南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Advanced Programming

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

Arrays

Arrays

- A contiguously allocated memory
- Fixed number of objects (The array size cannot be changed)
- Its element type can be any fundamental type (int, float, bool, etc), structure, class, pointer, enumeration,

array.cpp

```
int num_array1[5]; //uninitialized array, random values
int num_array2[5] = {0, 1, 2, 3, 4}; //initialization
```

```
yushiqi: examples $ g++ array.cpp
yushiqi: examples $ ./a.out
0 0 0 0 169944840
0 1 2 3 4
yushiqi: examples $ ./a.out
0 0 0 0 163256072
0 1 2 3 4
yushiqi: examples $ ./a.out
0 0 0 0 142948104
0 1 2 3 4
```

Variable-length arrays

- If the length is not an integer constant expression, the array will be a variable-length one.

variable-array.cpp

```
int len = 1;
while ( len < 10 )
{
    int num_array2[len]; //variable-length array
    cout << "len = " << len;
    cout << ", sizeof(num_array2)) = "
         << sizeof(num_array2) << endl;
    len ++;
}
```

Arrays of unknown size

- The number is not specified in the declaration.

```
int num_array[ ] = {1, 2, 3, 4}; // the type of num_array is "array of 4 int"
```

- The arguments of a function

```
float array_sum(float values[], size_t length);

float array_sum(float *values, size_t length);
```

Element accessing

```
int array1[4] = {9,8,7,6};
int array2[4];
array2 = array1; //error!
array2[0] = array1[0]; //okay
array2[1] = array1[1]; //okay
array2[2] = array1[2]; //okay
array2[3] = array1[3]; //okay
```

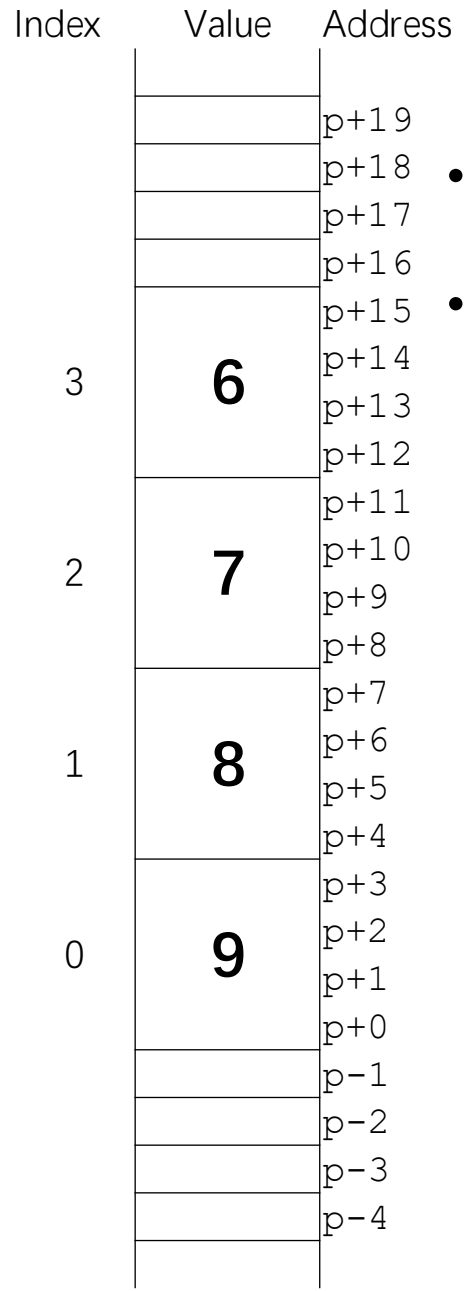
- No bounds-checking in C/C++.

index-bound.cpp

```
int num_array[5];

for(int idx = -1; idx <= 5; idx++)
   num_array[idx] = idx * idx;

for(int idx = -1; idx <= 5; idx++)
   cout << num_array[idx] << endl;
```



