CS 1502H 程序设计思想与方法 (C++)

Chapter 7. 指针-3

陈东尧
上海交通大学
2025年秋季

上节课回顾

- 指针概念及指针变量的定义
- 指针运算
- 指针与数组
- 动态内存分配
- 字符串与指针
- 指针与函数
- 引用类型与函数
- 指针数组与多级指针

引用概念

引用的意义：别名（alias)

给某个变量取一个别名,使一个内存单元可以通过不同的变量名来访问

定义格式

类型　&变量名 = 变量名;

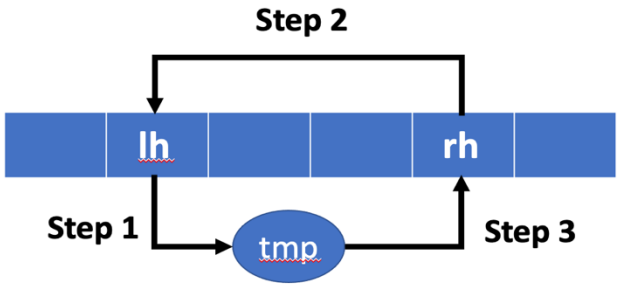
例：int i;

　　int &j = i;

　　j 是 i 的别名，i 与 j 是同一个内存单元。

引用的主要目的有两种：（1）将引用作为函数的参数；（2）将引用作为函数的返回值

【扩展阅读】C++中的引用https://www.tutorialspoint.com/cplusplus/cpp_references.htm




```
void swap(int x, int y);
{
     int temp;
     temp=x;
     x=y;
     y=temp;
}
```

指针参数

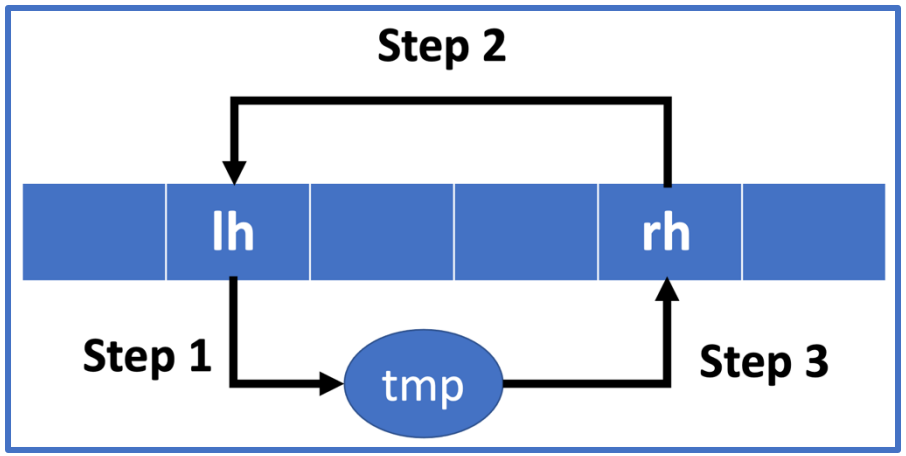
```
void swap(int *m,  int *n)
{
     int temp;
     temp=*m;
    *m=*n;
    *n=temp;
}
```

调用：swap(&x, &y)

引用参数

```
void swap(int &m,  int &n)
{
     int temp;
     temp=m;
     m=n;
     n=temp;
}
```

调用：swap(x, y)




```
void swap(int *m, int *n){
        int temp;
        temp=*m;
        *m=*n;
        *n=temp;
}
swap(&x, &y)
```

```
void swap(int &m, int &n){
        int temp;
        temp=m;
        m=n;
        n=temp;
}
swap(x, y)
```

第一种：引用作为参数传递 (Reference as Parameter)

- C++引用的主要目的是将引用作为函数的参数

指针参数

```
void swap(int *m,  int *n)
{
     int temp;
     temp=*m;
    *m=*n;
    *n=temp;
}
调用：swap(&x, &y)
```

引用参数

```
void swap(int &m,  int &n)
{
     int temp;
     temp=m;
     m=n;
     n=temp;
}
调用：swap( x, y)
```

注意：实参必须是变量，而不能是一个表达式或常量

验证引用传递

```
void f( int & r )
{
    cout <<   “r= “ << r <<endl;
    cout <<   “&r= “ << &r <<endl;
    r= 5;
    cout <<   “r= “ << r <<endl;
}

int main( )
{
    int x=47;
    cout <<   “x= “ << x <<endl;
    cout <<   “&x= “ << &x <<endl;
    f(x);
    cout <<   “x= “ << x <<endl;
}
```

验证引用传递的地址是否不变

```
void f( int & r )
{
    cout <<   “r= “ << r <<endl;
    cout <<   “&r= “ << &r <<endl;
    r= 5;
    cout <<   “r= “ << r <<endl;
}

int main( )
{
    int x=47;
    cout <<   “x= “ << x <<endl;
    cout <<   “&x= “ << &x <<endl;
    f(x);
    cout <<   “x= “ << x <<endl;
}
```

执行结果

x = 47

&x = **0x065FE00**

r = 47

&r = **0x065FE00**

r=5

x = 5

引用传递意义

- 在C++中，函数参数一般都采用引用传递
- 减少函数**调用时的开销**（在后续struct和类的内容展开后，这一点会更明显）
  - 不需要分配新的内存空间
  - 不需要复制实际参数的数值
- 作为输出参数
- 引用传递是地址传递的另一种更简单明了的实现方法。
- 如果在函数内不许改变参数的值，则形式参数用const限定。
- 对非const的参数采用引用传递时，实际参数不能是常量或临时量。

第二种：引用作为函数的返回值（Reference as Return Value）

- 函数的返回值**可以是一个引用**。它表示函数的返回值是函数内某一个变量（不能是局部变量）的引用。
- 如：交换a, b, c中的最大值和最小值的函数也可以用引用实现。
- 返回引用的函数原型：

  类型 &函数名 (形式参数表);

