C/C++ Program Design

LAB 3

CONTENTS

- Master the use of arrays
- Master character arrays and strings
- Master the use of Structure
- Learn about Union

2 Knowledge Points

2.1 Array

2.2 Character arrays and strings

2.3 Structure

2.4 Union

2.1 Array

- Arrays are fixed-size collections consisting of data items of the same type.
- The array name represents the first address of the contiguous storage space.
- The index of an array is from 0.
- C/C++ compiler does not report whether the array is out-off-bounds.

One-dimension array

lab03_examples > onedarray.cpp > ...

```cpp
#include <iostream>
using namespace std;

int main()
{
    int foo[] = {16, 2, 77, 40, 12071};
    int a = 1;

    foo[0] = a;
    foo[1] = -34;
    a = foo[2];

    cout << "foo[0] = " << foo[0] << endl;
    cout << "foo[1] = " << foo[1] << endl;
    cout << "foo[2] = " << foo[2] << endl;
    cout << "a = " << a << endl;

    cout << "The size of foo is:" << sizeof(foo) << " bytes" << endl;
    cout << "The size of an element of foo is:" << sizeof(foo[0]) << " bytes" << endl;
    cout << "There are " << sizeof(foo)/sizeof(foo[0]) << " elements in foo." << endl;

    return 0;
}
```

Define and initialize a one-dimension array

Use [ ] operator to access the elements of the array

The array index starts from 0

Use the **sizeof** operator with an **array name**, you'll get the amount of bytes of an array.

Use **sizeof** with an **array element**, you'll get the size, in bytes, of the element.

Count the number of elements in an array

```
foo[0] = 1
foo[1] = -34
foo[2] = 77
a = 77
The size of foo is:20 bytes
The size of an element of foo is:4 bytes
There are 5 elements in foo.
```

lab03_examples > arrayini.cpp > ...

```cpp
#include <iostream>
#include<iomanip>
#define SIZE 2
const int arrsize = 3;

int main()
{
    using std::cout;
    using std::endl;
    int a[SIZE] = {0};
    double b[arrsize] = {1};

//    cout << std::fixed;
    cout << "The elements in a are:" << a[0] << "," << a[1] << endl;
    cout << "The elements in b are:" << b[0] << "," << b[1] << "," << b[2] << endl;

    return 0;

}
```

