南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

C/C++ Program Design

CS205

Prof. Shiqi Yu (于仕琪)

yusq@sustech.edu.cn

http://faculty.sustech.edu.cn/yusq/

南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Pointers

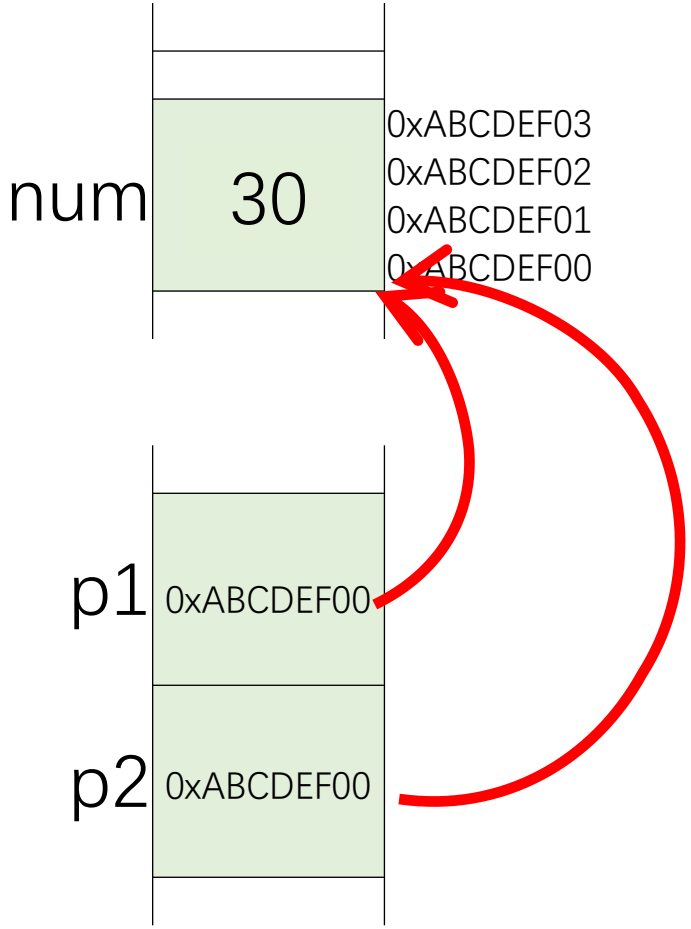
Pointers

- A pointer is declared like a variable, but with * after the type.
- What stored in a pointer variable is an address.
- Operator & can take the address of an object or a variable of fundamental types.
- Operator * can take the content that the pointer points to

```
int num = 10;
int * p1 = NULL, * p2 = NULL; // declaration two pointers, initialized to 0
p1 = &num; // take the address of num, assign to p1
p2 = &num; // take the address of num, assign to p2

*p1 = 20; // assign 20 to num
*p2 = 30; // assign 30 to num
```

How pointers work

```
int num = 10;
int * p1, * p2;
p1 = &num;
p2 = &num;
*p1 = 20;
*p2 = 30;
```

