Question 1 (1 point) ✓ *Saved*

Variables declared in a C program are stored on the call stack.

- ◯ True
- ◯ False

Question 2 (1 point) ✓ *Saved*

Storage space allocated on the heap is referenced with pointers.

- ◯ True
- ◯ False

Question 3 (1 point) ✓ *Saved*

The fundamental difference between stack and heap memory is:

- ◯ the allocated heap memory is available until it is freed.
- ◯ the allocated stack memory is no longer available once the function ends (returns).
- ◯ the heap memory locations are always referenced with pointers
- ◯ all of the above

Question 4 (1 point) ✓ *Saved*

In a typical make file, the decision which files need to be recompiled is based on:

- ◯ file name
- ◯ file size
- ◯ file modification time
- ◯ file location

Question 5 (3 points) ✓ *Saved*

Consider the following program:

```
int g2(char * c) {
   if (*c == '\0') return 0;
   return 1 + g2(c+1);
}
int main(int argc, char * * argv){
   char s[ ] = "abc";
   g2(s);
}
```

What would be the largest number of frames stored on the memory stack at any stage of the execution of this program? Enter your answer as a number in the box below.

Your Answer:

4

Answer

Question 6 (3 points) ✓ *Saved*

Consider the following C program that prints the contents of an array of integers:

```
int main()
{
    int a[] = {0,0,0,0,0};
    for (int i=0; i<5; i++)
       printf("%d", *(&a[i]+a[i]));

    return 0;
}
```

This program, as is, outputs the array elements in the storage order, i.e from the first array element to the last.

What contents of this array would result in printing the array elements in the REVERSED storage order, i.e. from the last array element to the first?

(Note: you are asked to show what values the array a[] = {?, ?, ?, ?, ?} needs to be initialized with so that, when the for loop is executed, the elements of the array will be printed in the reversed order, i.e. starting with the last element of the array and finishing with the first element.)

Enter your answer in the box below strictly following this formatting: enter integer values separated by commas. Do not insert any spaces between any characters! For example, if your answer was `int a[] = {1, 2, -3, 4, 5}` then you would enter in the box below (with NO SPACES!):
1,2,-3,4,5

0,0,0,0,0

Question 7 (6 points) ✓ *Saved*

Write a C function called `duplicateArgv()` that copies all command line arguments passed to `main` into a new 2D array allocated entirely on the heap. This function has two parameters, same as in the `main()` function: integer `argc` indicating how many command line arguments are stored in the array `argv`, and the array of strings `argv` (with each string representing one command line argument). This function should return (a pointer to) a new 2D array allocated on the heap storing a duplicate copy of the entire `argv`.

For example, this sample `main()` function with a call to `duplicateArgv()`:

```
int main(int argc, char * * argv) {
    char * * argv2;
    argv2 = duplicateArgv(argc, argv);
    printf("%s\n", argv[0]);
    printf("%s\n", argv2[0]);
}
```

should print the first string from the command line (the name of the program) twice: from the original `argv` on the stack and from the duplicated `argv2` on the heap.

Type your program in the box below.

```
char * * duplicateArgv (int argc, char  * * argv ) {
```

```
    copy[i] = (char *)malloc(strlen(argv[i]) + 1);
    if (copy[i] == NULL) {
        fprintf(stderr, "Memory allocation failed.\n");
        while (i > 0) {
            free(copy[--i]);
        }
        free(copy);
        exit(1);
    }
    strcpy(copy[i], argv[i]);
    i++;
  }

  return copy;
}
```

Question 8 (6 points) ✓ *Saved*

Write a C function called `linkedListArgv()` that copies all command line arguments passed to `main()` into a linked list. This function has two parameters, same as in the `main()` function: integer `argc` indicating how many command line arguments are stored in the array `argv`, and the array of strings `argv` (with each string representing one command line argument). This function should return a linked list (a pointer to the head, first node) with each node storing one command

line argument form `argv`.

For example, this sample `main()` function with a call to `linkedListArgv()`:

```
typedef struct listnode {
    struct listnode * next;
    char * data;
} Node;

int main(int argc, char * * argv) {
    Node * argvList;
    argvList = linkedListArgv(argc, argv);
    printf("%s\n", argv[0]);
    printf("%s\n", argvList->data);
}
```

should print the first string from the command line (the name of the program) twice: from the original `argv` on the stack and from the first node of the linked list `argvList`.

Type your program in the box below.

```
Node * linkedListArgv (int argc, char  * * argv ) {
```

```
        return NULL;
    }

    Node *head = genNode(argv[0]);
    Node *current = head;
    int i = 1;

    while (i < argc) {
        current->next = genNode(argv[i]);
        current = current->next;
        i++;
    }

    return head;
}
```

Question 9 (1 point) ✓ *Saved*

Consider the two data structures (2D array and linked list) described in two previous questions (Question 7 and Question 8) and proposed for storing a copy of the command line arguments. Which of these two representations would require a larger memory allocation (in total)?

- ◯ 2D array
- ◯ linked list

Question 10 (0.01 points) ✓ *Saved*

THIS IS NOT A QUESTION

If you have any comments you may write them in the text box below:

?

Thank you! it was nice taking course with you :)

Submit Quiz *10 of 10 questions saved*