Use preprocessor **#define** to define a symbolic constant

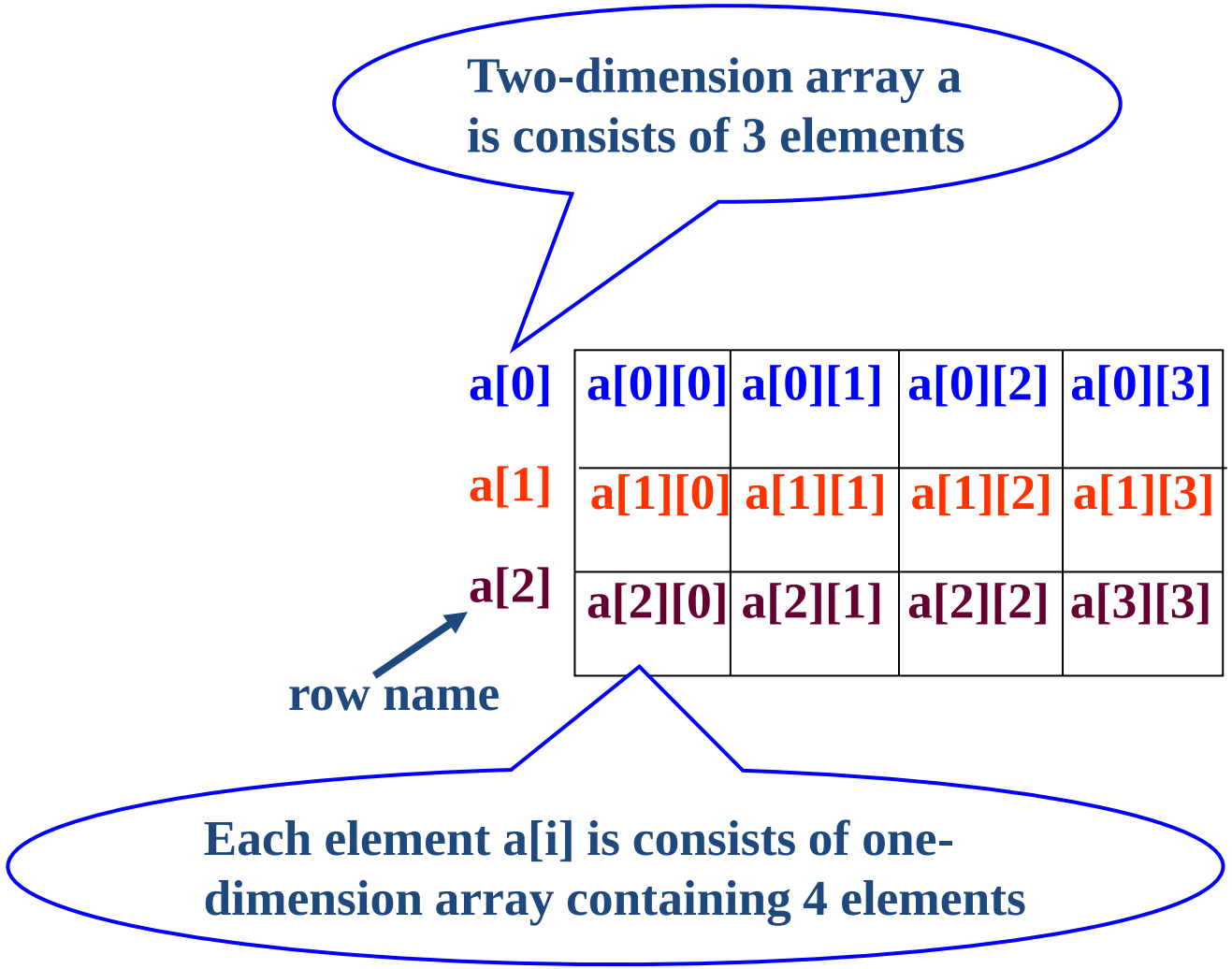
Use C++ style to define a constant

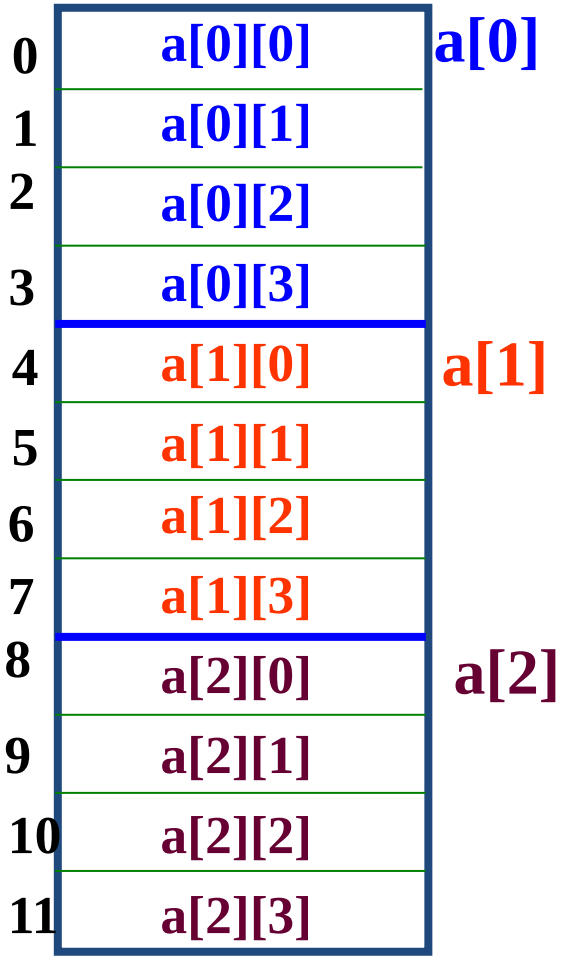
Initialize an array partially, the remaining elements are 0. The number of elements must be specified.

```
The elements in a are:0,0
The elements in b are:1,0,0
```

Two-dimension array

int a[3][4];





lab03_examples > twodarray.cpp > main()

```cpp
#include <iostream>
using namespace std;

int main()
{
    int test[3][2] =
    {
        {2, -5},
        {4, 0},
        {9, 1}
    };
    //Accessing two dimensional array
    cout << "test[0][1] = " << test[0][1] << endl;
    cout << "test[2][0] = " << test[2][0] << endl;

    cout << "The size of test is:" << sizeof(test) << endl;
    cout << "The size of the first row of test is:" << sizeof(test[0]) << endl;
    cout << "The size of an element in test is:" << sizeof(test[0][0]) << endl;

    return 0;

}
```

Define and initialize a two-dimension array

Use [ ] [ ]operator to access the elements of the array

```
test[0][1] = -5
test[2][0] = 9
The size of test is:24
The size of the first row of test is:8
The size of an element in test is:4
```