第二种：引用作为函数的返回值（Reference as Return Value）

```
//交换a, b, c中的最大值和最小值
int &max(int &a, int &b, int &c) {
    if (a > b)
    if (a > c)  return(a);   else return(c);
    else if (b > c)   return(b);  else return(c);
}

int &min(int &a, int &b, int &c) {
    if (a < b)
    if (a < c)  return(a);  else return(c);
    else if (b < c) return(b);   else return(c);
}

void swap(int &a, int &b) {
    int c;
    c = a; a = b; b = c;
}
```

第二种：引用作为函数的返回值（Reference as Return Value）

关键用途

将函数用于赋值运算符的左边，即作为**左值**

减少函数返回时的开销

实例

```
int a[] = {1, 3, 5, 7, 9};
int &index(int);    //声明返回引用的函数
void main()
{
    index(2) = 25;    //将a[2]重新赋值为25
    cout << index(2);
}
int &index(int j)
{   return a[j];    }    //返回第j个元素的引用
```

【扩展阅读】C++中的左值和右值：https://nettee.github.io/posts/2018/Understanding-lvalues-and-rvalues-in-C-and-C/

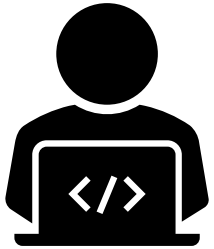
const与引用 （const reference）

定义格式

const 类型 &变量名 = 变量名;

例：int i;

const int &j = i;



代码演示

chapter7-const-reference-trivia.cpp

常量引用的意义

控制变量的修改，不管被引用对象是常量还是变量，用此名字时不能赋值

如 i= 5 是合法的， j = 5 是非法的

const引用的初值可以是常量或表达式。如

const int &y = 2+5;

常量的引用传递 void f(const double &);

返回常量引用的函数

用途

仅减少函数返回时的开销，保证返回变量不被修改

实例

```
int a[] = {1, 3, 5, 7, 9};
const  int &index(int);    //声明返回引用的函数
void main()
{
      index(2) = 25;     //错
      cout << index(2); // 正确
}
const  int &index(int j)
{   return a[j];     }     //函数是a[j]的一个常量引用
```

本章内容

- 指针概念及指针变量的定义
- 指针运算
- 指针与数组
- 动态内存分配
- 字符串与指针
- 指针与函数
- 引用类型与函数
- 指针数组与多级指针
- 指针数组 (Array of Pointers)
- 多级指针 (Multiple indirection)
- 动态二维数组（Dynamic two dimensional array）
- 指向函数的指针 (Function pointers)

指针数组 （Array of pointers）

指针本身也是变量，他们也可以像其他变量一样存储在数组中

指针数组

每个元素均为指针

一维指针数组的定义形式

类型名 *数组名[数组长度];

指针数组 （Array of pointers）

指针本身也是变量，他们也可以像其他变量一样存储在数组中

指针数组

每个元素均为指针

一维指针数组的定义形式

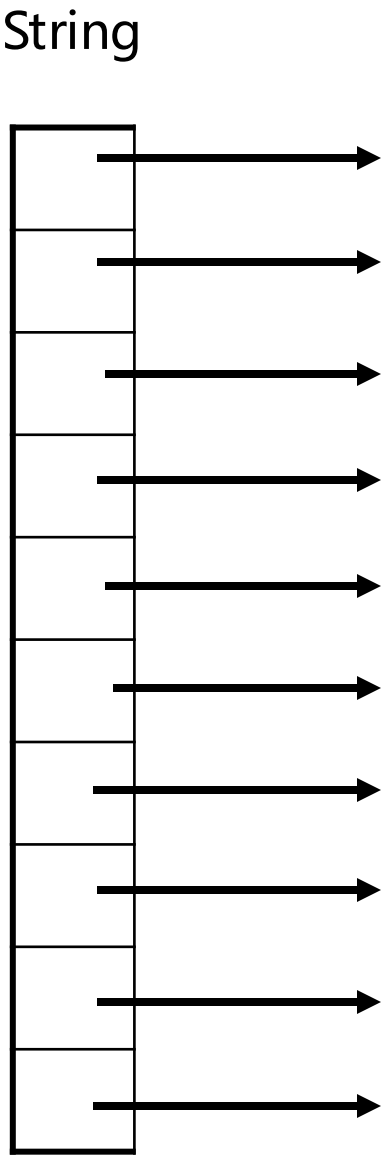
类型名 *数组名[数组长度]；

例如，

char *String[10];

