Exercises Pointer

**A. Lý thuyết**

What is a Pointer? 
• (1) A pointer is a variable, which contains the address of 
a memory location of another variable 
• (2) If one variable contains the address of another 
variable, the first variable is said to point to the 
second variable 
• (3) A pointer provides an indirect method of accessing the 
value of a data item 
• (4) Pointers can point to variables of other fundamental 
data types like int, char, or double or data aggregates 
like arrays or structures 

What are Pointers used for? 
Some situations where pointers can be used are 
• To return more than one value from a function 
• To pass arrays and strings more conveniently from 
one function to another 
• To manipulate arrays easily by moving pointers to them 
instead of moving the arrays itself 
• To allocate memory and access it 
(Direct Memory Allocation) 

Variable characteristics 
type name 
int var 
- Source code: 
value 
access 
o 
var) 
value: %d \n", 
address: %d \n", &var); 
Memory space 
(1 block = 1 byte, 32 bit OS) 
Address Memory blocks 
I 
2 
3 
4 
5 
6 
7 
8 
9 
le 
*include < stdio.h> 
int main(){ 
int var 
printf( "var' s 
printf( "var's 
return e; 
48758 
6487581 
6487582 
6487583 
var 
- Result: 
var's value: 18 
var's address: 6487588 

Pointer Variables 
General declaration syntax is : 
datatype *name ; 
Example: 
int var 
int *ptr var ; 
6487580 
6487581 
6487582 
6487583 
6487584 
6487585 
NULL 
6487586 
6487587 
var 
ptr_var 

Pointer Operators - 1 
• There are 2 special unary operators which are used with 
pointers 
& and * 
• The & operator used to 
returns the memory 
address of the operand 
Example: 
int var 
int *ptr 
ptr var 
6487580 
6487581 
6487582 
6487583 
6487584 
6487585 
6487586 
6487587 
6487580 
var 
ptr_var 
var ; 
= &var; 

Pointer Operators - 2 
• The operator used to returns the value contained in the 
memory location pointed to by the pointer variable's value 
temp = *ptr_var; 
Example: 
int var = 10, temp; 
int *ptr var = &var; 
temp = V ar ; 
temp 
= 10; 

Assigning Values To Pointers-I 
• Values can be assigned to pointers through the & operator 
ptr var = 
&var ; 
• Here the address of var is stored in the variable ptr_var 
• It is also possible to assign values to pointers through another 
pointer variable pointing to a data item of the same data type 
ptr var = 
&var ; 
ptr var2 = ptr var; 

Assigning Values To Pointers-2 
• Variables can be assigned values through their pointers as 
well 
*ptr_var = 10; 
• The above declaration will assign 10 to the variable var if 
ptr_var points to var 

#include <stdio.h> 
int main( 
int var= 10; 
int *ptr_var; 
ptr_var = &var; 
printf("var's value: %d \n", var); 
printf("var's address: %d \n", &var); 
printf("ptr_var's value: %d \n", ptr_var) 
Pointer Example - 1 
var's value: 10 
var's address: 6487572 
ptr_var's value: 6487572 
Value in memory pointed by ptr_var: 
var's value after change: 20 
18 
Address Memory blocks 
6487572 
printf("Value in memory pointed by ptr_var: 
%d \n", *ptr_var); 
6487576 
*ptr_var = 20; 
printf("var's value after change: %d" var) 
o; 
return 
NULL 
var 
ptr_var 

Pointer Example - 2 
100 
3 
4 
le 
11 
12 
13 
15 — 
16 
17 
18 
19 
2e 
21 
*include <stdio.h> 
void call_by_value(int n); 
void call by reference(int 
n' s value after call_by_value: 8 
n •s value after call by reference: 
6— int main(){ 
int n = e 
call_by_value(n); 
printf("n's value 
call by reference(&n); 
printf("n's value 
return e; 
call_by_value(int 
= lee; 
call _ by 
reference 
*n = lee; 
after 
after 
int 
*n); 
call_by_value: %d 
call _ by _ reference: 
main 
void 
void 
call_by_value 
=100 
call _ by _ reference 
&n 
=100 

NOTE - 1 
• A pointer provides an indirect method of accessing the 
value of a data item 
*ptr_var 
direct 
access 
&var 
indirect access 
var 
direct 
access 
10 

NOTE -2 
• The size of the pointer variable is not related to the type 
of variable it points to, because it depends on types the 
operating system 
1 
2 
3 
4 
5 
6 
7 
8 
9 
le 
11 
12 
13 
14 
15 
16 
#include <stdio.h> 
int maxn 
char 
int 
double 
char 
int 
double 
printf(' 
Result: 
Size of c, n, d: 1 byte, 4 byte, 8 byte 
Size of pc, pn, pd: 8 byte, 8 byte, 8 byte 
*pc = &c; 
'Size of c, n, d: %d byte, %d byte, %d byte \n", 
sizeof(c), 
sizeof(n), 
sizeof(d)); 
printf( "Size of pc, pn, pd: %d byte, %d byte, %d byte \n", 
sizeof(pc 
sizeof(pn), 
sizeof(pd)); 
return e; 