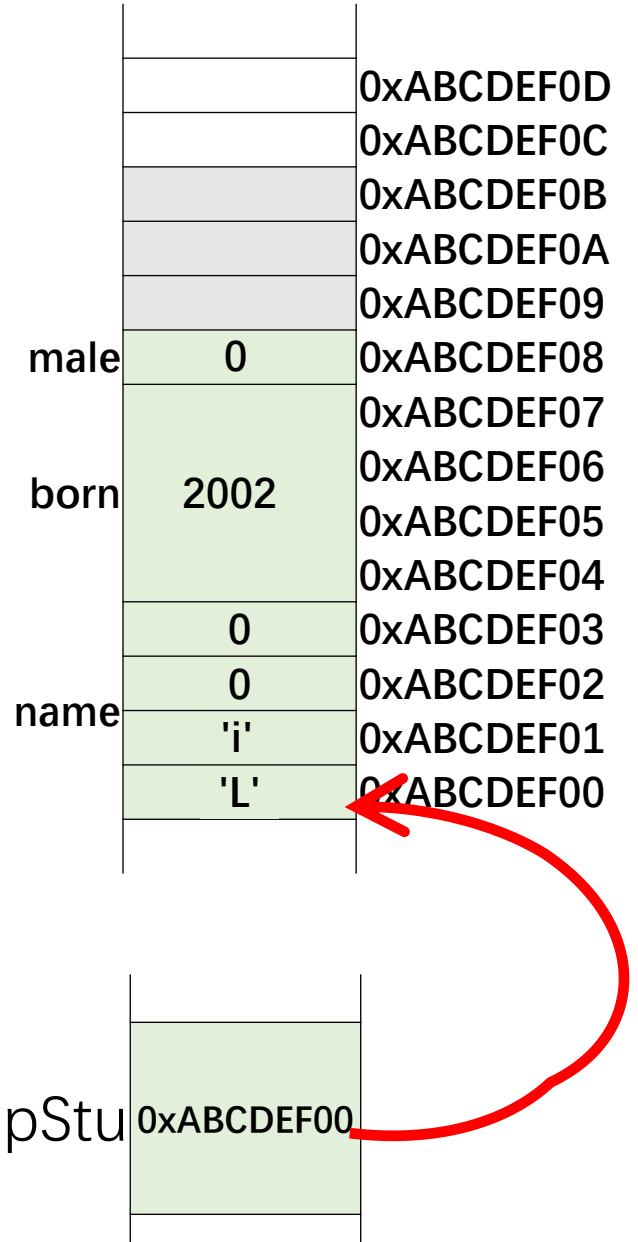


pointer.cpp

Structure member accessing

- `p->member`
- `(*p).member`

```
struct Student
{
    char name[4];
    int born;
    bool male;
};

Student stu = {"Yu", 2000, true};
Student * pStu = &stu;

strncpy(pStu->name, "Li", 4);
pStu->born = 2001;
(*pStu).born = 2002;
pStu->male = false;
```



pointer-struct.cpp

Print out the addresses

- Since the value of a pointer is an address, we can print it out

```
printf("Address of stu: %p\n", pStu); //C style
cout << "Address of stu: " << pStu << endl; //C++ style
cout << "Address of stu: " << &stu << endl;
cout << "Address of member name: " << &(pStu->name) << endl;
cout << "Address of member born: " << &(pStu->born) << endl;
cout << "Address of member male: " << &(pStu->male) << endl;
```

- The address should be an unsigned 32-bit or 64-bit integer.

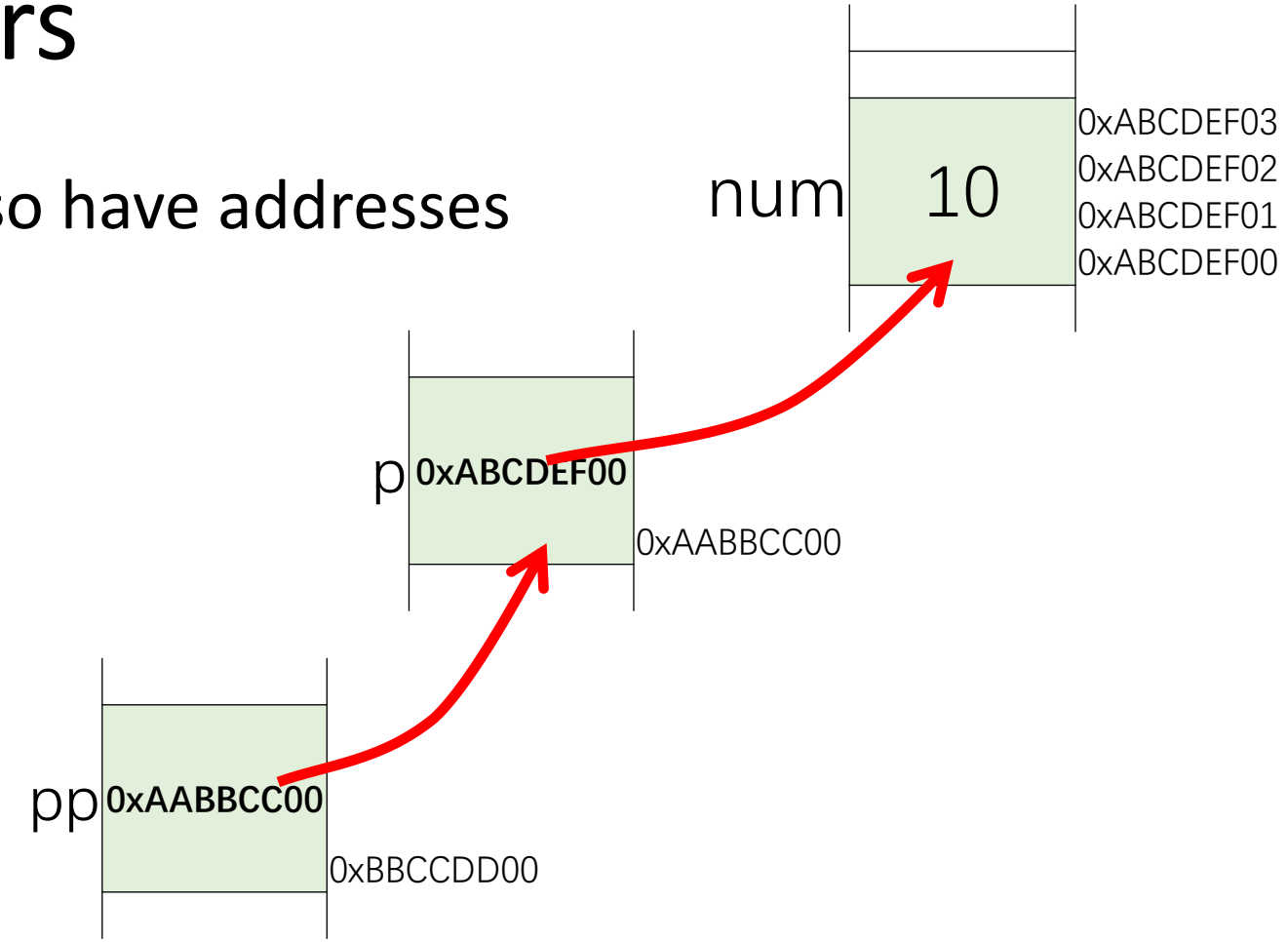
```
cout << "sizeof(pStu) = " << sizeof(pStu) << endl;
```