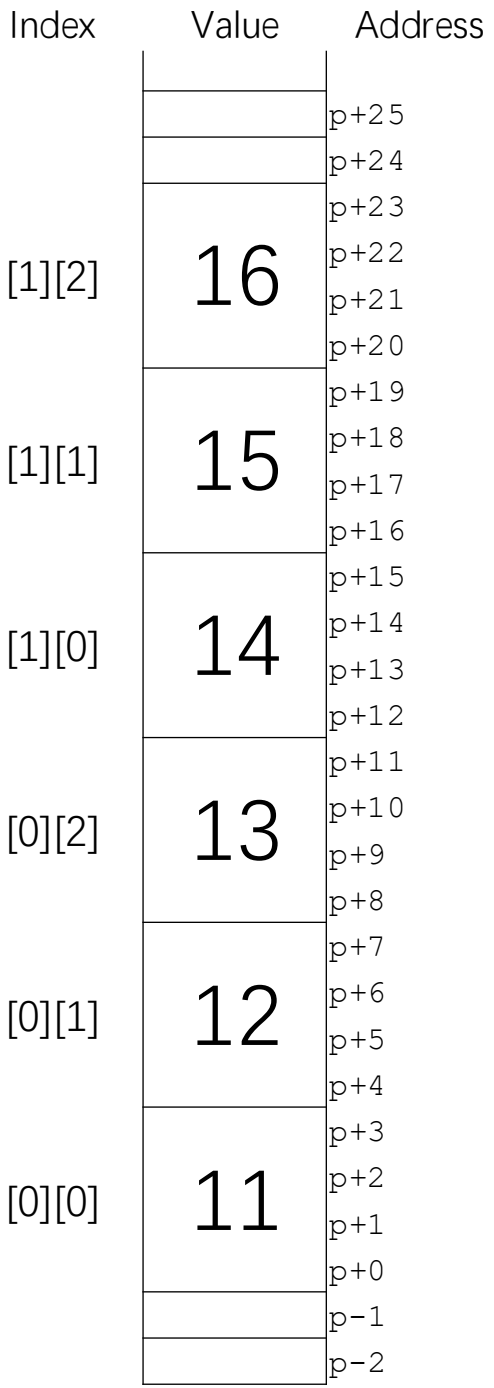
- **Arrays are not objects in C/C++ (different with Java);**
- **Arrays can be regarded as addresses**

Multidimensional arrays

```
int mat[2][3] = {{11,12,13}, {14,15,16}};

for (int r = 0; r < rows; r++)
{
   for(int c = 0; c < cols; c++)
      cout << mat[r][c] << ",";
   cout << endl;
}
```

md-array.cpp



Arrays of unknown bound

```
void init_2d_array(float mat[][], //error
          size_t rows, size_t cols)

void init_2d_array(float mat[][3],
          size_t rows, size_t cols)
```

const Arrays

```
const float PI = 3.1415926f;
PI += 1.f; // error
const float values[4] = {1.1f, 2.2f, 3.3f, 4.4f};
values[0] = 1.0f; // error
```

Used as function arguments

```
float array_sum(const float values[], size_t length)
{
    float sum = 0.0f;
    for (int i = 0; i < length; i++)
    {
        sum += values[i];
        //values[i] = 0; //error
    }
    return sum;
}
```

const-array.cpp

```
int main()
{
    float values[4] = {1.1f, 2.2f, 3.3f, 4.4f};
    float sum = array_sum(values, 4);
}
```

南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Strings

Array-Style Strings and `string` class

Array-style strings

- An array-style string (null-terminated strings/arrays of characters) is a series of characters stored in bytes in memory.
- This kind of strings can be declared as follows

initchar.cpp

```
char rabbit[16] = {'P', 'e', 't', 'e', 'r'};
char bad_pig[9] = {'P', 'e', 'p', 'p', 'a', ' ', 'P', 'i', 'g'}; //a bad one!
char good_pig[10] = {'P', 'e', 'p', 'p', 'a', ' ', 'P', 'i', 'g', '\0'};
```

- size_t strlen(const char *str);

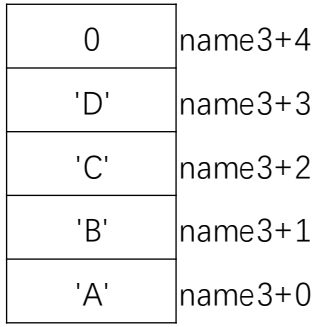
  Returns the number of characters, the first NULL will not be included.

```
char name[10] = {'Y', 'u', '\0', 'S', '.', '0'};
cout << strlen(name) << endl;
```

String literals

- It isn’t convenient to initial a string character by character.
- String literals can help.

```
char name1[] = "Southern University of Science and Technology";
char name2[] = "Southern University of "    "Science and Technology";
char name3[] = "ABCD"; //how many bytes for the array?
```




```
const wchar_t s1[] = L"ABCD";
const char8_t s2[] = u8"ABCD"; //since C++20
const char16_t s3[] = u"ABCD"; //since C++11
const char32_t s4[] = U"ABCD"; //since C++11
```

