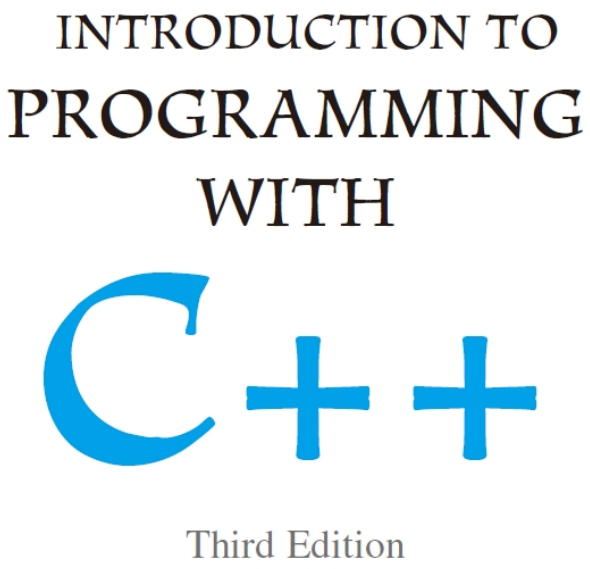
Introduction to Programming with C++

Chieh-Sen (Jason) Huang

Department of Applied Mathematics

National Sun Yat-sen University



Contents are based on book by Y. Daniel Liang

Two-Dimensional Array

- We can use a two-dimensional array to store a matrix or a table.

- An element in a two-dimensional array is accessed through a row and column index.

```
elementType arrayName[ROW_SIZE][COLUMN_SIZE];
```

```
int m[4][3] = {{1,   2,   3},
               {4,   5,   6},
               {7,   8,   9},
               {10, 11, 12}};
\\ m[2][1] = 8
```

	[0]	[1]	[2]
[0]	1	2	3
[1]	4	5	6
[2]	7	8	9
[3]	10	11	12

Processing Two-Dimensional Arrays

- When passing a two-dimensional array to a function, C++ requires that the column size be specified in the function parameter type declaration.

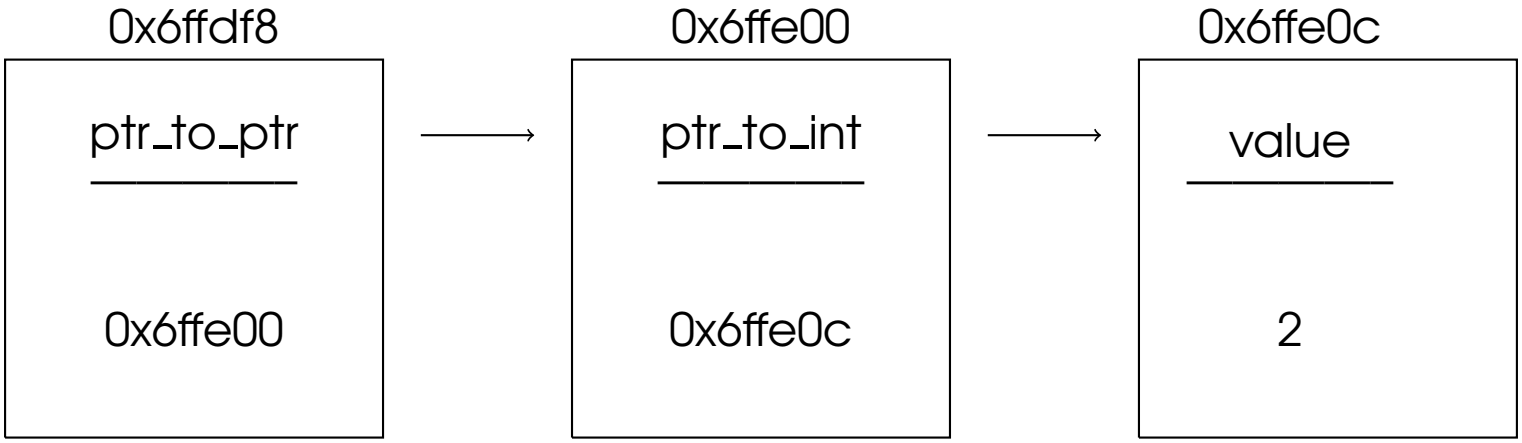
```
void sum(int x[][3], int y[][3], int z[][3])
{
  for(int i = 0; i < 2; i++) //Find sum of two matices
    for(int j = 0; j < 3; j++) //List8_1a.cpp
      z[i][j] = x[i][j] + y[i][j];
}
int main(void)
{
  int x[2][3]={{1, 2, 3},{4, 5, 6}};
  int y[2][3]={{1, 5, 8},{5, 3, 1}};
  int z[2][3]={{0, 0, 0},{0, 0, 0}};

  sum(x, y, z);
}
```

