南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

C/C++ Program Design

Lab 7, shared library

廖琪梅，王大兴

Building a shared library

- Suppose we have written the following code:

```h
// mymath.h
#ifndef __MY_MATH_H__
#define __MY_MATH_H__
float arraySum(const float *array,
size_t size);
#endif
```

```cpp
// mymath.cpp
#include <iostream>
#include "mymath.h"

float arraySum(const float *array, size_t
size)
{
    if(array == NULL)
    {
        std::cerr << "NULL pointer!" <<
std::endl;
        return 0.0f;
    }
    float sum = 0.0f;
    for(size_t i = 0; i < size; i++)
        sum += array[i];
    return sum;
}
```


```cpp
// main.cpp
#include <iostream>
#include "mymath.h"
int main()
{
    float arr1[8]{1.f, 2.f, 3.f, 4.f, 5.f, 6.f, 7.f, 8.f};
    float * arr2 = NULL;

    float sum1 = arraySum(arr1, 8);
    float sum2 = arraySum(arr2, 8);

    std::cout << "The result1 is " << sum1 << std::endl;
    std::cout << "The result2 is " << sum2 << std::endl;

    return 0;
}
```

Building shared libraries

- A **shared library** packs compiled code of functionality that the developer wants to **share** with other developers.
- Shared libraries in linux are **.so** files.
- Remember to use arguments "**-shared**" and "**-fPIC**" when building it.
- Now we should see "libmymath.so" in the directory

The name of .so must be started with "**lib**" followed by the .cpp name in which a function is defined.

```
maydlee@LAPTOP-U1MO0N2F:/mnt/d/mycode/CcodeVS/sharedlib$ g++ -shared -fPIC -o libmymath.so mymath.cpp
maydlee@LAPTOP-U1MO0N2F:/mnt/d/mycode/CcodeVS/sharedlib$ ls
libmymath.so  main.cpp  mymath.cpp  mymath.h
```

Create a shared library.

Generate Position-Independent-Code.

Using shared library

- Now we can use the ".so" shared library.
- Let's compile "main":

"lmymath" indicates to use "libmymath.so"

```
maydlee@LAPTOP-U1MO0N2F:/mnt/d/mycode/CcodeVS/sharedlib$ g++ -o main main.cpp -L. -lmymath
maydlee@LAPTOP-U1MO0N2F:/mnt/d/mycode/CcodeVS/sharedlib$ ls
libmymath.so  main  main.cpp  mymath.cpp  mymath.h
```

"-L." indicates to find a library file in the current directory.

Using shared library

- After the “main” has been compiled, try to run it:

```
maydlee@LAPTOP-U1MO0N2F:/mnt/d/mycode/CcodeVS/sharedlib$ ls
libmymath.so  main  main.cpp  mymath.cpp  mymath.h
maydlee@LAPTOP-U1MO0N2F:/mnt/d/mycode/CcodeVS/sharedlib$ ./main
./main: error while loading shared libraries: libmymath.so: cannot open shared object file: No such file or directory
```

- It failed because “main” now relys on “libmymath.so”. By default, libraries are located in **/usr/local/lib** or **/usr/lib** , but our “libmymath.so” is not in that directory. You must tell the terminal where to find “libmymath.so”.

Using a shared library

- Using **export** command to set environment variable "**LD_LIBRARY_PATH**"
- And then run "main" again

```
maydlee@LAPTOP-U1MO0N2F:/mnt/d/mycode/CcodeVS/sharedlib$ export LD_LIBRARY_PATH=.:$LD_LIBRARY_PATH
maydlee@LAPTOP-U1MO0N2F:/mnt/d/mycode/CcodeVS/sharedlib$ echo $LD_LIBRARY_PATH
.:
maydlee@LAPTOP-U1MO0N2F:/mnt/d/mycode/CcodeVS/sharedlib$ ./main
NULL pointer!
The result1 is 36
The result2 is 0
```

export LD_LIBRARY_PATH=.:$LD_LIBRARY_PATH

There is no space on either side of the equal sign

. indicates the current directory

Another choice is to move your .so file to **/usr/lib** folder by **mv** command

```
maydlee@LAPTOP-U1MO0N2F:/mnt/d/mycode/CcodeVS/sharedlib$ sudo mv libmymath.so /usr/lib
[sudo] password for maydlee:
maydlee@LAPTOP-U1MO0N2F:/mnt/d/mycode/CcodeVS/sharedlib$ ./main
NULL pointer!
The result1 is 36
The result2 is 0
```

Exercise 1

Define a function whose prototype is **const char* match(const char* s, char ch);**
s is a C-style string, **ch** is a character. If the ch is in the s, return the position of s at ch; if the ch is not in the s, return NULL.
Write a test program to call the function and show the result. The output samples are as follows:

You are required to compile the function into a shared library “libmatch.so”, and then compile and run your program with this shared library.

```
Please input a string:
Enjoy the holiday.
Please input a character:
h
he holiday.
```

```
Please input a string:
Class is over.
Please input a character:
m
Not Found
```

Exercise 2

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

void displaySquare(int side = 4, char filledChar = '*');

int main()
{
   displaySquare();
   displaySquare(10,'#');
   displaySquare(,'&');
   displaySquare(2);

   return 0;
}

void displaySquare(int side = 4, char filledChar = '*')
{
   for(int i = 0; i < side; i++)
   {
      for(int j = 0; j < side; j++)
         cout << filledChar;
      cout << "\n";
   }

}
```

Suppose there is a default arguments function to display a square of any character and the test program as follows.

Are there any bugs in the function or main? Fix them and run the program correctly.

Exercise 3

Overload a function **bool vabs(int * p, int n)** which can compute the absolute value for every element of an array, the array can be int, float and double.

Should n be int or size_t? what's the difference? Remember to check whether the pointer is valid.

Exercise 4

Write a program that uses a function template called ***Compare*** to compare the relationship between the values of the two arguments and return 1 when the first argument is greater than the second one; return -1 when the first argument is smaller than the second one, return 0 when the both values are equal. Test the program using integer, character and floating-point number arguments and print the result of the comparation.

If there is a structure as follows, how to define an explicit specialization of the template function **Compare** and print the result of the comparation?

```
struct stuinfo{
   string name;
    int age;
};
```

The prototype of the Compare:

```
template <typename T>
int Compare(const T &a, const T &b);
```

The output:

```
Compare of the two integers:-1
Compare of the two floats:1
Compare of the two characters:1
Compare of the two structs:1
```