定义了一个名为String的指针数组，该数组有10个元素

数组的每个元素是一个指向字符的指针



指针数组的应用

指向字符的指针数组

- 用来存储一组字符串

实例

- 例：写一个函数用二分法查找某一个城市在城市表中是否出现，用递归实现。

关键问题

- 一组字符串的存储：用指向字符的指针数组
- 查找时的比较：用字符串比较函数

指针数组的应用

指向字符的指针数组

- 用来存储一组字符串

实例

- 例：写一个函数用二分法查找某一个城市在城市表中是否出现，用递归实现。

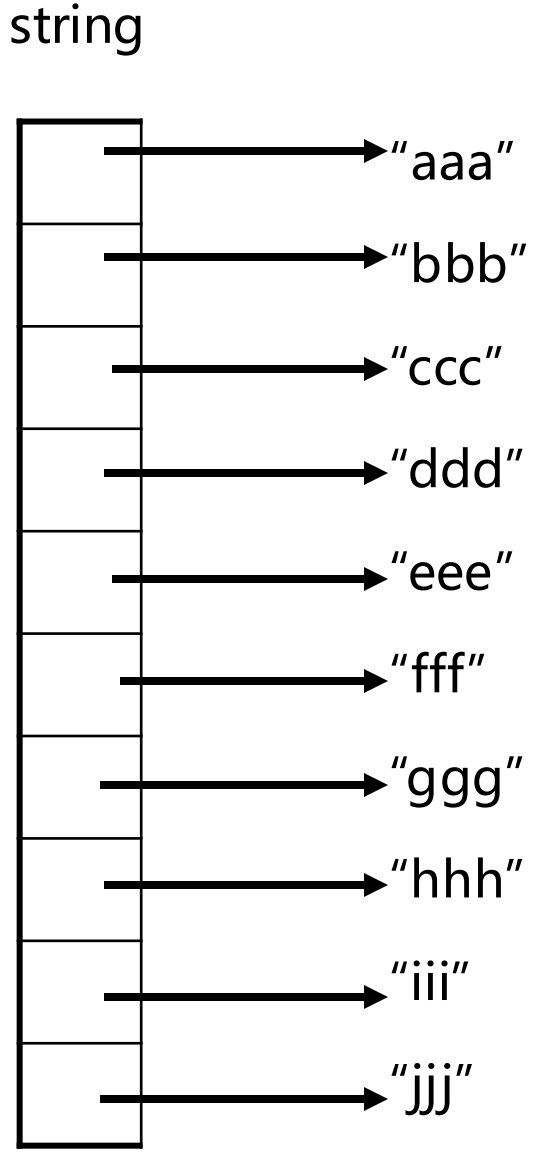
关键问题

- 一组字符串的存储：用指向字符的指针数组
- 查找时的比较：用字符串比较函数

比较函数strcmp

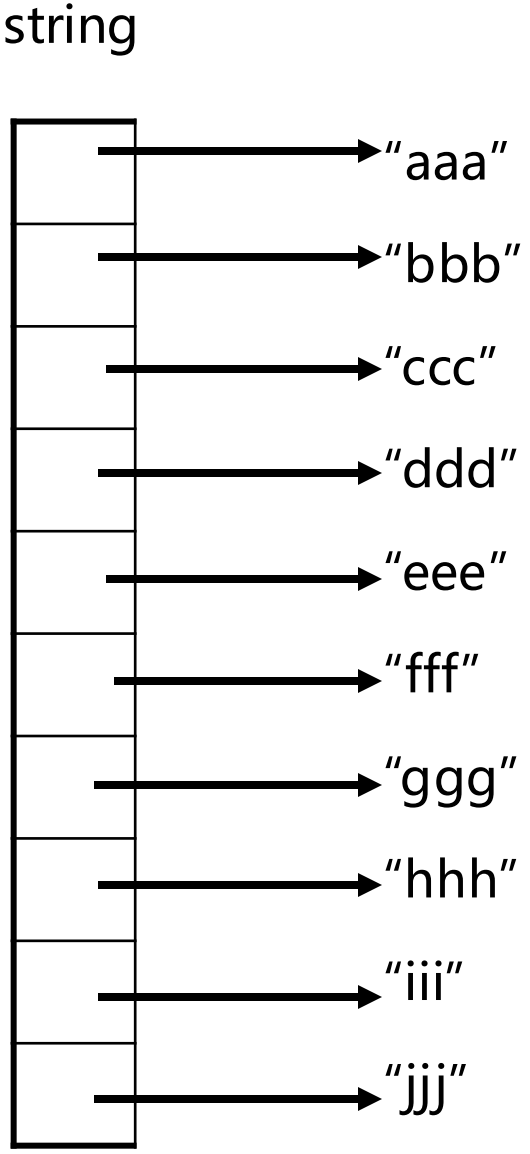
是string compare(字符串比较)的缩写，**用于比较两个字符串并根据比较结果返回整数。**基本形式为strcmp(str1, str2)，若str1==str2，则返回零；若str1<str2，则返回负数；若str1>str2，则返回正数。

函数的应用



函数的应用

```
// 注意const的用法!
int binarySearch(const char *table[ ], int lh, int rh, char *name);

int main()
{
        const char *string[10] = {"aaa", "bbb", "ccc", "ddd", "eee", "fff", "ggg", "hhh",
        "iii", "jjj"};
        char tmp[10];

        while (cin >> tmp)
                cout << binarySearch(string, 0, 9, tmp) << endl;
        return 0;
}
```



接下来如何用递归实现？

- 终止条件
- 递归步骤
- 函数原型的具体设计

接下来如何用递归实现？

- 终止条件

```
mid = (lh + rh) / 2;
result= strcmp(table[mid], name);
if (result == 0)
        return mid; //找到
```

- 递归步骤
- 函数原型的具体设计

接下来如何用递归实现？

- 终止条件

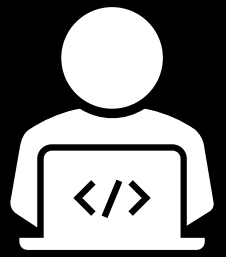
```
mid = (lh + rh) / 2;
result= strcmp(table[mid], name);
if (result == 0)
        return mid; //找到
```

- 递归步骤（注意是已经排好序的！）

```
else if (result > 0)
        return binarySearch(table, lh, mid - 1, name);
else
        return binarySearch(table, mid + 1, rh, name);
```

- 函数原型的具体设计

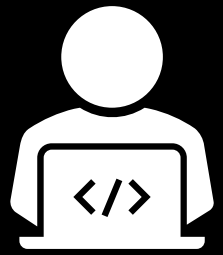
```
//该函数用二分查找在table中查找name是否出现
//lh和rh表示查找范围，返回出现的位置
int binarySearch( const char *table[], int lh, int rh, char *name)
{
int mid, result;

if (lh <= rh) {
	mid = (lh + rh) / 2;
	result= strcmp(table[mid], name);
	if (result == 0)
		return mid; //找到
	else if (result > 0)
		return binarySearch(table, lh, mid - 1, name);
	else
		return binarySearch(table, mid + 1, rh, name);
}

return -1; //没有找到
}
```



代码演示

chapter7-binary-search-recursion.cpp

```cpp
#include<iostream>
#include<cstring>
using namespace std;
int binarySearch(const char *table[ ], int lh, int rh, char *name);

int main()
{
	const char *string[10] = {"aaa", "bbb", "ccc", "ddd", "eee", "fff", "ggg", "hhh", "iii", "jjj"};
	char tmp[10];

	while (cin >> tmp)
		cout << binarySearch(string, 0, 9, tmp) << endl;
	return 0;
}

int binarySearch( const char *table[], int lh, int rh, char *name)
{
	int mid, result;
	if (lh <= rh) {
	mid = (lh + rh) / 2;
	result= strcmp(table[mid], name);
	if (result == 0)
	return mid; //找到
	else if (result > 0)
	return binarySearch(table, lh, mid - 1, name);
	else
	return binarySearch(table, mid + 1, rh, name);
	}
	return -1; //没有找到
}
```



