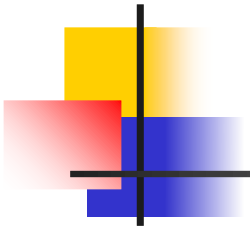
Functions with Pass-by-Reference

Lab 8: Strange Sorting

Rung-Bin Lin

International Bachelor Program in Informatics
Yuan Ze University

Nov. 14, 2023



Purposes of the Lab

- **Understanding the followings:**
  - **Pass by value & pass by reference**
  - **Reference type**

Reference Type

```
1  // Fig. 5.19: fig05_19.cpp
2  // Initializing and using a reference.
3  #include <iostream>
4  using namespace std;
5
6  int main()
7  {
8     int x = 3;
9     int &y = x; // y refers to (is an alias for) x
10
11    cout << "x = " << x << endl << "y = " << y << endl;
12    y = 7; // actually modifies x
13    cout << "x = " << x << endl << "y = " << y << endl;
14 } // end main
```

```
x = 3
y = 3
x = 7
y = 7
```

Fig. 5.19 | Initializing and using a reference.

```
1  // Fig. 5.20: fig05_20.cpp
2  // References must be initialized.
3  #include <iostream>
4  using namespace std;
5
6  int main()
7  {
8     int x = 3;
9     int &y; // Error: y must be initialized
10
11    cout << "x = " << x << endl << "y = " << y << endl;
12    y = 7;
13    cout << "x = " << x << endl << "y = " << y << endl;
14 } // end main
```

Fig. 5.20 | Uninitialized reference causes a compilation error. (Part 1 of 2.)

Pass-by-Value vs. Reference

```
1  // Fig. 5.18: fig05_18.cpp
2  // Comparing pass-by-value and pass-by-reference with references.
3  #include <iostream>
4  using namespace std;
5
6  int squareByValue( int ); // function prototype (value pass)
7  void squareByReference( int & ); // function prototype (reference pass)
8
9  int main()
10 {
11    int x = 2; // value to square using squareByValue
12    int z = 4; // value to square using squareByReference
13
14    // demonstrate squareByValue
15    cout << "x = " << x << " before squareByValue\n";
16    cout << "Value returned by squareByValue: "
17       << squareByValue( x ) << endl;
18    cout << "x = " << x << " after squareByValue\n" << endl;
19
20    // demonstrate squareByReference
21    cout << "z = " << z << " before squareByReference" << endl;
22    squareByReference( z );
23    cout << "z = " << z << " after squareByReference" << endl;
24 } // end main
```

Fig. 5.18 | Passing arguments by value and by reference. (Part 1 of 2.)

```
25
26  // squareByValue multiplies number by itself, stores the
27  // result in number and returns the new value of number
28  int squareByValue( int number )
29  {
30     return number *= number; // caller's argument not modified
31  } // end function squareByValue
32
33  // squareByReference multiplies numberRef by itself and stores the result
34  // in the variable to which numberRef refers in function main
35  void squareByReference( int &numberRef )
36  {
37     numberRef *= numberRef; // caller's argument modified
38  } // end function squareByReference
```

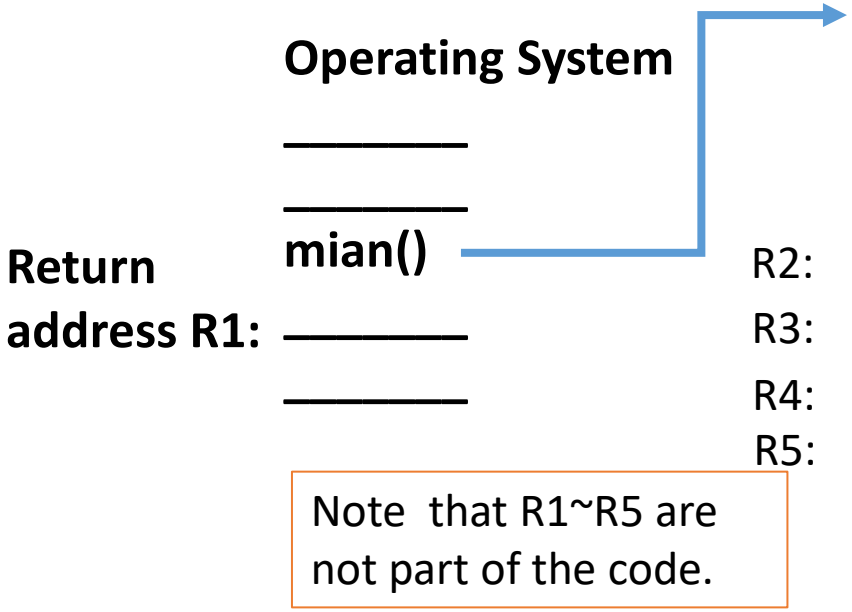
```
x = 2 before squareByValue
Value returned by squareByValue: 4
x = 2 after squareByValue

z = 4 before squareByReference
z = 16 after squareByReference
```

Fig. 5.18 | Passing arguments by value and by reference. (Part 2 of 2.)

