



DASF004: Basic and Practice in Programming

❖ Lab 8: Pointer



In this lab ...

- ❖ Using Pointer Variable
- ❖ What you need to submit in this lab (Lab #8):
 - » Lab Exercise before Wednesday 11:59 pm
 - » Assignment #8 by Tuesday 11:59 pm

Pointer



Pointers are variables that contain *memory addresses* as their values.

Pointer Declaration

```
type* pointer_name;
```

Examples of pointer declarations:

```
int *aPtr;
```

```
float *bPtr;
```

```
char *cPtr;
```

The **asterisk**, when used as above in the declaration, tells the compiler that the variable is a pointer.

Pointer Initialization

You can initialize pointer using the address of a variable:

```
int a = 10;
int *aPtr = &a; // Value of pointer aPtr initialized as the address of variable a
float b = 3.14;
float *bPtr = &b; // Value of pointer bPtr initialized as the address of variable b
char c = "a";
char *cPtr;      // Value of pointer cPtr initialized as the address of variable c
```

Use of & and *

When is & used?

& - "address operator" which gives or produces the memory address of a data variable

- Address of a variable

When is * used?

* - "dereferencing operator" which provides the contents in the memory location specified by a pointer

- Value stored in the address (pointed by the pointer)

Use of & and *

Example (Try this yourself!!):

```
int a = 10;                                // Assign value of a as 10
int * aPtr = &a;                           // Assign value of aPtr as address of a

printf("Value of a: %d\n", a);
printf("Address of a: %d\n", &a);
printf("Value of aPtr: %d\n", aPtr);        // Value of aPtr is address of a
printf("Address of aPtr: %d\n", &aPtr);     // Address of aPtr
printf("Value of the address pointed by aPtr: %d\n", *aPtr);
```

Use of & and *

 C:\Users\Arthur Tang\Documents\Untitled2.exe

```
Value of a: 10
Address of a: 6487628
Value of aPtr: 6487628
Address of aPtr: 6487616
Value of the address pointed by aPtr: 10

-----
Process exited after 0.018 seconds with return value 0
Press any key to continue . . .
```

Address	Value
.....	
6487612	
6487616	aPtr: 6487628
6487620	
6487624	
6487628	a: 10
6487632	
.....	

Pointer and Array

You can have a pointer pointing at an element of an array

```
int a[5] = {1,2,3,4,5};  
int * aPtr = &a[0]; // aPtr pointing at the address of a[0]  
printf("value of the item aPtr pointing at: %d\n", *aPtr);  
aPtr = &a[2];          // aPtr pointing at the address of a[2]  
printf("value of the item aPtr pointing at: %d\n", *aPtr);
```

 C:\Users\Arthur Tang\Documents\Untitled2.exe

value of the item aPtr pointing at: 1

value of the item aPtr pointing at: 3

Process exited after 0.0122 seconds with return value 0
Press any key to continue . . .

Pointer Operation

Moving the pointer along the array

```
double a[3] = {1.0,2.0,3.0};  
double * aPtr = &a[0]; // aPtr pointing at the address of a[0]  
printf("value of the item aPtr pointing at: %f\n", *aPtr);  
aPtr++;           // aPtr pointing to the next memory location  
printf("value of the item aPtr pointing at: %f\n", *aPtr);  
aPtr++;           // aPtr pointing to the next memory location  
printf("value of the item aPtr pointing at: %f\n", *aPtr);  
aPtr--;           // aPtr pointing to the previous memory location  
printf("value of the item aPtr pointing at: %f\n", *aPtr);  
aPtr--;           // aPtr pointing to the previous memory location  
printf("value of the item aPtr pointing at: %f\n", *aPtr);
```

C:\Users\Arthur Tang\Documents\Untitled2.exe

```
value of the item aPtr pointing at: 1.000000  
value of the item aPtr pointing at: 2.000000  
value of the item aPtr pointing at: 3.000000  
value of the item aPtr pointing at: 2.000000  
value of the item aPtr pointing at: 1.000000
```

```
-----  
Process exited after 0.01968 seconds with return value 0  
Press any key to continue . . .
```

Exercise: Pointer Operation

Moving the pointer along the array

Declare an int array as follow:

```
int a[10] = {2,5,8,1,4,7,3,10,6,9};
```

Declare two int pointers as follow:

```
int * MaxPtr, *MinPtr;
```

Initialize both pointers to the first item of the array as follow:

```
MaxPtr = &a[0];    MinPtr = &a[0];
```

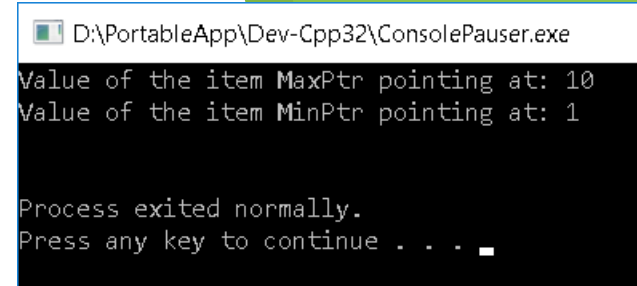
Finish the program to scan the array for the max value and min value **using pointer operations**.