String manipulation and examination

- Copy

```
char* strcpy( char* dest, const char* src );
```

  - Safer one:

```
char *strncpy(char *dest, const char *src, size_t count);
```

- Concatenate: appends a copy of src to dest

```
char *strcat( char *dest, const char *src );
```

- Compare

```
int strcmp( const char *lhs, const char *rhs );
```

stringop.cpp

`string` class

- Null-terminated strings are easy to be out of bound, and to cause problems.
- `string` class provides functions to manipulate and examinate strings.

```
std::string str1 = "Hello";
std::string str2 = "SUSTech";
std::string result = str1 + ", " + str2;
```

stdstring.cpp

- Different types of strings

```
std::string
std::wstring
std::u8string //(C++20)
std::u16string //(C++11)
std::u32string //(C++11)
```

南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Structures, Unions and Enumerations

struct

- A `struct` is a type consisting of a sequence of members.
- The members are allocated in an ordered sequence.

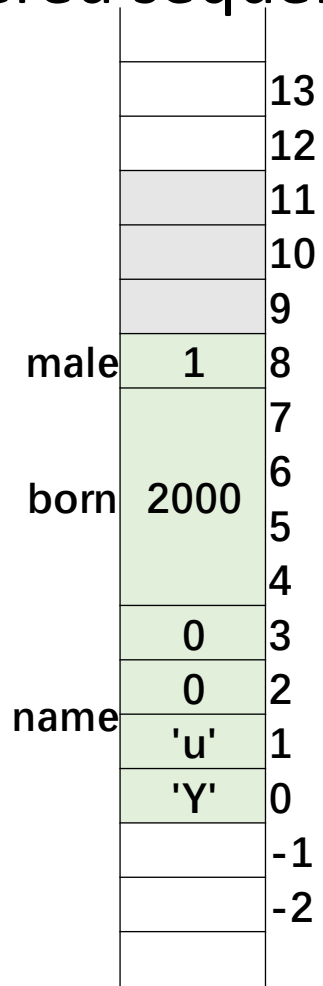
struct.c

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

struct Student stu;
strcpy(stu.name, "Yu");
stu.born = 2000;
stu.male = true;

struct Student stu = {"Yu", 2000, true};

struct Student students[100];
students[50].born = 2002;
```



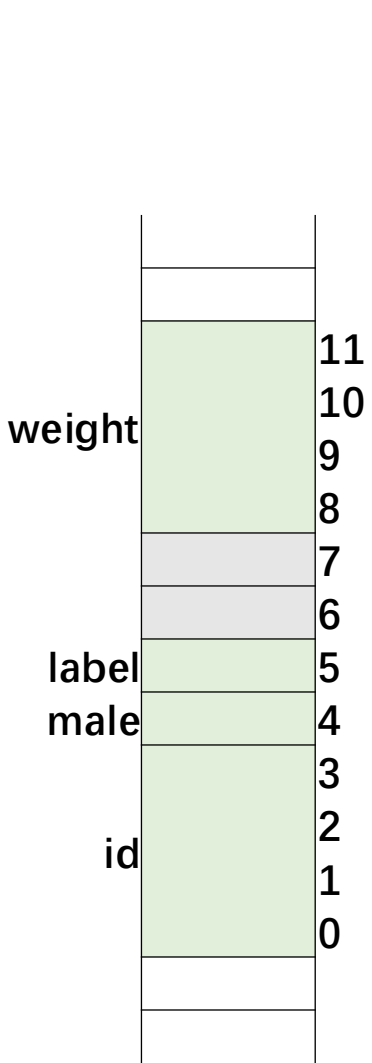
Structure padding

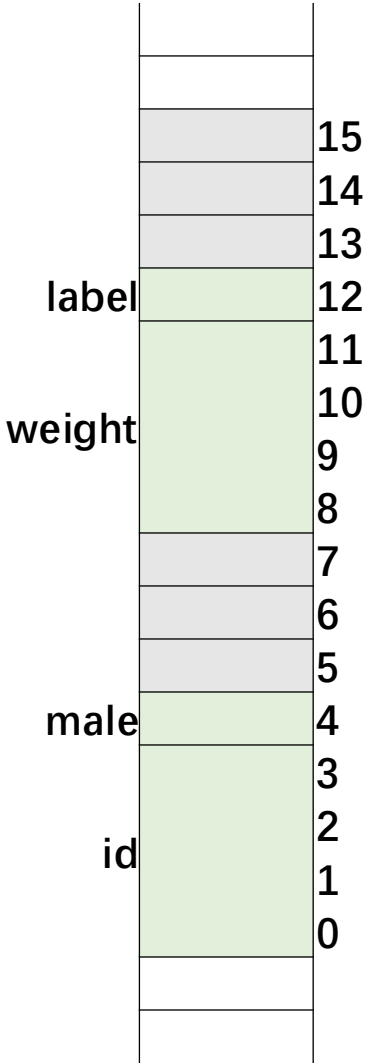
- In order to align the data in memory, some empty bytes will be padded

structpadding.cpp