Pointer Arithmetic-I 
• Addition and subtraction are the only operations that 
can be performed on pointers 
int var, 
ptr var 
& var; 
— 500; 
var 
• Let us assume that var is stored at the address 
1000 
Then ptr_var has the value 1000 stored in it. Since integers are 2 
bytes long, after the expression "ptr_var++;" ptr_var will have the 
value as 1002 and not 1001 

Example — Pointer Arithmetic 
1 
le 
11 
12 
13 
14 
15 
#include <stdio.h> 
pd 
pd 
pd 
with 
with 
with 
pc, 
pc, 
pc, 
+1: 
pn, 
int main(){ 
char 
int 
double 
char 
int 
double 
pd with +8: 6487559, 6487552, 6487544 
pd with +1: 6487560, 6487556, 6487552 
pd with +0: 6487561, 6487560, 6487560 
*pc 
*pn 
printf("\n\npc, pn, 
printf("\n\npc, pn, 
printf("\n\npc, pn, 
return e; 
%d, 
%d, 
pc, pn, pd) 
pc+l, pn+l, pd+l); 
pc+2, pn+2, pd+2); 

Pointer Arithmetic - 2 
++ptr var or ptr var++ 
—ptr_var or ptr_var— 
ptr var+i 
ptr_var - i 
pomts to next integer after var 
pomts to integer previous to var 
pomts to the ith integer after var 
pomts to the ith integer before var 
++*ptr var or (*ptr_var)++ will Increment vu by 1 
*ptr var++ 
Will fetch the value of the next Integer after var 
• Each time a pointer is incremented, it points to the 
memory location of the next element of its base type 
• Each time it is decremented it points to the location of the 
previous element 
All other pointers will increase or decrease depending on 
the length of the data type they are pointing to 

Pointer Comparisons 
Two pointers can be compared in a relational expression provided both 
the pointers are pointing to variables of the same type 
Consider that ptr_a and ptr_b are 2 pointer variables, which point to 
data elements a and b. In this case the following comparisons are 
possible: 
ptr_a < ptr_b 
ptr_a > ptr b 
a >= ptr_b 
ptr_a = ptr_b 
a ptr_b 
a — NULL 
Returns hue provided a is stored before b 
Returns hue provided a is stored after b 
Returns hue provided a is stored before b or ptr_a and ptr_b point to the sarne 
location 
Returns hue provided a is stored after b or ptr_a and ptr_b point to the satne 
location. 
Returns true provided both pomters ptr_a and ptr_b points to the satne data 
element. 
Retums hue provided both pointers ptr_a and ptr_b pomt to different data 
elements but of the sarne Bype. 
Returns hue if ptr_ais assigned NULL value (zero) 

Pointer and one-dimensional array 
Source code: 
1 *include <stdio.h> 
2 int main(){ 
Result: 
value of arr: 6487552 
address of arr[Ø]: 
6487552 
3 
4 
5 
6 
7 
8 
int arr[5] ¯ 
printf( "value of arr: %d \n", 
arr) 
printf("address of arr[e]: %d\n", 
return e; 
9 
Source code: 
11 
12 
13 
14 
15 
16 
17 
18 
19 
2e 
21 
*include <stdio.h> 
int main(){ 
int arr[5] = 
int *p arr = arr 
int i; 
// or : 
- &arrlel 
*p arr — 
p_arr[%d]= %d \n" , i, 
Result: 
( arr+€) 
* (arr+i), 
6, 
5, 
6, 
7, 
8, 
p_arr[l]= 
p_arr[2]= 
p_arr[3]= 
6 
6 
7 
8 
for(i=e; i<5; 
printf( (arr+%d) = 
return e; 
int 
%d, 
p_arr[i]); 

Pointer and one-dimensional array 
• In C language, array names can be thought of as a constant 
pointer type 
• The address of the array matches the first element address 
of the array 
• Use unary operators (++, --) to move between elements in 
the array pointed to by the pointer 

Pointer and one-dimensional array 
#include 
void input_array(int *p, int size); 
void output_array(int *p, int size); 
int add_array(int *p, int size); 
7— int 
int scores[20], n; 
printf("Enter the number of Element in scores array: 
1 
2 
3 
4 
5 
6 
8 
9 
10 
11 
12 
13 
15 
16 
17 
18 
scanf("%d% an); 
input _ array (scores, n) ; 
output _ array (scores, n) ; 
printf("\nTota1 value of the elements: 
return 0; 
add _ array (scores, n)); 
19B void input array(int 
27 
28B void output array(int 
35 
368 int add array(int *p, 
*p, int size){ 
*p, int size){ 
int size){ 

