南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

C/C++ Program Design

CS205

Week 5

Prof. Shiqi Yu (于仕琪)
<yusq@sustech.edu.cn>
Prof. Feng Zheng(郑锋)
<zhengf@sustech.edu.cn>

Content

- Pointer
- Allocate Memory: C Style
- Allocate Memory: C++ Style
- Managing memory for data

The slides are based on the book <Stephen Prata, C++ Primer Plus, 6th Edition, Addison-Wesley Professional, 2011>

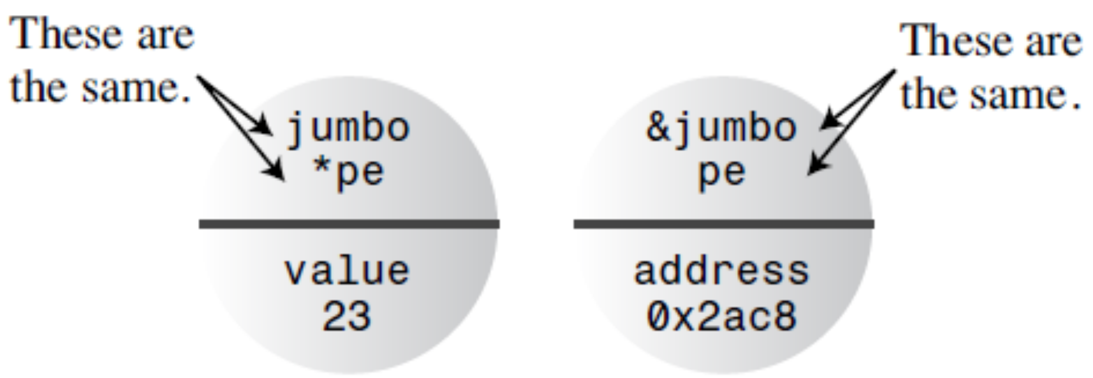
Pointers

What's a pointer?

address

- Three fundamental properties of declaration
  - Where the information is stored
  - What value is kept there
  - What type of information is stored
- How to know where the values are stored?
  - Using address operator & to access the address
  - Using hexadecimal notation to display the address values
  - Run program example address.cpp

```
int jumbo = 23;
int * pe = &jumbo;
```



Pointer Type

- Using ordinary variables
  - Naturally, the value is treated as a named quantity
  - The location as the derived quantity
- Using new strategy: pointer type
  - Inverse way
- Operator of asterisk * :
  - Indirect value
  - The dereferencing operator
- Program example pointer.cpp

Importance of pointers

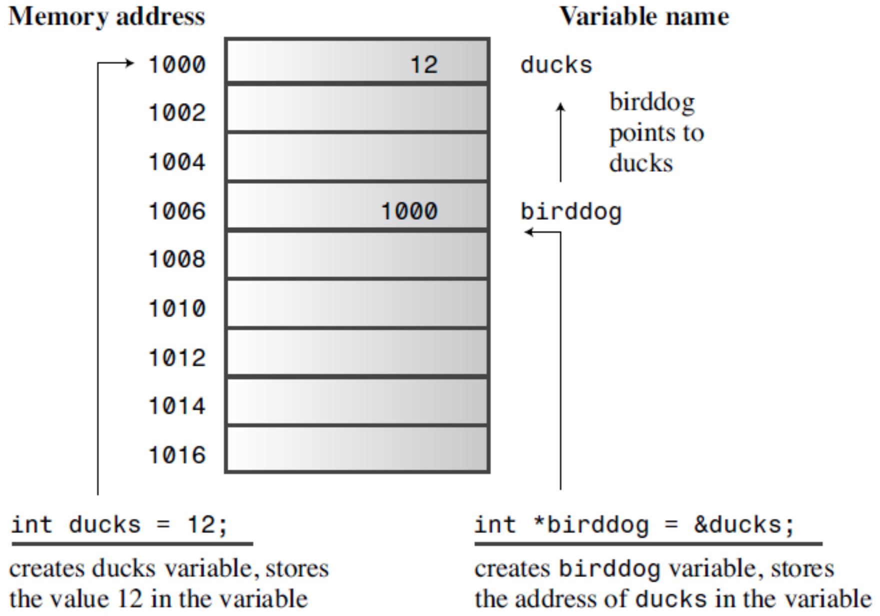
- One essential to the C/C++ programming philosophy of is the **memory management**

- Pointers would be the C/C++ Philosophy

- You can access memory more directly than Java, Python, etc to gain efficiency.