Pointer to Pointer (Multiple Indirection)

- A pointer to a pointer is a form of multiple indirection or a chain of pointers.
- Normally, a pointer contains the address of a variable.
- When we define a pointer to a pointer, the first pointer contains the address of the second pointer.

Pointer to Pointer (Multiple Indirection)




```
int value = 2; //List8_2a.cpp
int *ptr_to_int = &value; //1st pointer
int ** ptr_to_ptr =  &ptr_to_int ; //ptr to ptr

cout << "   Name      |   Address              |  Value \n";
cout << "-----------------------------------------\n";
cout << "ptr_to_ptr   |" << &ptr_to_ptr << "|" << ptr_to_ptr << endl;

cout <<"ptr_to_int   |" << &ptr_to_int << "|" << ptr_to_int << endl;

cout << "value        |" << &value << "   |  " << value << endl;

cout << "*ptr_to_ptr  |**********   |  "       << *ptr_to_ptr << endl;

cout << "**ptr_to_ptr |**********   |  "       << **ptr_to_ptr << endl;
-------------------------------------------------------------------------
  Name       |   Address   |  Value
-----------------------------------------
ptr_to_ptr   |  0x6ffdf8   |  0x6ffe00
ptr_to_int   |  0x6ffe00   |  0x6ffe0c
value        |  0x6ffe0c   |  2

*ptr_to_ptr  |  ********** |  0x6ffe0c
**ptr_to_ptr |  ********** |  2
```

Pointer to Pointer as a matrix

- `a` points to an array of pointer (pointer array).
- `a[i]` points to an integer array.
- We then use `a[i][j]` for values in `**a`.

```
int **a; // declare a pointer to pointer

a = new int* [ROW]; //allocate a pointer array
for ( i=0; i < ROW; i++ ){
 a[i] = new int[COL]; //allocate an int[COL] for
}                          // each pointer a[i]

for ( i=0; i < ROW; i++ ){
 for ( j=0; j < COL; j++ ){
   a[i][j] = i + j;
   printf("%d ", a[i][j]);
 }
 printf("\n");
}
```

- We could also pass pointer to pointer `**a` as `void sum(int **a)`, see List8_4a.cpp

Pointer to Pointer as a matrix

In-Class Exercise: Redo List8_1a.cpp using pointer to pointer.
Homework 8.1: Write a code to find the trace of a 3×3 matrix using pointer to pointer.