At the end of the program, print out the value of MaxPtr and MinPtr pointing to

```
printf("Value of the item MaxPtr pointing at: %d\n", *MaxPtr);
```

```
printf("Value of the item MinPtr pointing at: %d\n", *MinPtr);
```

Your program should product the following result:



```
D:\PortableApp\Dev-Cpp32\ConsolePauser.exe
Value of the item MaxPtr pointing at: 10
Value of the item MinPtr pointing at: 1

Process exited normally.
Press any key to continue . . . _
```

The sizeof () function

sizeof() is a very useful function in C

It returns the size (in terms of number of byte) of a variable

e.g. `int x = 10;`

`double y = 20.0;`

`int z[5] = {1,2,3,4,5};`

`printf("size of x: %d\n",sizeof(x)); // size of int is 4 byte`

`printf("size of y: %d\n",sizeof(y)); // size of double is 8 byte`

`printf("size of z: %d\n",sizeof(z)); // size of the array is 20 byte`

You can also use it to count the number of item in an array:

`printf("# of item in z: %d\n",sizeof(z)/sizeof(int)); // 5 items!!!`

Function with Pointer(s) As Argument(s)

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

```
int ArrayTotal(int * A, int size)
{ int total = 0;
  for(int i=0;i<size;i++)
    total += A[i];
  return total;
}
```

```
int main(void)
{ int A[5] = {1,2,3,4,5};
  int sum = ArrayTotal(A,5);
  printf("Sum: %d",sum);
  return 0;
}
```



Function with Pointer(s) As Argument(s)

When passing pointer(s) as functions' arguments, it is passing by reference

i.e. If the functions modify the values of the items pointed by the pointers, the modification will be reflected in the function calling it

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

```
void ArrayFunction(int *A, int size)
```

```
{ int total = 0;
```

```
  for(int i=0;i<size;i++)
```

```
    A[i]++;                                // Add one to each array item
```

```
}
```

```
int main(void)
```

```
{ int A[5] = {1,2,3,4,5};
```

```
  ArrayFunction(A,5);
```

```
  printf("{%d,%d,%d,%d,%d}",A[0],A[1],A[2],A[3],A[4]);
```

```
  return 0;
```

Lab Assignment #8: Pointer

1. You are given a source code as shown on the next page. Copy and paste the code into Dev C++.
2. In the main function, 4 testing `int` arrays was declared.
3. The number of items in each array were printed, as well as each individual items in the array.
4. A function named `ArrayReverse()` was then called using the 4 testing arrays. Note that the `ArrayReverse()` function is empty now. **You need to write this function.**
5. After the function call, the number of items in each array were printed again, as well as each individual items in the array.
6. The `ArrayReverse()` function:
The function should reverse the item in the `int` array. For example, the array `{1,2,3}` will become `{3,2,1}` after the function call. Note that the address of the array is passed (passing by reference). If you modify the array in the function, it will be reflected in the main function.
7. A sample output is provide. Your program should generate the exact output as shown in the sample output.