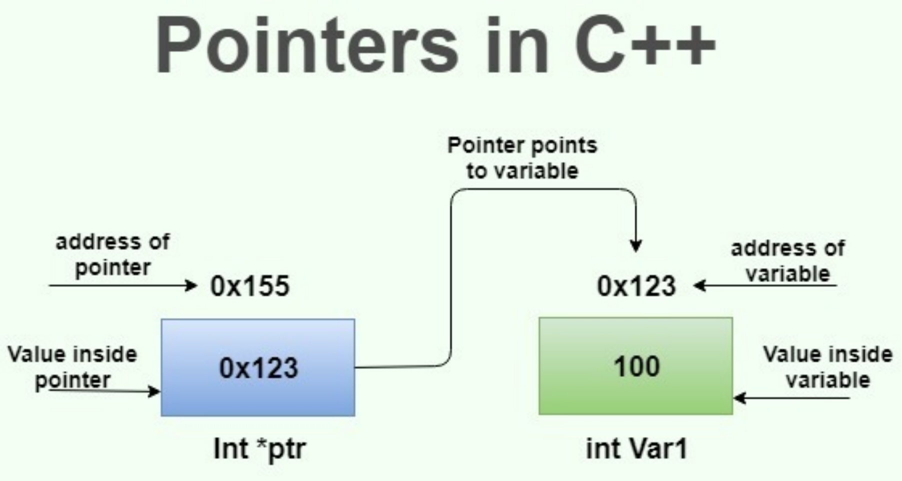
Declaring and Initializing Pointers

- Example: int* birddog;
  - ➢ * birddog is a int type variable
  - ➢ birddog is a pointer type variable
  - ➢ The type for birddog is pointer-to-int
  - ➢ Put the white space before or behind the * or no spaces
- int * is a compound type
  - ➢ double *, float *, char *



Pointer Danger

- A confusion for beginners
  - ➢ Creating a pointer in C++ means the computer allocates memory to hold an address
  - ➢ BUT it does not allocate memory to hold the data
    - ✓int * ptr; // create a pointer-to-int: may be NULL, may not
    - ✓*ptr= 223323; // place a value in never-never land: disaster
- Program example init.cpp



Pointers and Numbers

- Similarities and differences between pointer and integer
  - They are both **integers** but pointers are not the integer **type**
  - Both are numbers you can add and subtract but it doesn't make sense to multiply and divide two locations
- Why we need addition and subtraction operations?
- Can't simply assign an integer to a pointer
- You can do like this:
  - 0xB8000000 is an address literal (hexadecimal)
  - `int * ptr = (int *) 0xB8000000;`

Danger!!!

Pointers and Numbers

- Size of a pointer: How many bytes used to store a pointer/address?
- The output of the following code?

```
int * ptr1 = NULL;
char * ptr2 = NULL;
double * ptr3 = NULL;
cout << sizeof(ptr1) << endl;
cout << sizeof(ptr2) << endl;
cout << sizeof(ptr3) << endl;
```

Allocate Memory
C Style

Allocating Memory with **malloc()**

- `void* malloc( size_t size );`

- What's `size_t`?
  - `size_t` is the unsigned integer type of the result of the `sizeof` operator
  - `size_t` can store the maximum size of a theoretically possible object of any type (including array).

Allocating Memory with **malloc()**

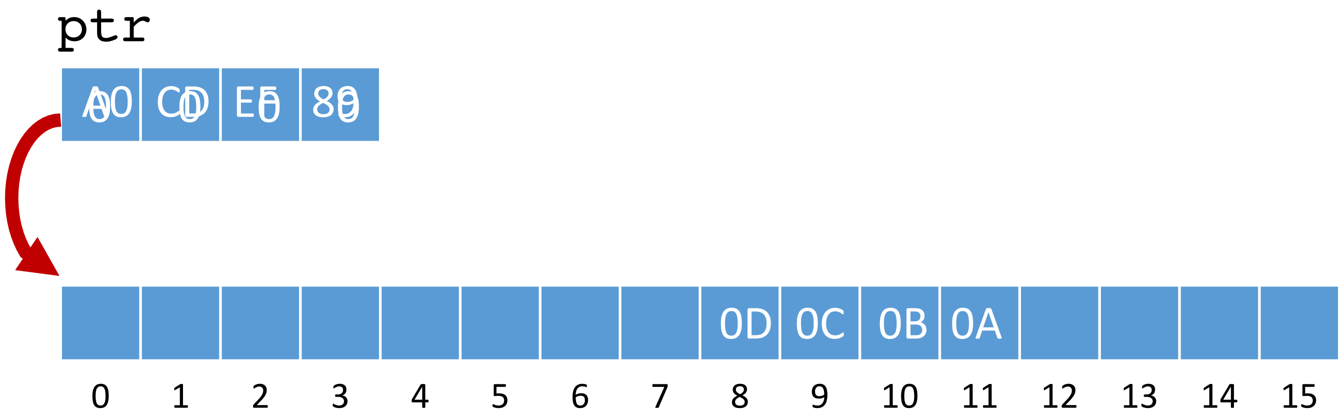
- DON'T forget to free the memeory!!!
- `void free( void* ptr );`
- The address of `ptr` will NOT be NULL(0) after you free the memory.
- Program example malloc.cpp