Passing a function as a parameter in C++

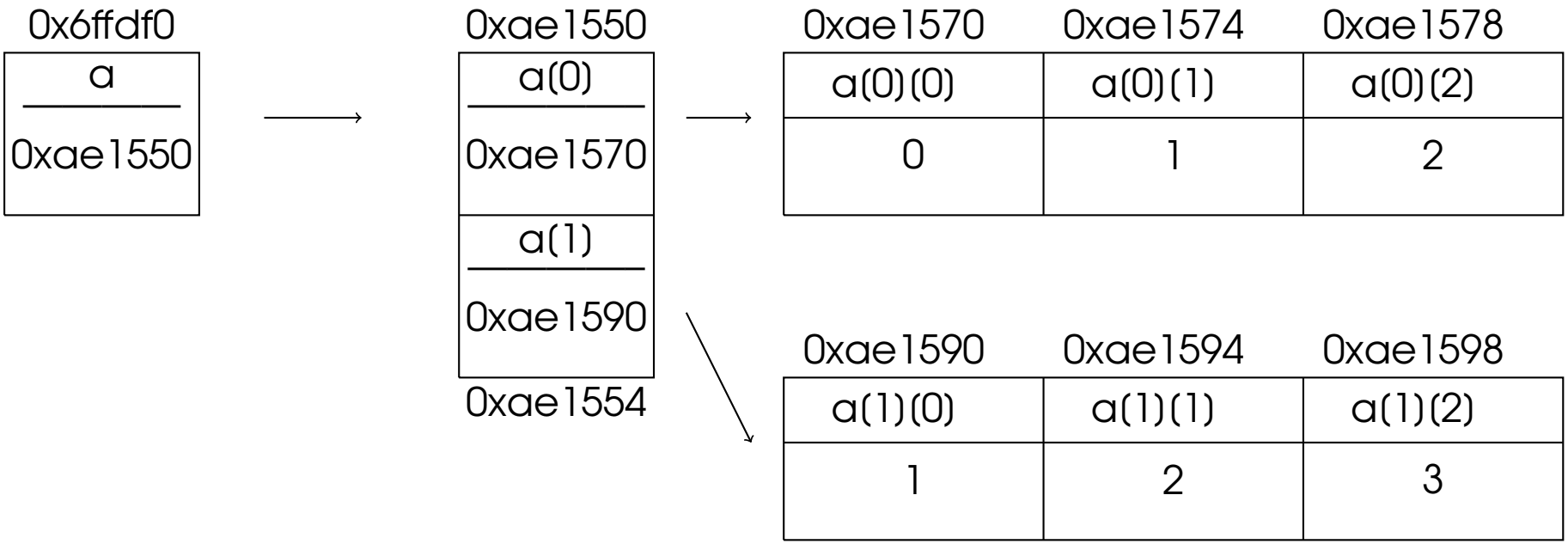
- Passing a function as a parameter is quite a useful concept in C/C++.
- A function can be passed to another function as an argument by passing its address to that function.
- We could retrive the address of a function using the **ampersand (&)** symbol.

```
int sum(int a, int b)
{
    return a + b;
}
// Function that takes a pointer
// to the function
int pointer_arg(int a, int b, int (*func)(int, int))
{
    return func(a, b);
}
int main()
{
    // Pass pointers for adding
    cout << "Sum of 30 and 20 = ";
    cout << pointer_arg(30, 20, &sum)<<endl;

    return 0;
}
```

- See List8_5a.cpp.

The difference between $a[][]$ and $**a$




```
int **a;
a = new int* [ROW];
for ( i = 0; i < ROW; i++ ){
  a[i] = new int[COL];
}
cout << "**a = " << **a <<endl;
cout << "*a = " << *a << " " << *(a+1) << endl;
cout << "a[0] = " << a[0] << " " << a[1] << endl;
cout << "&a[0][0] = " << &a[0][0] << endl;
cout << "&a[0][1] = " << &a[0][1] << endl;
cout << "&a[0][2] = " << &a[0][2] << endl;
cout << "&a[1][0] = " << &a[1][0] << endl;
cout << "&a[1][1] = " << &a[1][1] << endl;
cout << "&a[1][2] = " << &a[1][2] << endl;
cout << endl;
cout << "a = " << a << " " << &a[0] << endl;
cout << "&a = " << &a << endl;
```

```
**a = 0
*a = 0xae1570 0xae1590
a[0] = 0xae1570 0xae1590
&a[0][0] = 0xae1570
&a[0][1] = 0xae1574
&a[0][2] = 0xae1578
&a[1][0] = 0xae1590
&a[1][1] = 0xae1594
&a[1][2] = 0xae1598

a = 0xae1550 0xae1550
&a = 0x6ffdf0
```

The difference between $a[][]$ and $**a$

0x6ffdd0	0x6ffdd4	0x6ffdd8
b(0)(0)	b(0)(1)	b(0)(2)
1	2	3
b(1)(0)	b(1)(1)	b(1)(2)
4	2	3
0x6ffddc	0x6ffde0	0x6ffde4

```
int b[ROW][COL]={1, 2, 3, 4, 5, 6};
for ( i = 0; i < ROW; i++ ){
  for ( j = 0; j < COL; j++ ){
  cout<< &b[i][j] << "  ";
  }
  cout << endl;
  cout << "b[" << i << "] = " << b[i] << endl;
}
cout << endl;
cout << "&b = " << &b << endl;
```

```
0x6ffdd0  0x6ffdd4  0x6ffdd8
b[0] = 0x6ffdd0
0x6ffddc  0x6ffde0  0x6ffde4
b[1] = 0x6ffddc

&b = 0x6ffdd0
```