Pointer variables type Struct 
*include <stdio.h> 
struct Student{ 
1 
3 
4 
6 
8 
9 
le 
11 
12 
13 
15 
int 
char id[le]; 
char name[4e]; 
main(){ 
= {"sTee1", 
struct Student st 
struct Student *pst; 
pst = &st; 
printf("ID: #%s I Name: 
printf( "ID: #%s I Name: 
return e; 
*pst 
"Pham Ngoc 
st.id, st. 
Tho" 
name 
%s\n " , 
st 
pst->id, pst->name) 
Student 
id 
name 

**B. Bài tập**

**Ex1.**

Write a short C program that declares and initializes (to any value you like) a double, an int, and a char. Next declare and initialize a pointer to each of the three variables. Your program should then print the address of, and value stored in, and the memory size (in bytes) of each of the six variables.

Use the “0x%x” formatting specifier to print addresses in hexadecimal. You should see addresses that look something like this: "0xbfe55918". The initial characters "0x" tell you that hexadecimal notation is being used; the remainder of the digits give the address itself.

Use “%f” to print a floating value. Use the memory size allocated for each variable.

sizeof operator to determine the

-----

**Sample output**:

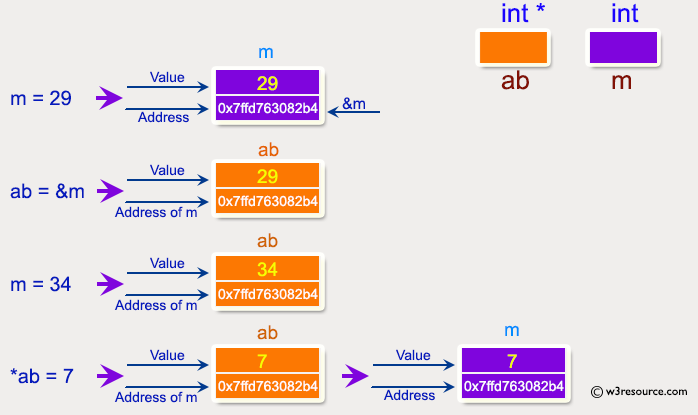
The 
The 
The 
The 
The 
The 
The 
The 
The 
The 
The 
The 
The 
The 
The 
The 
The 
The 
address 
address 
address 
address 
address 
address 
of 
of 
of 
of 
of 
of 
char 
int 
double 
char* 
int* 
double * 
value 
value 
value 
value 
value 
value 
size 
size 
size 
size 
size 
size 
of 
of 
of 
of 
of 
of 
of 
of 
of 
of 
of 
of 
char 
int 
double 
char* 
int* 
double* 
char 
int 
double 
char* 
int* 
double* 
is 
is 
is 
is 
is 
is 
is 
is 
is 
is 
is 
is 
Ox 
Ox 
Ox 
Ox 
Ox 
Ox 
Ox 
Ox 
Ox 
is 
is 
is 
is 
is 
is 
bytes 
bytes 
bytes 
bytes 
bytes 
bytes 