完整代码演示

chapter7-binary-search-recursion.cpp

字符串数组的一些具体用途

```
int main()
```

主函数有没有参数？

字符串数组的一些具体用途

```
int main()
```

主函数有没有参数？

只要有刚需，那通常就已经有解决方案

main函数的形参

如何将参数传递给程序?

main函数的参数

```
int main(int argc, char *argv[])
{ /* … */

}
```

argc (argument count)
argv(argument vector)

main函数有二个形式参数

int argc： 参数的数目（包括命令名本身）

char *argv[ ]：指向每个参数的指针，是一个指向字符串的指针数组

【扩展阅读：C++中传递参数到main里】https://www.geeksforgeeks.org/command-line-arguments-in-c-cpp/

main函数的形参

如何将参数传递给程序?

main函数的参数

```
int main(int argc, char *argv[])
{ /* … */

}
```

argc (argument count)
argv(argument vector)

main函数有二个形式参数

int argc： 参数的数目（包括命令名本身）

char *argv[ ]：指向每个参数的指针，是一个指向字符串的指针数组

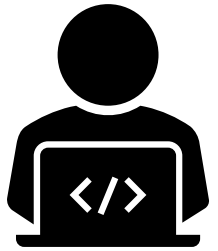
char **argv也是可以的，为什么?

（多级指针中会讲到）

【扩展阅读：C++中传递参数到main里】https://www.geeksforgeeks.org/command-line-arguments-in-c-cpp/

把参数传递给main()

```
// Name of program mainreturn.cpp
#include <iostream>
using namespace std;

int main(int argc, char** argv)
{
        cout << "You have entered " << argc
        << " arguments:" << "\n";

        for (int i = 0; i < argc; ++i)
        cout << argv[i] << "\n";

        return 0;
}
```



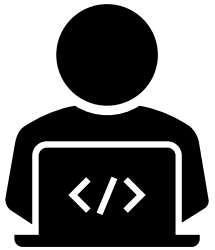
代码演示

chapter7-argc-argv.cpp

带参数的运行指令：

```
./main geeks for geeks
```

把参数传递给main()



代码演示

chapter7-argc-argv.cp

```
// Name of program mainreturn.cpp
#include <iostream>
using namespace std;

int main(int argc, char** argv)
{
        cout << "You have entered " << argc
        << " arguments:" << "\n";

        for (int i = 0; i < argc; ++i)
        cout << argv[i] << "\n";

        return 0;
}
```

编译指令：

```
g++ mainreturn.cpp -o main
```

带参数的运行指令：

```
./main hello world
```

输出：

```
You have entered 3 arguments:
./main
hello
world
```

main函数参数实例

问题

编写一个求任意 *n* 个正整数的平均数的程序

要求

如果该程序对应的可执行文件名为aveg，则可以在命令行中输入

./aveg 10 30 50 20 40↙

表示求10、30、50、20和40的平均值，对应的输出为30

设计考虑

将这些数据作为命令行的参数

从argc得到数据的个数

从argv得到每一个数值，但注意数值是以字符串表示，要进行计算，必须把它转换成真正的数值

设计一个字符串到整数的函数

字符串形式的数字转换到真正的数值

```
int ConvertStringToInt( char *s )
{
      int num = 0;

       while(*s) {
            num = num * 10 + *s -   '0' ;  //注意字符型向整形的转换
             ++s; //指向下一个字符
        }

        return num;
}
```

主程序

```
int main( int argc,  char  *argv[] )
{
      int sum = 0;

      for (int i = 1; i < argc; ++i)
           sum += ConvertStringToInt(argv[i]);

       cout << sum / (argc - 1) << endl;

       return 0;
}
```

本章内容

- 指针概念及指针变量的定义
- 指针运算
- 指针与数组
- 动态内存分配
- 字符串与指针
- 指针与函数
- 引用类型与函数
- 指针数组与多级指针
- 指针数组 (Array of Pointers)
- 多级指针 (Multiple indirection)
- 动态二维数组（Dynamic two dimensional array）
- 指向函数的指针 (Function pointers)

多级指针

指针指向的内容还是一个指针，称为多级指针

如有定义：char *string[10]； string是一个数组，数组元素可以通过指针来访问

- string是指向string[0]的指针
- string[0]的值是一个指针

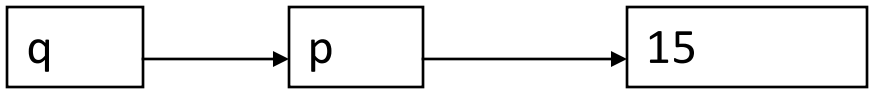
=>string是指向一个指针的指针

多级指针的定义

两级指针

类型名 **变量名;

如：int **q; 表示q指向的内容是一个指向整型的指针。可以这样使用：int x=15, *p=&x; **q = &p;**



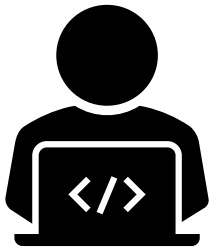
同样：char **s;

表示s指向的内容是一个指向字符的指针

多级指针的应用

用指向指针的指针访问指针数组的元素。如

```
#include <iostream>
using namespace std;
int main()
{
	const char *city[] = {"aaa", "bbb", "ccc", "ddd", "eee"};
	const char **p;
	p = city;
	for ( ; p < city + 5; ++p)
		cout << *p << endl;
		// **p = "kkk";

	return 0;
}
```



代码演示

chapter7-multiple-indirection.cpp

多级指针的应用

用指向指针的指针访问指针数组的元素。如

```
#include <iostream>
using namespace std;
int main()
{
	const char *city[] = {"aaa", "bbb", "ccc", "ddd", "eee"};
	const char **p;
	p = city;
	for ( ; p < city + 5; ++p)
		cout << *p << endl;
		// **p = "kkk";

	return 0;
}
```