Activation Records

- When a function is called, an activation record (AR) is pushed into a *stack*. After executing the function, the activation record is popped (removed) from the stack. Stack is a piece of Last-in-first-out memory. Data can only be stored or retrieved from the top of the stack.



Note that R1~R5 are not part of the code.

AR of main()

Return address R1	
x	10
number	20

```
Int main(){
Int x=10;
Int number = 20;
squareByValue(x);  // call_1
R2: squareByValue(number); // call_2
R3: squareByReference(x); // call_3
R4: squareByReference(number); // call_4
R5: Return 0;
}
Int squareByValue (int number){
return number*number;
}

void squareByReference(int &numberRef){
numberRef = numberRef * numberRef;
}
```

More on Pass-by-Value vs. Reference (1)

```
Int main(){
Int x = 10 , number = 20;
squareByValue(x);  // call_1
Ret addr R2: squareByValue(number); // call_2
Ret addr R3: squareByReference(x); // call_3
Ret addr R4: squareByReference(number); // call_4
Ret addr R5: return 0; }
```

```
Int squareByValue (int number){
return number*number; }
```

```
void squareByReference(int &numberRef){
numberRef = numberRef * numberRef; }
```

Making call_1

AR of call_1

Return address R2
number 10

AR of main()

Return address R1
x 10
number 20

After executing call_1

AR of main()

Return address R1
x 10
number 20

Making call_2

AR of call_2

Return address R3
number 20

AR of main()

Return address R1
x 10
number 20

After executing call_2

AR of main()

Return address R1
x 10
number 20

Also see
https://courses.washington.edu/css342/zander/css332/passby.html
for another example.

More on Pass-by-Value vs. Reference (2)

```
                Int main(){
                Int x = 10 , number = 20;
                squareByValue(x);  // call_1
Ret addr R2:    squareByValue(number); // call_2
Ret addr R3:    squareByReference(x); // call_3
Ret addr R4:    squareByReference(number); // call_4
Ret addr R5:    return 0; }

                Int squareByValue (int number){
                return number*number; }

                void squareByReference(int &numberRef){
                numberRef = numberRef * numberRef; }
```

Making call_3

AR of call_3

AR of main()

Return address R4
Address of *x* in main() for *numberRef*
Return address R1
x 10
number 20

After executing call_4

AR of main()

Return address R1
x 100
number 400

Making call_4

AR of call_4

AR of main()

Return address R5
Address of *number* in main() for *numberRef*
Return address R1
x 100
number 20

After executing call_3

AR of main()

Return address R1
x 100
number 20

Passing an Array to Function

- An array can only be passed to a function by reference.
- Two ways – both need to pass the starting address of an array and the number of elements in the array to a function.
  - Using array name. Array name itself can be used as the starting address of an array.

    int anArray[100] = {0}; // an integer array whose elements are initialized to 0
    int aFunc(**int [ ], int**); // a function prototype that has an array parameter

    Making calls:
    aFunc(anArray, 100);

  - Using the address of the first element in the array.

    Making calls:
    aFunc(&anArray[0], 100); // **&** is an operator that takes the address of a variable

Lab 8: Strange Sorting

- **Write a program to sort integers in ascending order.**
  - **You should write two functions with the following prototypes:**

    int findMax(int [], int, int&);

    The first parameter **int[ ]** holds a list of unsorted integers.
    The second parameter **int** holds the number of integers.
    The third parameter **int&** holds the location of the maximum number in the array.
    The return value should be the maximum number in the array.

    void findMin(int [], int, int &, int&);

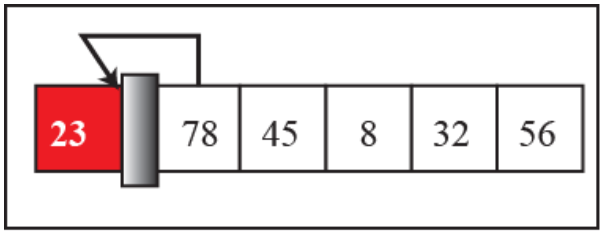
    The first parameter **int[ ]** holds a list of unsorted integers.
    The second parameter **int** holds the number of integers.
    The **third** parameter **int&** holds the minimum number in the array.
    The fourth parameter **int&** holds the location of the minimum number in the array.

  - **You should also write a main() function that uses only the above two functions to sort the given numbers. You certainly are allowed to use more than one array to hold data. The main function should look like as follows:**

```
int main()
{
    for each test case:
     read a list of integers.
     Use a loop to sort the integers by calling the above two functions.
     Print out the sorted integers on a line.
}
```

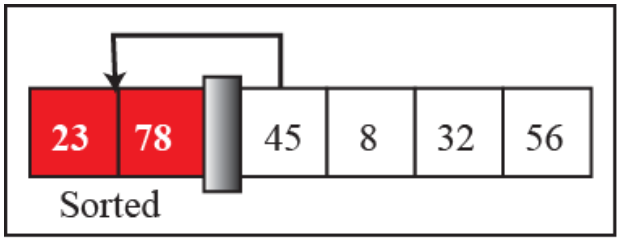
Example of Sorting in Ascending Order

- The number of moves is 15.