pointer-struct.cpp

Pointers of Pointers

- Pointers are variables, they also have addresses

```
int num = 10;
int * p = &num;
int ** pp = &p;
```



pointer-pointer.cpp

Constant pointers

```
int foo(const char * p)
{
   // the value that p points to cannot be changed

   // play a trick?
   char * p2 = p; //syntax error
   //...
   return 0;
}
```

```
int num = 1;
int another = 2;
//You cannot change the value the p1 points to through p1
const int * p1 = &num;
*p1 = 3; //error
num = 3; //okay

//You cannot change value of p2 (address)
int * const p2 = &num;
*p2 = 3; //okay
p2 = &another; //error

//You cannot change either of them
const int* const p3 = &num;
```

const-pointer.cpp

南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

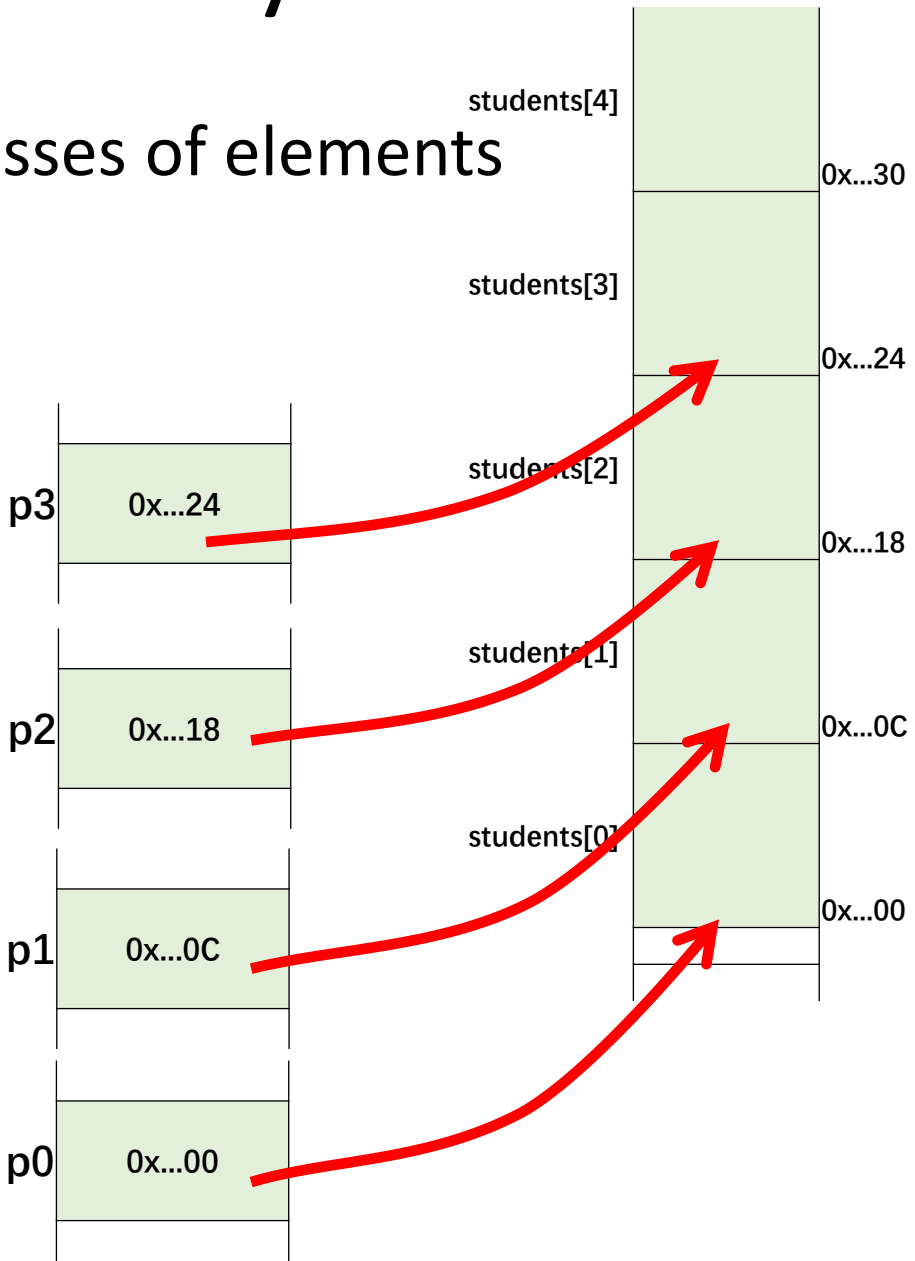