How it works

```
int * ptr = 0;

ptr = (int*) malloc(16);

ptr[2] = 0x0A0B0C0D;
```



The slides are based on the book <Stephen Prata, C++ Primer Plus, 6th Edition, Addison-Wesley Professional, 2011>

Allocate Memory C++ Style

Allocating Memory with **new**

- In C++, we use new
  ①Tell new for what data type you want memory
  ②Let new find a block of the correct size
  ③Return the address of the block
  ④Assign this address to a pointer
  ⑤This is an example:

```
int * ptr_int = new int;
*ptr_int  = 1;
```

- Program example use_new.cpp
  - Operation: sizeof

Freeing Memory with **delete**

- delete operator enables you to return memory to the memory pool
  - The memory can then be reused by other parts of the program
  - Balance the uses of new and delete
  - Memory leak—memory has been allocated but no longer being used
- Beware of
  - Cannot free a block of memory that you have previously freed
  - Cannot use delete to free memory created by ordinary variable

```
int * ps = new int; // allocate memory with new
. . .               // use the memory
delete ps;          // free memory with delete when done
```

```
int * ps = new int;   // ok
delete ps;            // ok
delete ps;            // not ok now
int jugs = 5;         // ok
int * pi = &jugs;     // ok
delete pi;            // not allowed, memory not allocated by new
```

Using **new** to Create Dynamic Arrays

- Use **new** with larger chunks of data, such as arrays, strings, and structures
  - ➢ Static binding: the array is built into the program at compile time
  - ➢ Dynamic binding: the array is created during runtime
    - ✓ The size of block can be confirmed during runtime

```
int * psome = new int [10]; // get a block of 10 ints
delete [] psome;                  // free a dynamic array
```

① Don't use delete to free memory that new didn't allocate

② Don't use delete to free the same block of memory twice in succession

③ Use delete [] if you used new [] to allocate an array

④ Use delete (no brackets) if you used new to allocate a single entity

⑤ It's safe to apply delete to the null pointer (nothing happens)

Using a Dynamic Array

- How do you use the dynamic array?
  - Identify every element in the block
  - Access one of these elements
  - You can increase a pointer with +1 (or ++, or +n)
- A pointer points to the first element

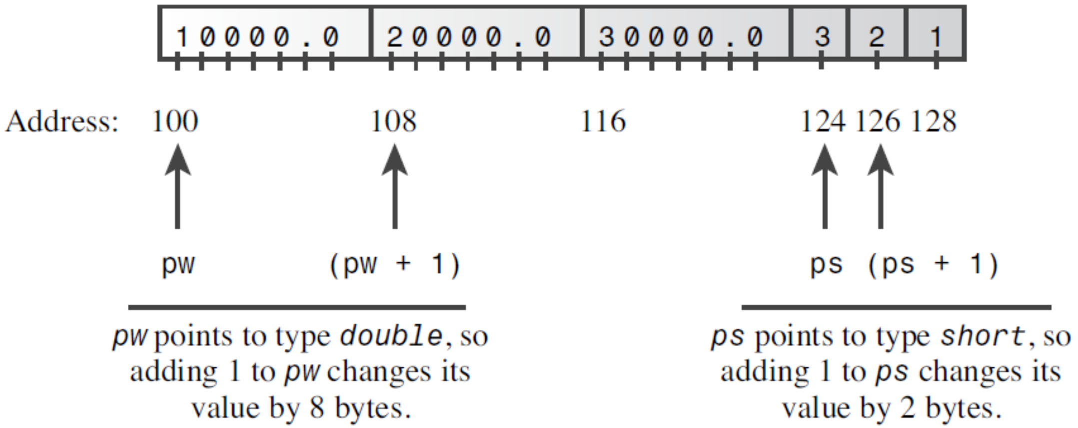
```
double * p3 = new double [3]; // space for 3 doubles
p3 = p3 + 1; // increment the pointer
p3 = p3 - 1; // point back to beginning
```