2.2 Character array and strings

2.2.1 Define a C-style string

You can use one of the four ways below to define a character array:

char str[ ] = “C++” ;
char str[4] = “C++” ; ← **Strings end with \0**
char str[ ] = {‘C’, ‘+’, ‘+’, ‘\0’};
char str[4] = {‘C’, ‘+’, ‘+’, ‘\0’}

```
char str1[] = "C++";
char str2[] = {'C','+','+'};

cout << "The sizeof str1 is " << sizeof(str1) << ",the length of str1 is " << strlen(str1) << endl;
cout << "The sizeof str2 is " << sizeof(str2) << ",the length of str2 is " << strlen(str2) << endl;
```

```
The sizeof str1 is 4,the length of str1 is 3
The sizeof str2 is 3,the length of str2 is 6
```

2.2.2 string class

string is a class of C++, it can be used as a type.

string str1 = “C and C++”; //initialize str1 with a string

string str2 = str1 + “programming is very interesting.”; //use + to concatenate two or more strings

```
string str3 = "C and C++";
string str4 = str3 + "programming is very interesting.";

cout << "The sizeof str3 is " << sizeof(str3) << ",the length of str3 is " << str3.length() << endl;
cout << "The sizeof str4 is " << sizeof(str4) << ",the length of str4 is " << str4.size() << endl;
```

```
The sizeof str3 is 32,the length of str3 is 9
The sizeof str4 is 32,the length of str4 is 41
```

2.2.3 Keyboard input and terminal output of character array

1. C: scanf & printf
   - %d ----int
   - %f ----float
   - %c -----char
   - %s -----string

lab03_examples > C scanf_printf.c > ...

```c
#include <stdio.h>

int main()
{
    char str[20];
    printf("Enter a string:\n");
    scanf("%s", str);
    printf("You entered: %s\n",str);

    return 0;

}
```

There is no &

Why only Computer?

```
maydlee@LAPTOP-U1MO0N2F:/mnt/d/mycode/CcodeVS/lab03_examples$ gcc scanf_printf.c
maydlee@LAPTOP-U1MO0N2F:/mnt/d/mycode/CcodeVS/lab03_examples$ ./a.out
Enter a string:
Computer
You entered: Computer
maydlee@LAPTOP-U1MO0N2F:/mnt/d/mycode/CcodeVS/lab03_examples$ ./a.out
Enter a string:
Computer Science
You entered: Computer
```

scanf uses whitespace—spaces, tabs, and newlines to delineate a string.

2. C: gets & puts

```
fgets(str, 20, stdin);
```

gets_puts.c

```c
#include <stdio.h>

int main()
{
    char str[20];
    printf("Enter a string:\n");
    gets(str);
    printf("You entered: ");
    puts(str);

    return 0;
}
```

Use gets function to gain the whole line

```
maydlee@LAPTOP-U1MO0N2F:/mnt/d/mycode/CcodeVS/lab03_examples$ gcc gets_puts.c
maydlee@LAPTOP-U1MO0N2F:/mnt/d/mycode/CcodeVS/lab03_examples$ ./a.out
Enter a string:
Computer Science
You entered: Computer Science
```

scanf()
when *scanf()* is used to read string input it stops reading when it encounters **whitespace, newline** or **End Of File**
It is used to read input of **any datatype**

gets()
when *gets()* is used to read input it stops reading input when it encounters **newline** or **End Of File.** It does not stop reading the input on encountering whitespace as it considers whitespace as a string.
It is used **only for string** input.

3. C++: cin & cout

lab03_examples > cin_cout.cpp > ...

```cpp
#include <iostream>
using namespace std;

int main()
{
    char str[100];

    cout << "Enter a string:";
    cin >> str;
    cout << "You entered: " << str << endl;

    cout << "Enter an other string:";
    cin >> str;
    cout << "You entered: " << str << endl;

    return 0;
}
```

```
Enter a string:C++
You entered: C++
Enter an other string:programming is fun.
You entered: programming
```

```
Enter a string:C++ plus
You entered: C++
Enter an other string:You entered: plus
```

The cin is to use whitespace-- spaces, tabs, and newlines to delineate a string.

4. C++: cin.get( )

Input a single character:
istream& get(char&);
int get(void);

Input a string:
istream& get(char*,int);

lab03_examples > cin_get.cpp > ...

```cpp
#include <iostream>
using namespace std;

int main()
{
    char str[20];

    cout << "Enter a string:";
    cin.get(str, 20);
    cout << "You entered: " << str << endl;

    cin.get();
    cout << "Enter an other string:";
    cin.get(str, 20);
    cout << "You entered: " << str << endl;

    return 0;
}
```

If the statement is omitted, what will be the output?

```
maydlee@LAPTOP-U1MO0N2F:/mnt/d/mycode/CcodeVS/lab03_examples$ g++ cin_get.cpp
maydlee@LAPTOP-U1MO0N2F:/mnt/d/mycode/CcodeVS/lab03_examples$ ./a.out
Enter a string:Programming is fun.
You entered: Programming is fun.
Enter an other string:C/C++ programming is fun.
You entered: C/C++ programming i
```

If the length of input string is greater than 20, it can only store first 19 characters in it.