输出结果：

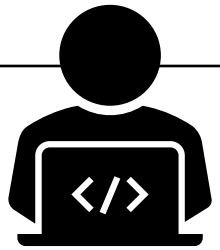
aaa

bbb

ccc

ddd

eee



代码演示

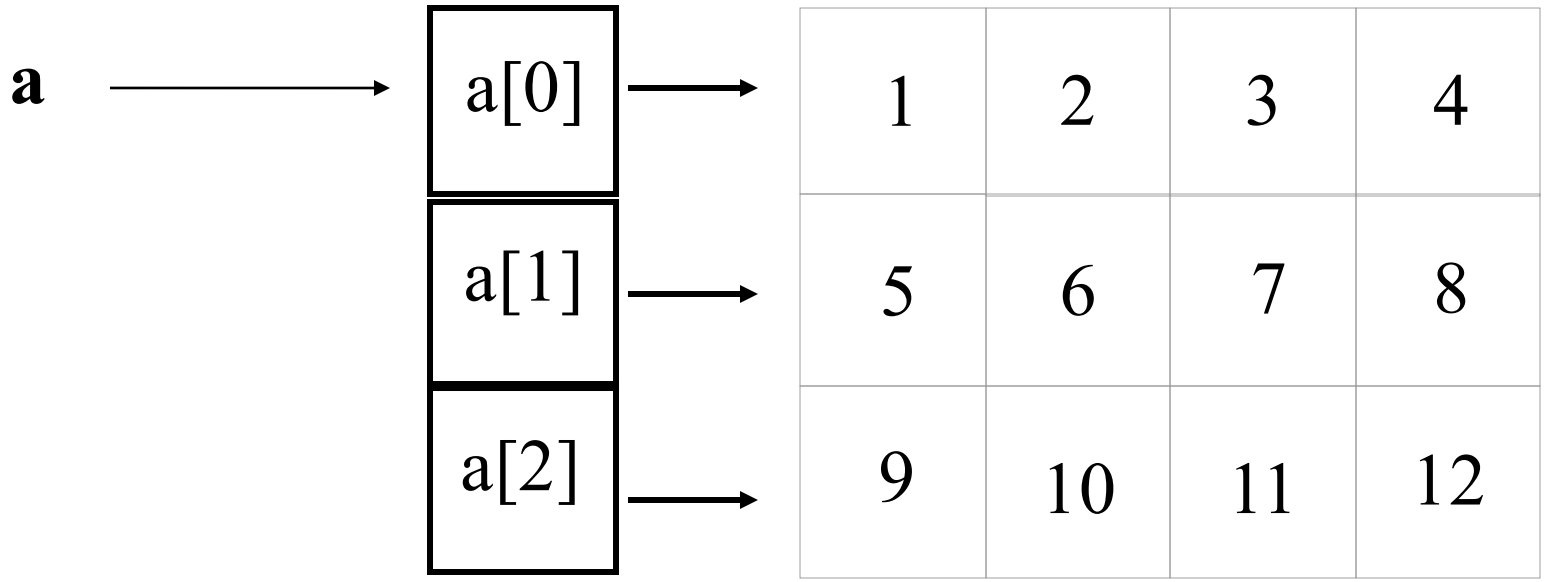
chapter7-multiple-indirection.cpp

本章内容

- 指针概念及指针变量的定义
- 指针运算
- 指针与数组
- 动态内存分配
- 字符串与指针
- 指针与函数
- 引用类型与函数
- 指针数组与多级指针
- 指针数组 (Array of Pointers)
- 多级指针 (Multiple indirection)
- 动态二维数组（Dynamic two dimensional array）
- 指向函数的指针 (Function pointers)

回顾二维数组

- int a[3][4]; //等价于定义了3个指针变量



回顾动态内存申请

- int *a = new int[10]

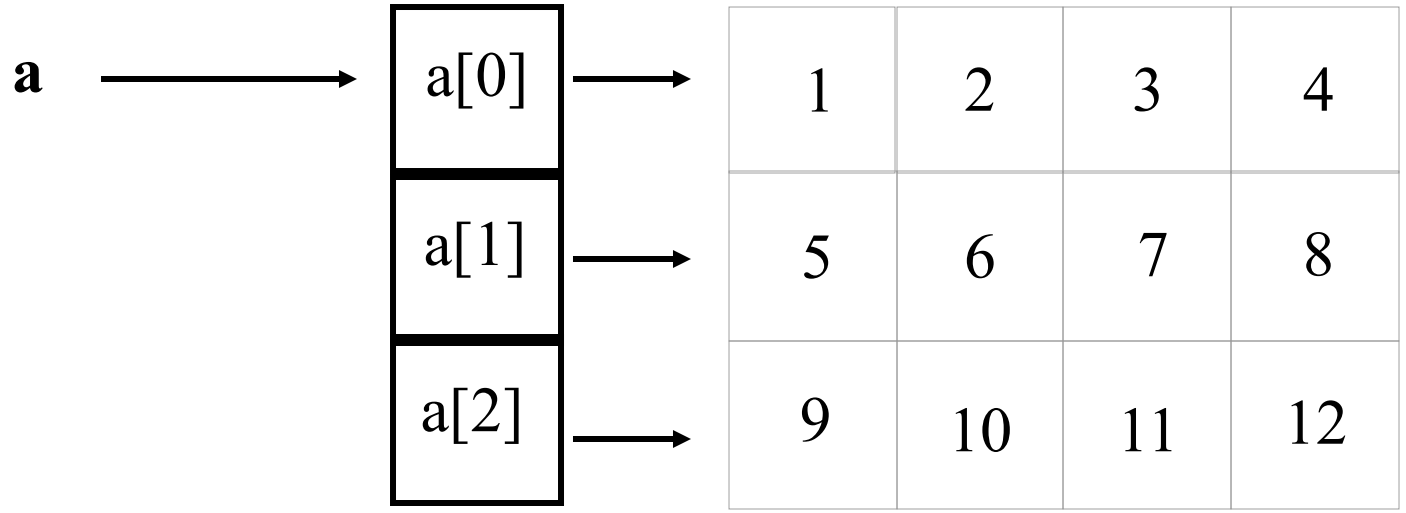
动态二维数组

- **C++不能直接申请动态二维数组，如new int[5][9]是非法的表达式**

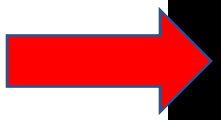
动态二维数组

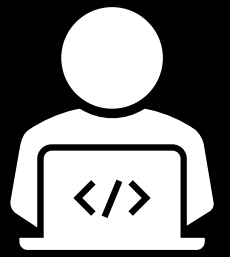
- 解决方法
  - `int **a; a = new int*[3]; // 动态定义了3个指针变量`