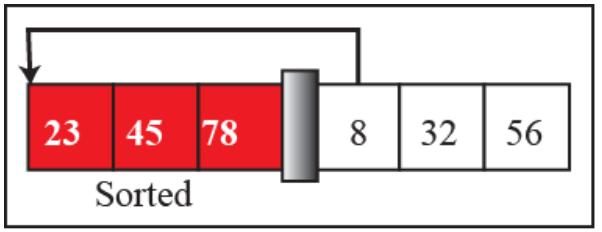
Original list

0 move



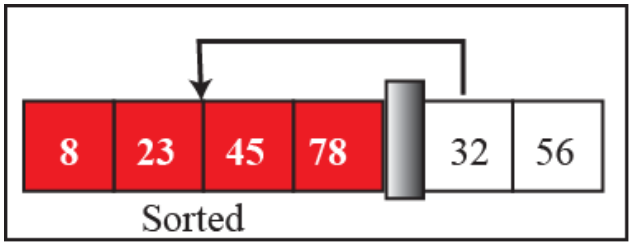
After pass 1

3 moves



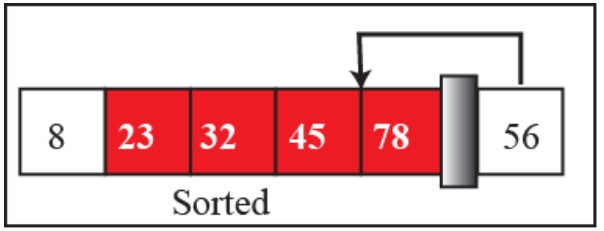
After pass 2

5 moves



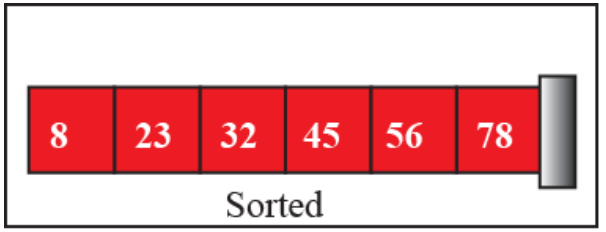
After pass 3

4 moves



After pass 4

3 moves



After pass 5

The third smallest integer is 32. The location of its last occurrence is 4. We need l5 moves to sort the integers.

Don’t use this algorithm for this Lab.

Input

The first line gives the number of test cases. The input of a test case may take several lines which provide a list of integers separated by space characters. The first number in such a line is the number of integers need be sorted in this test case. There are at least two and at most 1024 integers to be sorted.

Output

- **The output of each test case may take several lines. Each line should start with “# “ except the first line and be followed by 10 numbers except the last line. The first line should start with “## “ and the last line should contain not more than 10 numbers.**

Grading

- All outputs are correct, get 100 points.
- TA must ensure the two functions `int findMax(int [], int, int&)` and `void findMin(int [], int, int &, int&);` should be written.

Example of Input & Output

Input	Output
10 7 1 2 3 5 4 -100 100 3 1 2 1 20 1 3 -1 -1 4 201 100 7 9 11 13 15 14 12 10 8 6 4 2 2 2 2 1 14 7 9 10 35 -2 -10 23 45 99 98 101 100001 54 -4 10 -1 -1 -6 -6 1 1 2 1 2 2 24 1 2 3 4 5 6 7 8 9 10 1 2 3 4 5 6 7 8 9 10 -1 -2 -3 -4 20 10 9 8 7 6 5 4 3 2 1 1 2 3 4 5 6 7 8 9 10 15 1 1 3 2 2 3 -1 -1 -2 -3 -3 -2 -1 0 0 20 87 88 22 55 23 44 65 99 78 66 -1 -1 -1 40 103 88 103 87 87 20	## -100 1 2 3 4 5 100 ## 1 1 2 ## -1 -1 1 2 2 3 4 4 6 7 # 8 9 10 11 12 13 14 15 100 201 ## 1 2 ## -10 -4 -2 7 9 10 23 35 45 54 # 98 99 101 100001 ## -6 -6 -1 -1 1 1 1 2 2 2 ## -4 -3 -2 -1 1 1 2 2 3 3 # 4 4 5 5 6 6 7 7 8 8 # 9 9 10 10 ## 1 1 2 2 3 3 4 4 5 5 # 6 6 7 7 8 8 9 9 10 10 ## -3 -3 -2 -2 -1 -1 -1 0 0 1 # 1 2 2 3 3 ## -1 -1 -1 20 22 23 40 44 55 65 # 66 78 87 87 87 88 88 99 103 103