5. C++: cin.getline( )

Input a string:

istream& getline(char*,int);

lab03_examples > cin_getline.cpp > ...

```cpp
#include <iostream>
using namespace std;

int main()
{
    char str[20];

    cout << "Enter a string:";
    cin.getline(str, 20);
    cout << "You entered: " << str << endl;

    cout << "Enter an other string:";
    cin.getline(str, 20);
    cout << "You entered: " << str << endl;

    return 0;
}
```

```
maydlee@LAPTOP-U1MO0N2F:/mnt/d/mycode/CcodeVS/lab03_examples$ g++ cin_getline.cpp
maydlee@LAPTOP-U1MO0N2F:/mnt/d/mycode/CcodeVS/lab03_examples$ ./a.out
Enter a string:Programming is fun.
You entered: Programming is fun.
Enter an other string:C/C++ programming is fun.
You entered: C/C++ programming i
```

If the length of input string is greater than 20, it can only store first 19 characters in it.

cin.get( ) vs cin.getline( )

getline() and get() both read an entire input line—that is, up until a newline character. However, getline() discard the newline character, whereas get() leave it in the input queue.

lab03_examples > get_getline.cpp > ...

```cpp
#include <iostream>
using namespace std;

int main()
{
    char str[20];

    cout << "Enter a string:";
    cin.get(str, 20);
    cout << "You entered: " << str << endl;

    cout << "Enter an other string:";
    cin.getline(str, 20);
    cout << "You entered: " << str << endl;

    return 0;
}
```

Program runs without entering another string

```
maydlee@LAPTOP-U1MO0N2F:/mnt/d/mycode/CcodeVS/lab03_examples$ g++ get_getline.cpp
maydlee@LAPTOP-U1MO0N2F:/mnt/d/mycode/CcodeVS/lab03_examples$ ./a.out
Enter a string:C and C++
You entered: C and C++
Enter an other string:You entered:
```

lab03_examples > get_getline.cpp > main()

```cpp
#include <iostream>
using namespace std;

int main()
{
    char str[20];

    cout << "Enter a string:";
    cin.get(str, 20);
    cout << "You entered: " << str << endl;

    cin.get();
    cout << "Enter an other string:";
    cin.getline(str, 20);
    cout << "You entered: " << str << endl;

    return 0;
}
```

```
maydlee@LAPTOP-U1MO0N2F:/mnt/d/mycode/CcodeVS/lab03_examples$ g++ get_getline.cpp
maydlee@LAPTOP-U1MO0N2F:/mnt/d/mycode/CcodeVS/lab03_examples$ ./a.out
Enter a string:C and C++
You entered: C and C++
Enter an other string:Programming is fun.
You entered: Programming is fun.
```

6. string class I/O

getline() function takes the input stream as the first parameter which is cin and str as the location of the line to be stored.

lab03_examples > string.cpp > ...

```cpp
#include <iostream>
using namespace std;

int main()
{
    string str;
    cout << "Enter a string:";
    getline(cin, str);
    cout << "You entered: " << str << endl;

    cout << "Enter another string:";
    getline(cin,str);
    cout << "You entered: " << str << endl;

    return 0;
}
```

```
maydlee@LAPTOP-U1MO0N2F:/mnt/d/mycode/CcodeVS/lab03_examples$ g++ string.cpp
maydlee@LAPTOP-U1MO0N2F:/mnt/d/mycode/CcodeVS/lab03_examples$ ./a.out
Enter a string:Programming is fun.
You entered: Programming is fun.
Enter another string:C/C++ programming is fun.
You entered: C/C++ programming is fun.
```

2.3 Structure

A **structure** is a user defined data type available in C that allows to combine **data items of different kinds** under a single name. **When a structure is declared, no memory is allocated.**

```
struct Employee
{
    int id;
    string name;
};

struct Person
{
    char name[7];
    int age;
    double salary;
};

int main()
{
    Employee employee1;
    Person person1;

    cout << "The size of Employee is:" << sizeof(Employee) << endl;
    cout << "The size of Person is:" << sizeof(person1) << endl;

    return 0;
}
```

Normally, the size of a structure is not the sum of the sizes of its members because of memory alignment.

```
The size of Employee is:40
The size of Person is:24
```

2.4 Union

A **union** is a special data type available in C that allows to **store different data types in the same memory location**. You can define a union with many members, but **only one member can contain a value at any given time**. The memory occupied by a union will be large enough to hold the largest member of the union. Unions provide an efficient way of using the same memory location for multiple-purpose.

