memory and performance.

- A reference must be initialized and assigned to an object when declared. Once a reference is created, it cannot be used to reference another object

```
int a = 20241015;
int& r = a; ✅
// r = 20241015

int& r; ❌

int a = 20241015;
int b = 3
int& r = a
r = b
// r is not a reference to b.
Instead, the value of b is
assigned to r
// Now, r = 3; a = 3
```

Passing Arguments by Value/Pointer/Reference to a function

- Pass by value: When a variable is passed to a function by value, a copy of that variable is created into the stack in memory. The original variable will not be affected.

- Pass by pointer: The memory address of a variable is passed to the parameter in the function. The value of that variable can be changed in the function by dereference.

- Pass by reference: If a variable is passed to a function by reference, the function can modify the value of the variable by using its reference passed in.

Pass by Value, Pass by Pointer, Pass by reference

```c
void swap_by_value(int x, int y) {
    int tmp = x;
    x = y;
    y = tmp;
}

void swap_by_pointer(int * px, int * py) {
    int tmp = *px;
    *px = *py;
    *py = tmp;
}

void swap_by_reference(int & x, int & y) {
    int tmp = x;
    x = y;
    y = tmp;
}
```

```c
int main(){
    int n1 = 1;
    int n2 = 2;
    swap_by_value(n1, n2);
    printf("passing by value: n1 = %d, n2 = %d\n", n1, n2);
    n1 = 1;
    n2 = 2;
    swap_by_pointer(&n1, &n2);
    printf("passing by pointer: n1 = %d, n2 = %d\n", n1, n2);
    n1 = 1;
    n2 = 2;
    swap_by_reference(n1, n2);
    printf("passing by reference: n1 = %d, n2 = %d\n", n1, n2);
}
```

```
lanruo@ubuntu20:~/Documents/CSC3002/tutorial5$ g++ Swap.cpp -o Swap
lanruo@ubuntu20:~/Documents/CSC3002/tutorial5$ ./Swap
passing by value: n1 = 1, n2 = 2
passing by pointer: n1 = 2, n2 = 1
passing by reference: n1 = 2, n2 = 1
lanruo@ubuntu20:~/Documents/CSC3002/tutorial5$
```

(You may find these codes on Blackboard: Swap.cpp)

Pass by Value, Pass by Pointer, Pass by reference

```cpp
// Call by value
void increment_1(int a){
    a++;
    std::cout << "Pass by value:"<< std::endl;
    std::cout << "a = " << a
        << " The address of a is "
        << &a << std::endl;
}

// Call by pointer
void increment_2(int* a){
    (*a)++;
    std::cout << "Pass by pointer:"<< std::endl;
    std::cout << "(*a) = " << *a
    << " The address of (*a) is "
    << a << std::endl;
}
// Call by reference
void increment_3(int& a){
    a++;
    std::cout << "Pass by reference:"<< std::endl;
    std::cout << "a = " << a
    << " The address of a is "
    << &a << std::endl;
}
```

```cpp
int main(){
    int x = 1;
    std::cout << "x = " << x << " The address of x is "
     << &x << std::endl;
    printf("----------------------------------------\n");
    increment_1(x);
    printf("After increment: x = %d\n", x);
    printf("----------------------------------------\n");
    x = 1;
    increment_2(&x);
    printf("After increment: x = %d\n", x);
    printf("----------------------------------------\n");
    x = 1;
    increment_3(x);
    printf("After increment: x = %d\n", x);

}
```

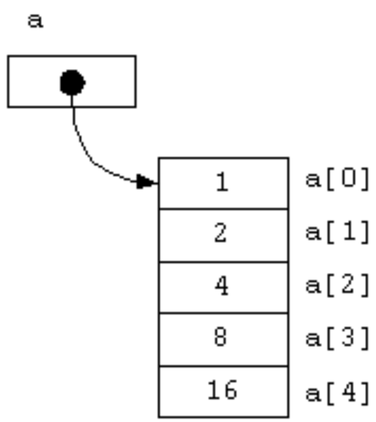
```
lanruo@ubuntu20:~/Documents/CSC3002/tutorial5$ g++ Increment.cpp -o Increment
lanruo@ubuntu20:~/Documents/CSC3002/tutorial5$ ./Increment
x = 1 The address of x is 0xffffe7ac18f4
----------------------------------------
Pass by value:
a = 2 The address of a is 0xffffe7ac18dc
After increment: x = 1
----------------------------------------
Pass by pointer:
(*a) = 2 The address of (*a) is 0xffffe7ac18f4
After increment: x = 2
----------------------------------------
Pass by reference:
a = 2 The address of a is 0xffffe7ac18f4
After increment: x = 2
```