Pointers and Arrays

The addresses of array elements

- Use & operator to get the addresses of elements

```
Student students[128];
Student * p0 = &students[0];
Student * p1 = &students[1];
Student * p2 = &students[2];
Student * p3 = &students[3];

printf("p0 = %p\n", p0);
printf("p1 = %p\n", p1);
printf("p2 = %p\n", p2);
printf("p3 = %p\n", p3);

//the same behavior
students[1].born = 2000;
p1->born = 2000;
```

pointer-array.cpp



Array name

- You can consider an array name as a pointer

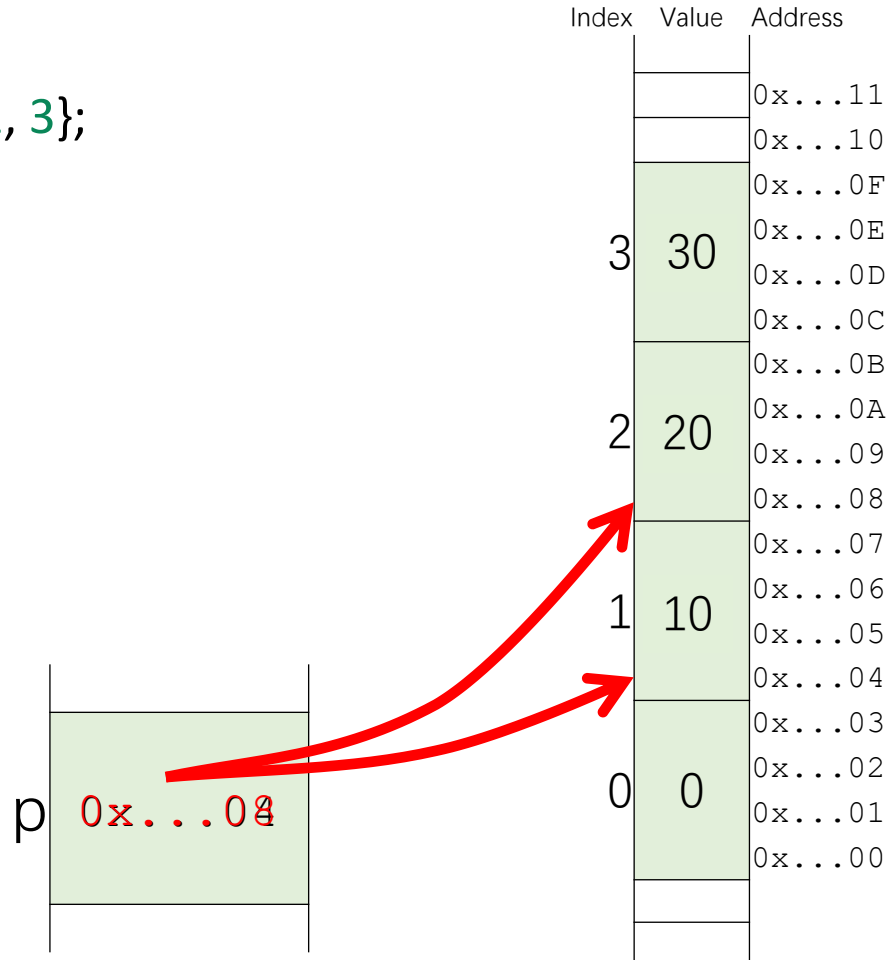
```
Student students[128];

printf("&students = %p\n", &students);
printf("students = %p\n", students);
printf("&students[0] = %p\n", &students[0]);

Student * p = students;
p[0].born = 2000;
p[1].born = 2001;
p[2].born = 2002;
```