    `for(int i=0; i<3; ++i) a[i] = new int[4];`



```
int main()
{
    int **a, i, j, k = 0;
    a = new int *[3]; // 申请指针数组

    // 申请每一行空间
    for (i = 0; i < 3; ++i)
        a[i] = new int[4];
    for (i = 0; i < 3; ++i){ // 二维数组赋值
        for (j = 0; j < 4; ++j)
            a[i][j] = k++;
    }
    for (i = 0; i < 3; ++i) { // 二维数组输出
        cout << endl;
        for (j = 0; j < 4; ++j)
            cout << a[i][j] << '\t';
    }
    for (i = 0; i < 3; ++i) // 二维数组空间释放
        delete [] a[i];
    delete [] a;
    return 0;
}
```

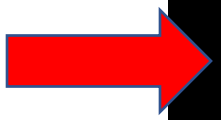


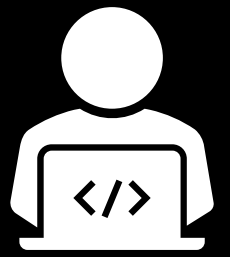


代码演示

chapter7-dynamic-2D-array.cpp

```
int main()
{
    int **a, i, j, k = 0;
    a = new int *[3]; // 申请指针数组

    // 申请每一行空间
    for (i = 0; i < 3; ++i)
        a[i] = new int[4];
    for (i = 0; i < 3; ++i){ // 二维数组赋值
        for (j = 0; j < 4; ++j)
            a[i][j] = k++;
    }
    for (i = 0; i < 3; ++i) { // 二维数组输出
        cout << endl;
        for (j = 0; j < 4; ++j)
            cout << a[i][j] << '\t';
    }
    for (i = 0; i < 3; ++i) // 二维数组空间释放
        delete [] a[i];
    delete [] a;
    return 0;
}
```

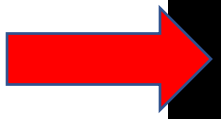


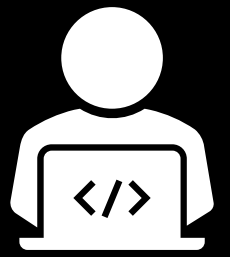


代码演示

chapter7-dynamic-2D-array.cpp

```
int main()
{
    int **a, i, j, k = 0;
    a = new int *[3]; // 申请指针数组

    // 申请每一行空间
    for (i = 0; i < 3; ++i)
        a[i] = new int[4];
    for (i = 0; i < 3; ++i){ // 二维数组赋值
        for (j = 0; j < 4; ++j)
            a[i][j] = k++;
    }
    for (i = 0; i < 3; ++i) { // 二维数组输出
        cout << endl;
        for (j = 0; j < 4; ++j)
            cout << a[i][j] << '\t';
    }
    for (i = 0; i < 3; ++i) // 二维数组空间释放
        delete [] a[i];
    delete [] a;
    return 0;
}
```





代码演示

chapter7-dynamic-2D-array.cpp

指向数组的指针

- 对于int a[3][4]，a[ ]是一个指针数组，它的每个元素是一个整型指针，指向每一行的第零个元素。a是一个指向一维数组的指针，指向a[ ]的第零个元素（即第零行）。对a加1，事实上是跳到下一行。

- 指向一维数组的指针可以这样定义：

  类型名 （*指针变量名）[一维数组的元素个数]；//定义了一个指针

  - 注意：数组指针不是指针数组，**圆括号不能省略**，如果省略了圆括号就变成了指针数组，即定义了多个指针。

指向数组的指针（pointer to array）-扩展阅读（含实例）：
https://www.geeksforgeeks.org/pointer-array-array-pointer/

指向数组头的指针和指向数组的指针是不同的

- P指向arr的第0个元素；ptr指向整个arr数组
- 因此，p的基类是一个int，ptr的基类是“含有5个整形的数组”

结果：

p =0x7ff7bb33bcf0, ptr = 0x7ff7bb33bcf0

p =0x7ff7bb33bcf4, ptr = 0x7ff7bb33bd04

代码演示

chapter7-pointertozero-vs-pointertoarray.cpp

```
#include <iostream>
using namespace std;
int main()
{
        // Pointer to an integer
        int *p;
        // Pointer to an array of 5 integers
        int (*ptr)[5];
        int arr[5];
        // Points to 0th element of the arr.
        p = arr;
        // Points to the whole array arr.
        ptr = &arr;
        cout << "p =" << p <<", ptr = "<< ptr<< endl;
        p++;
        ptr++;
        cout << "p =" << p <<", ptr = "<< ptr<< endl;
        return 0;
}
// This code is contributed by SHUBHAMSINGH10
```

指向数组头的指针和指向数组的指针是不同的

- P指向arr的第0个元素；ptr指向整个arr数组
- 因此，p的基类是一个int，ptr的基类是“含有5个整形的数组”

起始地址是一致的

结果：

p =0x7ff7bb33bcf0, ptr = 0x7ff7bb33bcf0

p =0x7ff7bb33bcf4, ptr = 0x7ff7bb33bd04

+1后，进了4bytes　　+1后，进了20bytes

```cpp
#include <iostream>
using namespace std;
int main()
{
        // Pointer to an integer
        int *p;
        // Pointer to an array of 5 integers
        int (*ptr)[5];
        int arr[5];
        // Points to 0th element of the arr.
        p = arr;
        // Points to the whole array arr.
        ptr = &arr;
        cout << "p =" << p <<", ptr = "<< ptr<< endl;
        p++;
        ptr++;
        cout << "p =" << p <<", ptr = "<< ptr<< endl;
        return 0;
}
// This code is contributed by SHUBHAMSINGH10
```

代码演示

chapter7-pointertozero-vs-pointertoarray.cpp

指向数组的指针

- 等价于a[i][j]的表达形式

a[i][j]
*(a[i] + j)
(*(a + i))[j]
(((a + i)) + j)