(You may find these codes on Blackboard: Increment.cpp)

C/C++ Arrays

- Arrays in C/C++ are used to store multiple variables of the same type in a single variable

- An array is stored at a contiguous memory location




```
Array Initialization

int a [5] // Assignment values to elements later

Or

int a [5] = { 1, 2, 4, 8, 16 }

// multidimensional arrays: (2 rows x 3 columns)

int arr[2][3] = { 10, 20, 30, 40, 50, 60 }
```

Properties of C/C++ Arrays

- Fixed Size: The size of an array must be known at the compile time and is fixed once the array is declared.

- No *Index Out of Bounds* Checking: You will access other parts of the memory if you access array with an out-of-bounds index (can be dangerous). Bounds checking of array should be maintained manually.

```
int main(){
    int arr[5] = {1, 2, 3, 4, 5};
    std::cout << arr[-1] << std::endl; // array index out of bounds: undefined behavior
}

lanruo@ubuntu20:~/Documents/CSC3002/tutorial5$ g++ demo.cpp -o demo
lanruo@ubuntu20:~/Documents/CSC3002/tutorial5$ ./demo
output: 65535
lanruo@ubuntu20:~/Documents/CSC3002/tutorial5$
```

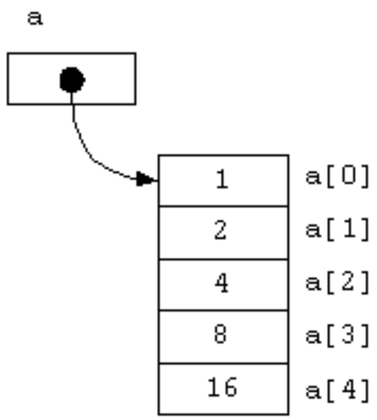
- Get length of an array:

```
int a [5] = { 1, 2, 4, 8, 16 }
int len = sizeof(a)/sizeof(*a)
```

20 (bytes) 4 (bytes)

Relationship between Arrays and Pointers

- The name of an array is a pointer to the first element of the array




```
int a [5] = { 1, 2, 4, 8, 16 }
int * ptr = a // ptr = &a[0]
*(ptr + 2) = 3 // is the same as a[2] = 3
```

Pointer Arithmetic

Pass/Return an array

```c
void passArray(int arr[], int size)
{
    int i;

    for(i = 0; i < size; i++)
    {
        printf("%d, ", arr[i]);
    }
}
```

```c
void passArray(int *arr, int size)
{
    int i;

    for(i = 0; i < size; i++)
    {
        printf("%d, ", arr[i]);
    }
}
```

```c
int* returnArray()
{
    static int arr[5] = { 1, 2, 3, 4, 5 };

    return arr;
}
int main()
{
    int i;

    int * a;
    a = returnArray();

    for (i = 0; i < 5; ++i)
    {
        printf("%d\n", a[i]);
    }

    return 0;
}
```

Exercise

- Try compiling and running PointerAndAddress.cpp by yourself. (You may find this on Blackboard)
- You may print out different results for addresses each time you run the codes, but what do you observe from the results printed out?
- How does the address of an array element change from doubleArray[i] to doubleArray[I+1]?
- What does (doubleArray + 1) mean?
- How to use pointer to access each array element?
- What is the difference between *doubleArray + 1 and *(doubleArray + 1)?
- More observations?

```
double doubleArray[] = {0, 2, 4, 6, 8,
10, 12, 14, 16, 18};
double* doublePointer = doubleArray;
```

```
&doubleArray[0]: 006DFE80
*doubleArray: 00000000
doubleArray[0]: 00000000
doubleArray+1: 006DFE88
&doubleArray[1]: 006DFE88
*doubleArray+1: 00000001
*(doubleArray+1): 00000002
doubleArray[1]: 00000002
doubleArray+9: 006DFEC8
&doubleArray[9]: 006DFEC8
*(doubleArray+9): 00000012
doubleArray[9]: 00000012
doubleArray+10: 006DFED0
&doubleArray[10]: 006DFED0
*(doubleArray+10): 00000000
doubleArray[10]: 00000000
doubleArray-1: 006DFE78
*doubleArray-1: FFFFFFFF
*(doubleArray-1): 00000000
&doubleArray: 006DFE80
&doubleArray+1: 006DFED0
*(&doubleArray+1): 006DFED0
&doubleArray-1: 006DFE30
*(&doubleArray-1): 006DFE30
```

Thank you!

Q & A