- Program example arraynew.cpp

Pointers, Arrays, and Pointer Arithmetic

- Adding **one** to a pointer variable increases its value by **the number of bytes of the type** to which it points
- Program example addpntrs.cpp
  - You can use pointer names and array names in **the same way**
  - **Differences** between them
    1. ① You can **change** the value of a **pointer**, whereas an **array name is a constant**
    2. ② Applying the sizeof operator to an array name yields the **size of the array**, but applying sizeof to a pointer yields the **size of the pointer**

```
double wages[3] = {10000.0, 20000.0, 30000.0};
short stacks[3] = {3, 2, 1};
double * pw = wages;
short * ps = &stacks[0];
```



Using **new** to Create Dynamic Structures

- Dynamic means the memory is allocated during runtime
  - Creating the structure
  - Accessing its members

```
inflatable * ps = new inflatable;
```

  - The arrow membership operator (->) of a hyphen and then a greater-than symbol
- Program example newstrct.cpp (single element)

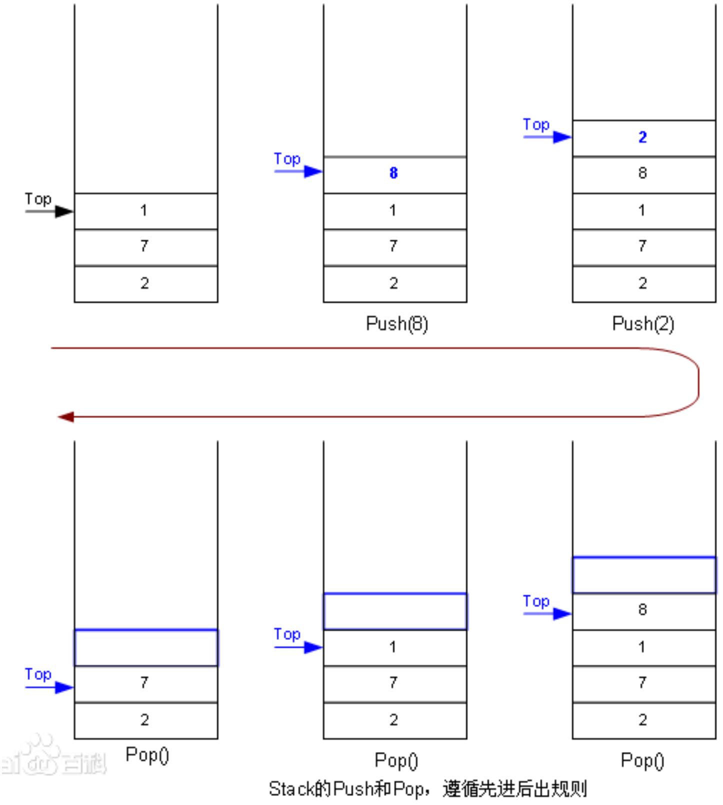
An Example of Using **new** and **delete** for Functions

- Program example delete.cpp
  - Return the address of the string copy
  - It's usually not a good idea to put new and delete in separate functions

Managing memory for data

Automatic Storage

- Automatic Storage
  - Ordinary variables defined inside a function use automatic storage and are called automatic variables
  - They expire when the function terminates
  - Automatic variables typically are stored on a stack
  - A last-in, first-out, or LIFO, process



Static Storage

- Static Storage
  - Static storage is storage that exists throughout the execution of an entire program
  - Two ways

    ① Define it externally, outside a function

    ② Use the keyword static when declaring a variable

```
static double fee = 56.50;
```

Dynamic Storage

- Dynamic Storage
  - The new and delete operators provide a more flexible approach than automatic and static variables
  - Refer to as the free store or heap
  - Lifetime of the data is not tied arbitrarily to the life of the program or the life of a function

Combinations of Types

- Combinations
  - Include arrays, structures, and pointers

- Program example mixtypes.cpp: array of structures
  - const event * arp[3] = {&s01, &s02, &s03};
  - const event ** ppa = arp;

Array Alternatives

- The vector Template Class
  - It is a **dynamic** array (Similar to the string class)
  - Use new and delete to manage memory
  - The vector identifier is part of the std namespace
- The array Template Class
  - The array identifier is part of the std namespace
  - The number of elements can't be a variable
  - **Static** memory allocation
- See Program Example choice.cpp
  - Comparing Arrays, Vector Objects, and Array Objects