本章内容

- 指针概念及指针变量的定义
- 指针运算
- 指针与数组
- 动态内存分配
- 字符串与指针
- 指针与函数
- 引用类型与函数
- 指针数组与多级指针
- 指针数组 (Array of Pointers)
- 多级指针 (Multiple indirection)
- 动态二维数组（Dynamic two dimensional array）
- 指向函数的指针 (Function pointers)

指向函数的指针(Function pointers)

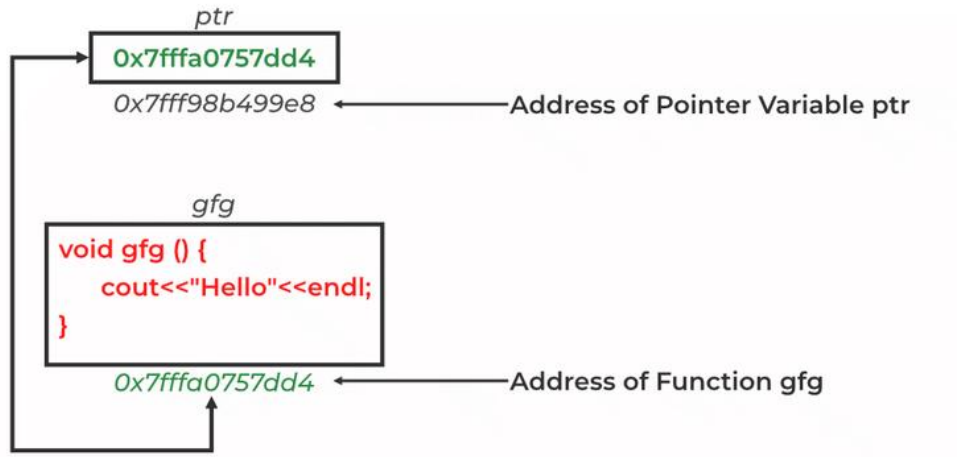
函数的指针

指向函数代码的**入口地址**

定义

返回类型（*指针变量名)(形式参数表）;

赋值(注意原型要一致)，例如：



指向函数的指针(Function pointers)

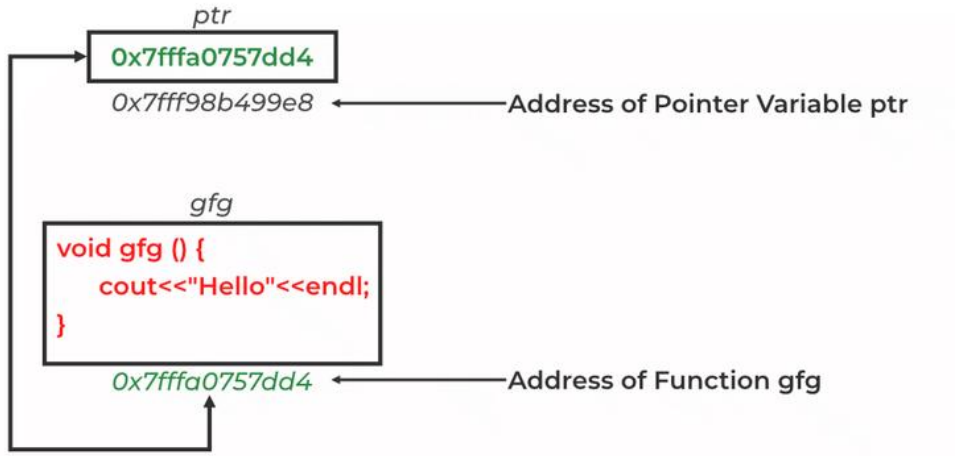
函数的指针

指向函数代码的**入口地址**

定义

返回类型（*指针变量名)(形式参数表）;

赋值(注意原型要一致)，例如：




```
int multiply(int a, int b) { return a * b; }

int main() {
        int (*func)(int, int); // func is pointing to the multiply function
        func = multiply;
        int prod = func(15, 2);
        cout << "The value of the product is: " << prod << endl;
        return 0;
}
```

指向函数的指针(Function pointers)

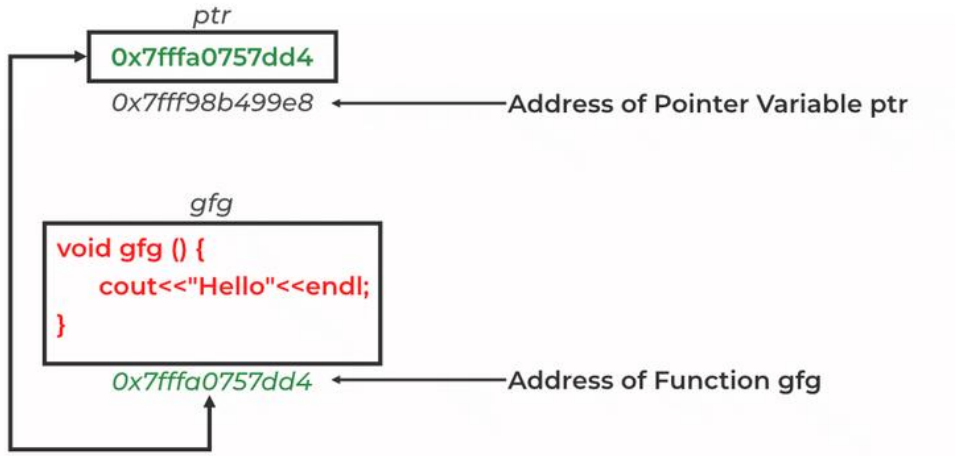
函数的指针

指向函数代码的**入口地址**

定义

返回类型（*指针变量名)(形式参数表）;