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

```
void ArrayReverse(int * InputPtr,int size)
{ // You need to write this function!!!
}
```

```
int main(void)
{ int Test1[10] = {10,20,30,40,50,60,70,80,90,100};
  int Test2[6] = {112,110,108,106,14,12};
  int Test3[7] = {18,35,1024,23,68,51,51};

  printf("Test1 contains %d items: ",sizeof(Test1)/sizeof(int));
  for(int i=0;i<sizeof(Test1)/sizeof(int);i++)
    printf("%d ",Test1[i]);
  printf("\n");          // Print Array Test1

  printf("Test2 contains %d items: ",sizeof(Test2)/sizeof(int));
  for(int i=0;i<sizeof(Test2)/sizeof(int);i++)
    printf("%d ",Test2[i]);
  printf("\n");          // Print Array Test2

  printf("Test3 contains %d items: ",sizeof(Test3)/sizeof(int));
  for(int i=0;i<sizeof(Test3)/sizeof(int);i++)
    printf("%d ",Test3[i]);
  printf("\n");          // Print Array Test3
```

```
ArrayReverse(Test1,sizeof(Test1)/sizeof(int));
ArrayReverse(Test2,sizeof(Test2)/sizeof(int));
ArrayReverse(Test3,sizeof(Test3)/sizeof(int));

printf("Test1 after Reverse: ");
for(int i=0;i<sizeof(Test1)/sizeof(int);i++)
  printf("%d ",Test1[i]);
printf("\n");          // Print Array Test1

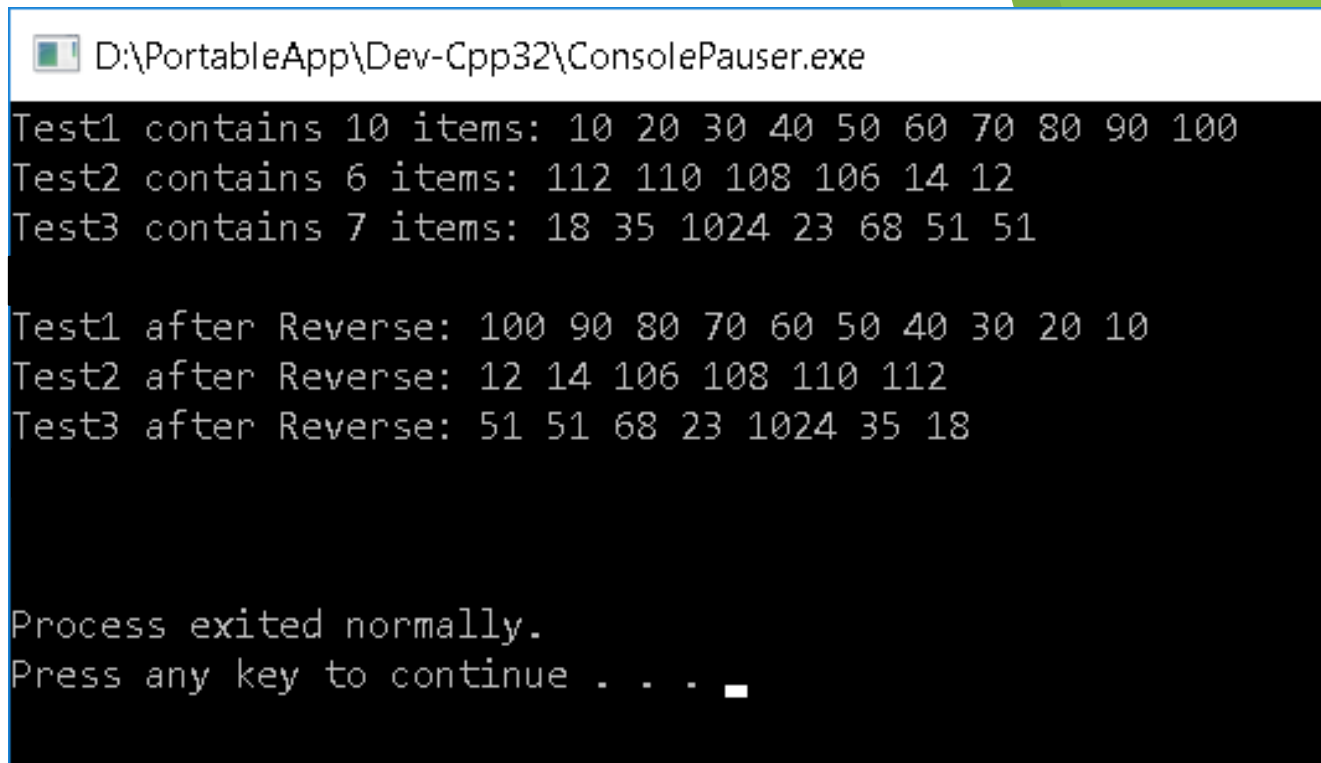
printf("Test2 after Reverse: ");
for(int i=0;i<sizeof(Test2)/sizeof(int);i++)
  printf("%d ",Test2[i]);
printf("\n");          // Print Array Test2

printf("Test3 after Reverse: ");
for(int i=0;i<sizeof(Test3)/sizeof(int);i++)
  printf("%d ",Test3[i]);
printf("\n");          // Print Array Test3

return 0;
}
```


Lab Assignment #8: Pointer

Sample outputs:



```
D:\PortableApp\Dev-Cpp32\ConsolePauser.exe
Test1 contains 10 items: 10 20 30 40 50 60 70 80 90 100
Test2 contains 6 items: 112 110 108 106 14 12
Test3 contains 7 items: 18 35 1024 23 68 51 51

Test1 after Reverse: 100 90 80 70 60 50 40 30 20 10
Test2 after Reverse: 12 14 106 108 110 112
Test3 after Reverse: 51 51 68 23 1024 35 18

Process exited normally.
Press any key to continue . . .
```

Lab Assignment #8: Pointer

Submit your source code on iCampus before Tuesday 11:59 pm