pointer-array.cpp

Pointer arithmetic

- `p + num` or `num + p` points to the *num-th* element of the array `p`.
- `p - num` points to the *-num-th* element.

```
int numbers[4] = {0, 1, 2, 3};

int * p = numbers + 1;
p++;

*p = 20;
*(p-1) = 10;
p[1] = 30;
```



arithmetic.cpp

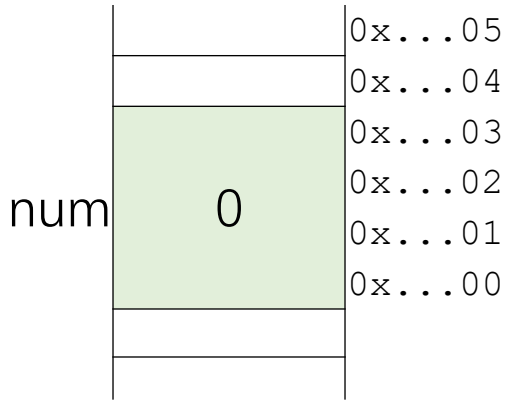
Pointer arithmetic

- The following are equivalent.

```
int i = ...;
int * p = ...;

p[i] = 3;

*(p + i) = 3;

int * p2 = p + i; *p2 = 3;
```

- Be careful of out-of-bound.

```
int num = 0;
int * p = &num;

p[-1] = 2; //out of bound
p[0] = 3; //okay
*(p+1) = 4; //out of bound
```



bound.cpp

Differences between a pointer and an array

- Array is a constant pointer.
- The total size of all elements in an array can be got by operator `sizeof`
- `sizeof` operator to a pointer will return the size of the address (4 or 8)

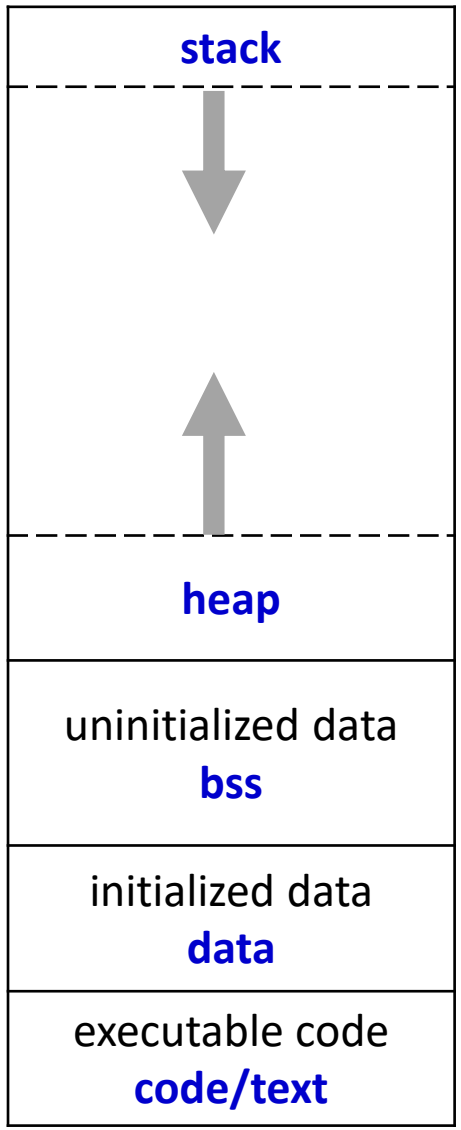
```
int numbers[4] = {0, 1, 2, 3};
int * p = numbers;
cout << sizeof(numbers) << endl; //4*sizeof(int)
cout << sizeof(p) << endl; // 4 or 8
cout << sizeof(double *) << endl; // 4 or 8
```