赋值(注意原型要一致)，例如：



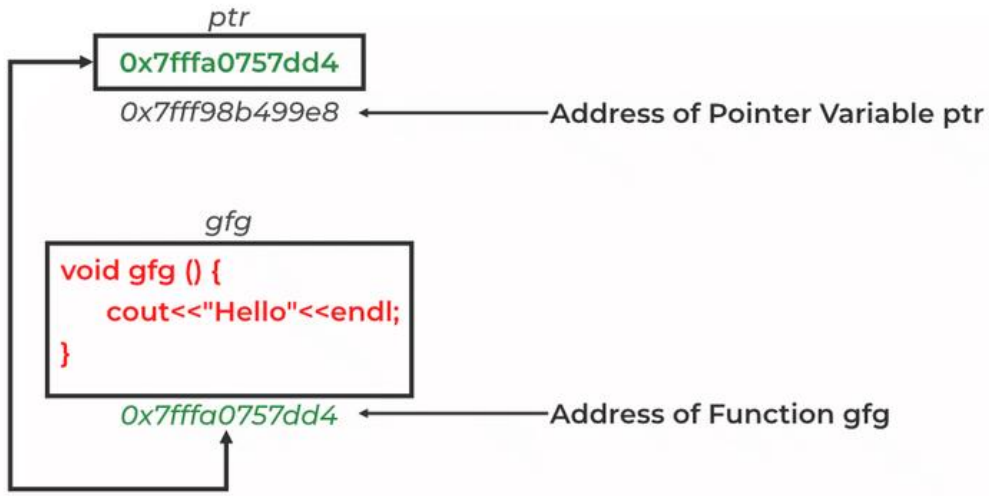

```
int multiply(int a, int b) { return a * b; }

int main() {
        int (*func)(int, int); // func is pointing to the multiply function
        func = multiply;
        int prod = func(15, 2);
        cout << "The value of the product is: " << prod << endl;
        return 0;
}
```

用途（指向同类函数入口，便于访问），典型例子：菜单、函数参数（callback）

指向函数的指针

函数的指针

指向函数代码的**入口地址**



定义

返回类型（*指针变量名)(形式参数表）;

赋值

int isdigit(int n, int k) { ... }

int (*p)(int, int);

p=isdigit;

原型一致

用途（刚需为：对同一大类的函数入口编有序号，可以便于访问）

菜单选择

函数参数

函数指针用于菜单选择

例如，在一个工资系统中有如下功能

- 添加员工
- 删除员工
- 修改员工信息
- 打印工资单
- 打印汇总表

设计考虑：把每个功能设计成一个函数

- 添加员工的函数为add
- 删除员工的函数为delete
- 修改员工信息的函数为modify
- 打印工资单的函数为printSalary
- 打印汇总表函数为printReport

主程序是一个循环，显示所有功能和它的编号，请用户输入编号，根据编号调用相应的函数。

```
int main()
{
        int select;
        while(true) {
            cout << "1--add \n";
            cout << "2--delete\n";
            cout << "3--modify\n";
            cout << "4--print salary\n";
            cout << "5--print report\n";
            cout << "0--quit\n";
            cin >> select;

            switch(select) {
                case 0: return 0;
                case 1: add(); break;
                case 2: erase(); break;
                case 3: modify(); break;
                case 4: printSalary(); break;
                case 5: printReport(); break;
                default: cout << "input error\n";
            }
    }
}
```

假设add、erase等功能都已经定义好

```
void add(){
cout<<"add func"<<endl;
}
void erase(){
cout<<"erase func"<<endl;
}
void modify(){
cout<<"modify func"<<endl;
}
void printSalary(){
cout<<"printSalary func"<<endl;
}
void printReport(){
cout<<"printReport func"<<endl;
}
```

缺点

需要一个长长的switch

利用指向函数的指针

```
int main()
{
        int select;
        void ( *func[6] )() = {NULL, add, erase, modify, printSalary, printReport};
        while(1) {
        cout << "1--add \n";
        cout << "2--erase\n";
        cout << "3--modify\n";
        cout << "4--print salary\n";
        cout << "5--print report\n";
        cout << "0--quit\n";
        cin >> select;
        if (select == 0) return 0;
        if (select > 5) cout << "input error\n";
        else func[select]();
        }
}
```

函数指针作为函数参数

问题

设计一个**通用的**冒泡排序函数，可以**排序任何类型的数据**

关键问题

如何解决任意类型数据的存问题？

将冒泡排序设计成一个函数模板，将待排序的数据类型设计成模板参数

如何解决不同类型的数据**有不同的比较方式**？

向排序函数传递一个比较函数来解决

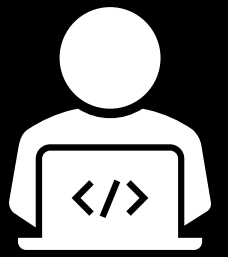
```
void sort(int a[], int size)
{
    bool flag;
    int i, j;

    for (i = 1; i < size; ++i)   {
        flag = false;
        for (j = 0;  j <size - i;  ++j)
            if (a[j+1] < a[j])  {
                int tmp = a[j];
                a[j] = a[j+1];
                a[j+1] = tmp;
                flag = true;
            }
        if (!flag) break;
    }
}
```

```
template <class T>
void sort(T a[], int size, bool (*f)(T,T))
{
   bool flag;
   int i, j;

   for (i = 1; i < size; ++i)   {
        flag = false;
        for (j = 0;  j <size - i;  ++j)
            if ( f( a[j ], a[j +1 ] ))  {
                  T  tmp = a[j];
                  a[j] = a[j+1];
                  a[j+1] = tmp;
                  flag = true;
            }
         if (!flag) break;
   }
}
```

如何兼容templat和函数指针?

应用

```
bool decreaseInt( int x, int y ) { return x < y; }
bool increaseString( char *x, char *y) { return strcmp( x, y ) > 0; }

int main()
{
        int a[] = {3,1,4,2,5,8,6,7,0,9}, i;
        char *b[]= {"aaa", "bbb", "fff", "ttt", "hhh", "ddd", "ggg", "www", "rrr" ,"vvv"};
        sort( a, 10, decreaseInt);
        for ( i = 0; i < 10; ++i) cout << a[i] << "\t";
        cout << endl;

        sort(b, 10, increaseString);
        for (i = 0; i < 10; ++i) cout << b[i] << "\t";
        cout << endl;
        return 0;
}
```



代码演示

chapter7-bubblesort-template.cpp

总结

- 本章介绍了指针的概念，指针变量的定义、运算。
- 指针与数组的关系。
- 动态分配内存；堆和栈的概念。
- 用指针表示字符串及其相关操作。
- 指针与函数；将指针作为形式参数可以使一个函数与其调用函数共享数据。
- 引用类型作为函数的参数或返回值。
- 指针数组、多级指针与多维数组之间的关系。