CS 4375/5340 Fall 2023

Shirley Moore, Instructor

August 29 Class

**C Pointer Exercises Solutions**

Question 1

Consider the following declarations and assignments.

int \*a, b[5], c, \*d[5];

for(c=0; c<5; c++) {

b[c] = 1+c;

}

\*d = b;

a = &c;

c = (\*d)[3];

What are the TYPE and VALUE of each of the following expressions after execution of the assignment statements? (if the expression is invalid, write “Illegal Expression”, and if it is an address describe what it is the address of).

TYPE VALUE

1. a pointer to an int address of c
2. \*a int 4
3. b pointer to an int address of b[0]
4. \*b int 1
5. c int 4
6. &b[1] pointer to an int address of b[1]
7. d pointer to a pointer to an int. address of d[0]
8. \*d pointer to an int address of b[0]
9. \*\*d int 1
10. \*\*b Illegal Expression

See <https://www.geeksforgeeks.org/array-of-pointers-in-c/> for help with \*d[5].

Question 2

What would be the output from the program below? Try to figure out the answer without running the program and then run the program to check your answer.

#include <stdio.h>

#include <stdlib.h>

int \*fun(int \*a, int \*b, int s);

int main() {

int \*arr = NULL, x = 6, y = 7, i;

arr = fun(&x, &y, 5);

printf("x = %d y = %d\n", x, y);

if (arr != NULL) {

for (i = 0; i < 5; i++) {

printf("arr[%d] = %d\n", i, arr[i]);

}

free(arr);

}

}

int \*fun(int \*a, int\*b, int s) {

int \*tmp, i;

tmp = malloc(sizeof(int) \* s);

if (tmp != NULL) {

for (i = 0; i < s; i++) {

tmp[i] = i + \*b;

}

\*a = tmp[2];

\*b = 8;

}

return tmp;

}

Output:

x = 9 y = 8

arr[0] = 7

arr[1] = 8

arr[2] = 9

arr[3] = 10

arr[4] = 11

Question 3

3.1. What would be the output from the program below? Try to figure out the answer without running the program and then run the program to check your answer.

array element 0: 45

array element 1: 67

array element 2: 89

array\_ptr[1] = 89

3.2. What do you think you will see if you use change array\_ptr[1] to array\_ptr[2] in the last line? Explain.

A reference to array\_ptr[2] in the last line would be out of bounds for the allocated storage for the array. This could access some other location in the user’s address space, which would be allowed, or could access some location outside the user’s address space resulting in a segmentation fault error.

#include <stdio.h>

int main() {

int array[] = {45, 67, 89};

int \*array\_ptr = array;

int i;

for (i = 0; i < 3; i++) {

printf("array element %d: %d\n", i, \*array\_ptr);

array\_ptr++;

}

array\_ptr = &array[1];

printf("array\_ptr[1] = %d\n", array\_ptr[1]);

}

Questions 4 and 5 refer to the xv6 code at <https://github.com/mit-pdos/xv6-riscv> .

Question 4. Refer to the definitions of struct proc and the proc array in files kernel/proc.h and kernel/proc.c, respectively.

1. Write a C statement that assigns the pid of the parent of proc[4] to the integer variable ppid.

ppid = (proc[4]->parent).pid;

1. Assuming that the process with child\_pid has been orphaned – i.e., its parent has died, write a C statement that changes its parent to its parent’s parent.

Assume child\_proc points to the proc table entry with pid=child\_pid.

Assume the dead parent is a zombie process – i.e., its proc table entry still exists.

child\_proc->parent = (child\_proc->parent)->parent;

1. Write a loop that prints the pid of each process along the parent chain from mypid to the init process, assuming the init process does not have a parent.

Assume my\_proc points to the proc table entry with pid = mypid.

proc\_ptr = my\_proc;

while (proc\_ptr != NULL) {

printf(“%d\n”, proc\_ptr->pid);

proc\_ptr = proc\_ptr->parent;

}

Question 5. In writing OS code, we often use queues of processes that are waiting on something, for example the ready queue of processes that are waiting to be scheduled to run. Design a process queue data structure with a head and a tail and that uses a new pointer field in struct proc to point to the next process in the queue. Write the code for enqueue that adds a process to the tail of the queue and for dequeue that removes a process from the head of the queue.

#include <stddef.h>

//Add a field called next to struct proc

struct proc {

int pid;

struct proc \*next;

};

struct queue {

struct proc \*head;

struct proc \*tail;

};

int queue\_init(struct queue \*q) {

q->head = NULL;

q->tail = NULL;

return(0);

};

int enqueue(struct queue \*q, struct proc \*p) {

p->next = NULL;

if (q->tail == NULL) { // empty queue

q->head = p;

q->tail = p;

} else {

q->tail->next = p;

q->tail = p;

}

return(0);

}

/\* returns pointer to proc removed from queue,

\* or NULL in the case of an empty queue

\*/

struct proc \*dequeue(struct queue \*q) {

struct proc \*p;

if (q->head == NULL) // empty queue

return(NULL);

p = q->head;

q->head = q->head->next;

return(p);

}