sizeof (int*)

南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Allocate memory: C style

Program memory



The address space of a program contains several data segments.

- Code: executable code
- Data: initialized static variables
- BSS: uninitialized static data including variables and constants
- Heap: dynamically allocated memory
- Stack: local variables, call stack

Program memory

```
int a = 0;
int b = 0;
int c = 0;
cout << &a << endl;
cout << &b << endl;
cout << &c << endl;

int * p1 = (int*) malloc (4);
int * p2 = (int*) malloc (4);
int * p3 = (int*) malloc (4);

cout << p1 << endl;
cout << p2 << endl;
cout << p3 << endl;
```

stack

heap

- But different CPU architectures may be different

arm64	X86_64
0x16cf7b648	0x3064676e8
0x16cf7b644	0x3064676e4
0x16cf7b640	0x3064676e0
0x145606790	0x7ff835c059c0
0x1456067a0	0x7ff835c059d0
0x1456067b0	0x7ff835c059e0

Memory allocation

- Allocate `size` bytes of uninitialized storage.

```
void* malloc( size_t size )
```

- Allocate 4 bytes and convert the pointer to (int *) explicitly.

```
int * p1 = (int*) malloc (4);
```

- Question:

```
int * p1 = (int*) malloc (3);
```

Memory deallocation

- The dynamically allocated memory must be deallocated explicitly!

  ```
  void free( void* ptr ) ;
  ```

- Question:

```
p = (int *) malloc(4 * sizeof(int));
// ...
p = (int *) malloc(8 * sizeof(int));
// ...
free (p);
```

内存泄漏

```
void foo()
{
   int* p = (int *) malloc( sizeof(int));
   return;
} //memory leak
```

Memory leak:

No variable to keep the first address. The memory management system will not deallocate it automatically. Waste of memory!

memoryleak.c

南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Allocate memory: C++ style

Operator `new` and `new[]`

- Operator `new` is similar with `malloc()` but with more features.

```
//allocate an int, default initializer (do nothing)
int * p1 = new int;
//allocate an int, initialized to 0
int * p2 = new int();
//allocate an int, initialized to 5
int * p3 = new int(5);
//allocate an int, initialized to 0
int * p4 = new int{};//C++11
//allocate an int, initialized to 5
int * p5 = new int {5};//C++11

//allocate a Student object, default initializer
Student * ps1 = new Student;
//allocate a Student object, initialize the members
Student * ps2 = new Student {"Yu", 2020, 1}; //C++11
```

Operator `new` and `new[]`

- Operator `new` is similar with `malloc()` but with more features.

```
//allocate 16 int, default initializer (do nothing)
int * pa1 = new int[16];
//allocate 16 int, zero initialized
int * pa2 = new int[16]();
//allocate 16 int, zero initialized
int * pa3 = new int[16]{}; //C++11
//allocate 16 int, the first 3 element are initialized to 1,2,3, the rest 0
int * pa4 = new int[16]{1,2,3}; //C++11

//allocate memory for 16 Student objects, default initializer
Student * psa1 = new Student[16];
//allocate memory for 16 Student objects, the first two are explicitly initialized
Student * psa2 = new Student[16]{{"Li", 2000,1}, {"Yu", 2001,1}}; //C++11
```

Operator `delete` and `delete[]`

- Destroys object/objects allocated by `new` and free memory

```
//deallocate memory
delete p1;
//deallocate memory
delete ps1;

//deallocate the memory of the array
delete pa1;
//deallocate the memory of the array
delete []pa2;

//deallocate the memory of the array, and call the destructor of
the first element
delete psa1;
//deallocate the memory of the array, and call the destructors
of all the elements
delete []psa2;
```