```
struct Student1{
    int id;
    bool male;
    char label;
    float weight;
};

struct Student2{
    int id;
    bool male;
    float weight;
    char label;
};
```





`struct` in C and C++

- `struct` and `class` in C++ are identical except for several features.

```
struct Student1{
   int id;
   bool male;
   char label;
   float weight;
};
Student1 stu;
stu.id = 123;
```

- No `typedef` needed in C++!

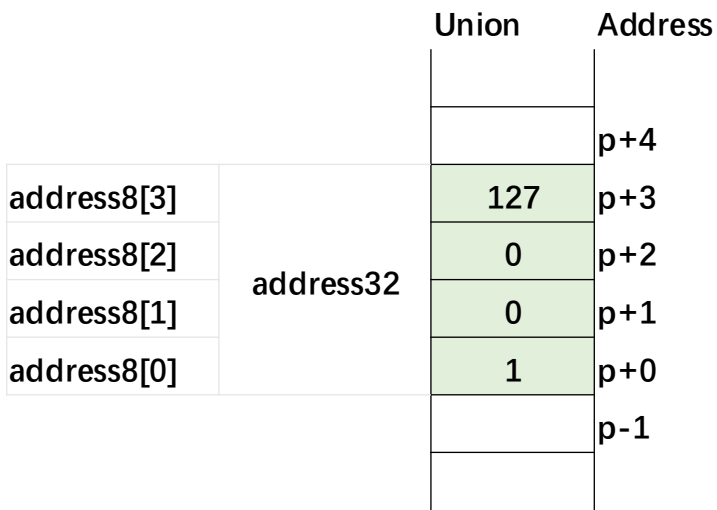
union

- `union` declaration is similar to `struct` declaration.
- The storage of members overlaps/shared.

```
union ipv4address{
   std::uint32_t address32;
   std::uint8_t address8[4];
};
```

- sizeof(union ipv4address) is 4.

union.cpp



enum

- `enum` makes a new type.
- It provides an alternative to `const` for creating symbolic constants.
- Its members are integers, but they cannot be operands in arithmetic expressions.

```
enum color {WHITE, BLACK, RED, GREEN, BLUE, YELLOW, NUM_COLORS};
enum color pen_color = RED;
pen_color = color(3);

cout << "We have " << NUM_COLORS << " pens." << endl;

pen_color += 1; //error!
int color_index = pen_color;
color_index += 1;
cout << "color_index = " << color_index << endl;
```

An example with struct, union and enum

```
enum datatype {TYPE_INT8=1, TYPE_INT16=2, TYPE_INT32=4, TYPE_INT64=8};
```

```
struct Point{
   enum datatype type;
   union {
      std::int8_t data8[3];
      std::int16_t data16[3];
      std::int32_t data32[3];
      std::int64_t data64[3];
   };
};
size_t datawidth(struct Point pt)
{
   return size_t(pt.type) * 3;
}
```

```
int64_t l1norm(struct Point pt)
{
 int64_t result = 0;
 switch(pt.type)
 {
  case (TYPE_INT8):
   result = abs(pt.data8[0]) +
            abs(pt.data8[1]) +
            abs(pt.data8[2]);
    break;
  ...
```

enum.cpp

南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

typedef

typedef

- typedef can create an alias for a type.
- It can be used to replace a possibly complex type name.

typedef.cpp

```
typedef int myint;

typedef unsigned char vec3b[3];

typedef struct _rgb_struct{//name _rgb_struct can be omit
   unsigned char r;
   unsigned char g;
   unsigned char b;
} rgb_struct;

myint num = 32;

unsigned char color[3];
vec3b color = {255, 0, 255};

rgb_struct rgb = {0, 255, 128};
```

Typical `typedef` usages

_uint8_t.h

```
#ifndef _UINT8_T
#define _UINT8_T
typedef unsigned char uint8_t;
#endif /* _UINT8_T */
```

```
#if defined(_LP64)
typedef int wchar_t;
#else
typedef long wchar_t;
#endif
```