**EX2.**

Write a program in C to demonstrate how to handle the pointers in the program

-----

**Pictorial Presentation:**



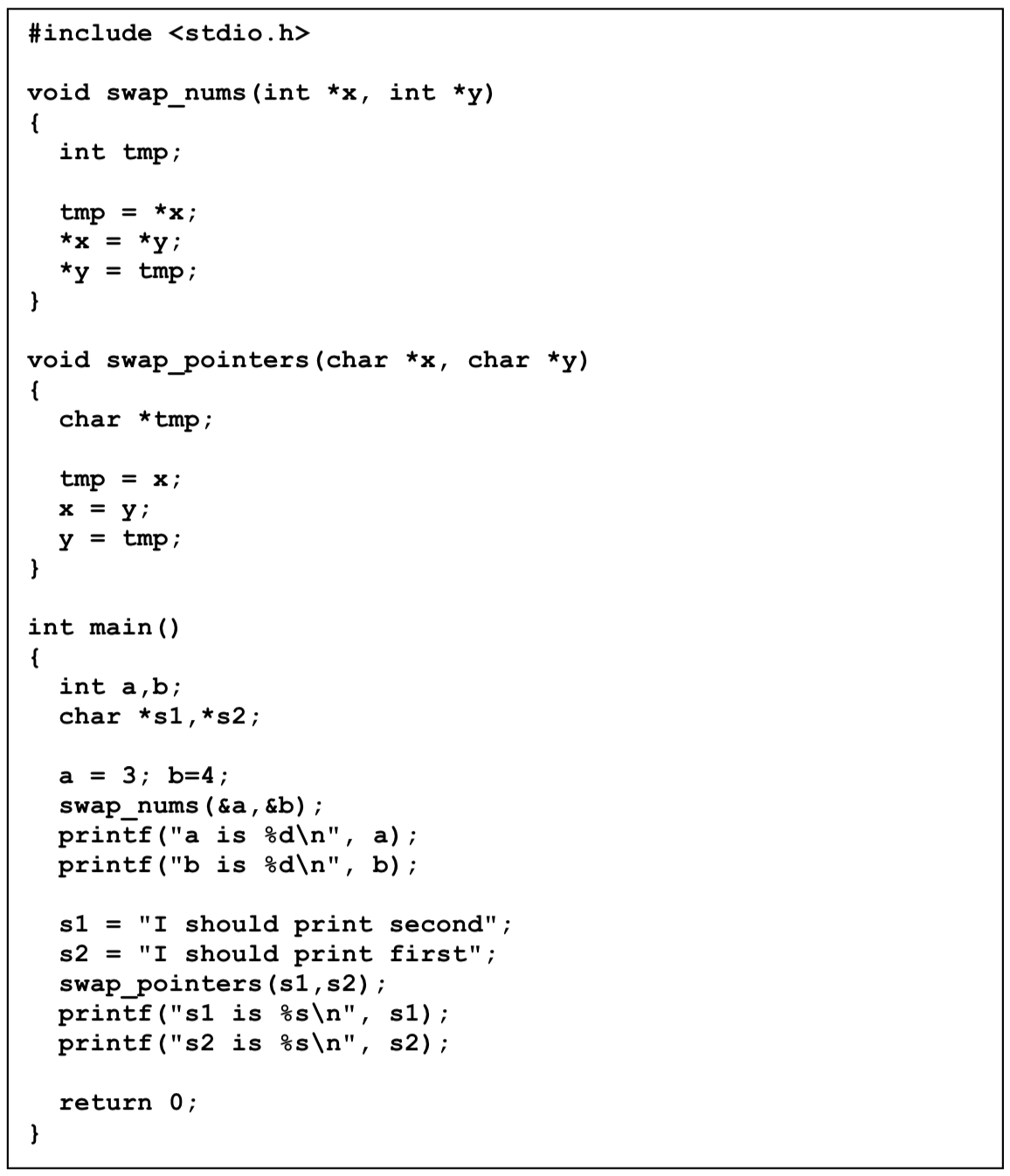
-----

**Expected output**:

Address of m : 0x7ffcc3ad291c  
 Value of m : 29   
   
 Now ab is assigned with the address of m.   
 Address of pointer ab : 0x7ffcc3ad291c   
 Content of pointer ab : 29   
   
 The value of m assigned to 34 now.   
 Address of pointer ab : 0x7ffcc3ad291c   
 Content of pointer ab : 34   
   
 The pointer variable ab is assigned with the value 7 now.   
 Address of m : 0x7ffcc3ad291c   
 Value of m : 7

**Ex3.**

swap\_nums seems to work, but not swap\_pointers. Fix it.

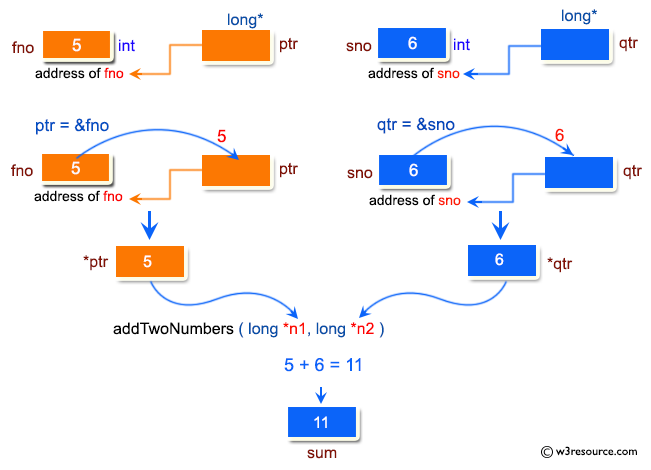


**Ex4.**

Write a program in C to add numbers using call by reference

-----

**Pictorial Presentation:**



-----

**Sample output**:

Pointer : Add two numbers using call by reference:   
-------------------------------------------------------   
 Input the first number : 5   
 Input the second number : 6   
 The sum of 5 and 6 is 11

**EX5.**

Write a program, use the pointer to enter a sequence of integers consist N element. Ouput to screen:

• Maximum value of the element in array

• Minimum value of the element in array

• Sum the elements in array