A union is big enough to hold the '' widest'' member, and the alignment is appropriate for all of the types in the union.

```
sizeof(data) is 20
data.i : 1917853763
data.f : 4122360580327794860452759994368.000000
data.str : C Programming
```

Only the value is valid.

```c
#include <stdio.h>
#include <string.h>

union Data
{
   int i;
   float f;
   char str[18];
};

int main( )
{

   union Data data;

   data.i = 10;
   data.f = 220.5;
   strcpy( data.str, "C Programming");

   printf("sizeof(data) is %lu\n",sizeof(data));
   printf( "data.i : %d\n", data.i);
   printf( "data.f : %f\n", data.f);
   printf( "data.str : %s\n", data.str);

   return 0;
}
```

Big-Endian and Little-Endian

BE stores the big-end first, the lowest memory address is the biggest.
LE stores the little-end first, the lowest memory address is the littlest.

Big-Endian

2000	11
2001	22
2002	33
2003	44

Little-Endian

2000	44
2001	33
2002	22
2003	11

```
#include<stdio.h>
union data
{
   int a;
   char c;
};

int main()
{
   union data endian;
   endian.a = 0x11223344;

   if(endian.c == 0x11)
      printf("Big-Endian\n");
   else if(endian.c == 0x44)
      printf("Little-Endian\n");

   return 0;
}
```

Exercise 1

```
#include <iostream>
#include <string.h>
using namespace std;

int main()
{
   int cards[4]{};
   int price[] = {2.8,3.7,5,9,-1};
   char direction[4] {'L',82,'U',68};
   char title[] = "ChartGPT is an awesome tool.";

   cout << "sizeof(cards) = " << sizeof(cards) << ",sizeof of cards[0] = " << sizeof(cards[0]) << endl;
   cout << "sizeof(price) = " << sizeof(price) << ",sizeof of price[0] = " << sizeof(price[1]) << endl;
   cout << "sizeof(direction) = " << sizeof(direction) << ",length of direction = " << strlen(direction) << endl;
   cout << "sizeof(title) = " << sizeof(title) << ",length of title = " << strlen(title) << endl;

   //Print the address of each array variable.
   …….

   return 0;
}
```

First, complete the code, then run the program and explain the result to SA. If it has bugs, fix them.

Exercise 2

```
#include <stdio.h>

union data
{
   int n;
   char ch;
   short m;
};

int main()
{
   union data a;
   printf("%lu, %lu\n", sizeof(a), sizeof(union data) );
   a.n = 0x40;
   printf("%X, %c, %hX\n", a.n, a.ch, a.m);
   a.ch = '9';
   printf("%X, %c, %hX\n", a.n, a.ch, a.m);
   a.m = 0x2059;
   printf("%X, %c, %hX\n", a.n, a.ch, a.m);
   a.n = 0x3E25AD54;
   printf("%X, %c, %hX\n", a.n, a.ch, a.m);

   return 0;
}
```

Run the program and explain the result to SA.

Exercise 3

The **CandyBar** structure contains three members: name(character array), weight(float) and the number of calories(integer). Write a program that creates an array of **three CandyBar** structures, initializes them to value of your input, and then displays the contents of each structure. Find the greatest calories per weight, display the name and calories per weight of which satisfies the condition.

```
struct CandyBar
{
    char name[20];
    float weight;
    int calories;

};
```

Sample output:

```
Please input three CandyBars' information:
Enter brand name of a Candybar: Ferro Rocher
Enter weight of the Candybar: 23.6
Enter calories (an integer value) in the Candybar: 893
Enter brand name of a Candybar: Hershey's
Enter weight of the Candybar: 13.2
Enter calories (an integer value) in the Candybar: 658
Enter brand name of a Candybar: Mars Wrigley
Enter weight of the Candybar: 3.2
Enter calories (an integer value) in the Candybar: 127
-------------------------------------------------
Display the CandyBar array contents
Brand name: Ferro Rocher
Weight: 23.6
Calories: 893
Brand name: Hershey's
Weight: 13.2
Calories: 658
Brand name: Mars Wrigley
Weight: 3.2
Calories: 127
-------------------------------------------------
The greatest calories per weight is:
Brand name: Hershey's
Calories per weight: 49.8485
```