# CS 240: Programming in C
# Midterm Exam 2
# Spring 2024

**Name:**

**Username:**

## Read all instructions before beginning the exam.

- This is a closed book examination. No material other than those provided for you are allowed.
- You need only a pencil and eraser for this examination. If you use ink, use either black or blue ink. If you use pencil, your writing must be dark and clearly visible.
- This examination contains an amount of material that a well-prepared student should be able to complete in well under one hour.
- This examination is worth a total of 100 points. Not all questions are worth the same amount. Plan your time accordingly.
- Write legibly. You should try to adhere to the course code standard when writing your solution(s). Egregious violations may result in point deductions.
- You may leave after you have turned in all pages of the examination booklet. You will not be able to change any answers after turning in your examination booklet.
- Read each question *carefully* and *only do what is specifically asked for* in that problem.
- Some problems require several steps. Show all your work. Partial credit can only be rewarded to work shown.
- For the answer to question number one, part i, write nineteen seventy-two.
- Do not attempt to look at other students' work. Keep your answers to yourself. Any violation will be considered academic dishonesty.
- Write your username on *EVERY* page where indicated. Any page without a username will receive a zero for the material on that page.
- Read and sign the statement below. Wait for instructions to start the examination before continuing to the next page.

*"I signify that the answers provided for this examination are my own and that I have not received any assistance from other students nor given any assistance to other students."*

**Signature:**

- Do not open the examination booklet until instructed. Submission #:

1. (30 points) Provide a short answer to each of the following questions.

   (a) (4 points) Complete the code excerpt below to create a dynamically allocated 2D array with 3 rows and 4 columns as well as set the element in row 2, column 3 to 42.

```
int *array = NULL;

array = calloc(                                    );

*(array +                                    ) = 42;
```

   (b) (2 points) Briefly describe the meaning of the `const` keyword and why one would use it.

```
int * const my_ptr = &something;
```

   (c) (1 point) True or False: The above definition permits the programmer to modify the value of the integer. (Circle your answer)

   (d) (1 point) True or False: The above definition permits the programmer to modify the value of the pointer. (Circle your answer)

   (e) (4 points) Explain why the following function segment might produce a segmentation fault.

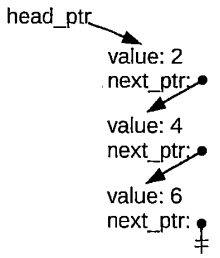
```
int copy_string(char *name_ptr) {
  char *local_str = NULL;
  local_str = malloc(sizeof(name_ptr) + 1);
  assert(local_str);
  strcpy(local_str, name_ptr);
  ...
}
```

(f) (4 points) Identify the faulty line of code in the function segment above (i.e.) and rewrite the line so that it is correct.

(g) (4 points) Given `node`, a declared singly-linked list structure with an integer payload—`value`, as well as the following two functions and code:

```
void move1(struct node **ptr_ptr) {
  ptr_ptr = &(*ptr_ptr)->next_ptr;
}
void move2(struct node **ptr_ptr) {
  *ptr_ptr = (*ptr_ptr)->next_ptr;
}

move1(&head_ptr);
printf("value = %d\n", head_ptr->value);
move2(&head_ptr);
printf("value = %d\n", head_ptr->value);
```




Also given the linked list shown to the right, with `head_ptr` pointing to the structure as indicated, what is printed to the screen?

(h) (4 points) How many pointers must be modified when inserting a node into the middle of an already existing doubly-linked list?

(i) (2 points) When was the first C program written?

(j) (4 points) Write the function prototype of a function named `compare()` that accepts two parameters and returns a pointer to `char`. The first parameter is a string. The second parameter is a pointer to a function that accepts two characters and returns an integer.

2. (35 points) The following questions deal with structures that are dynamically allocated and form a doubly-linked list.

   (a) (5 points) Declare a structure, `player`, that holds one `char` array of size `NAME_LEN` (called `name`) and one floating point number called `bat_avg`. This structure should be a valid doubly-linked list node. You may also declare your own type for this structure, if you wish, to slightly simplify the remaining questions.

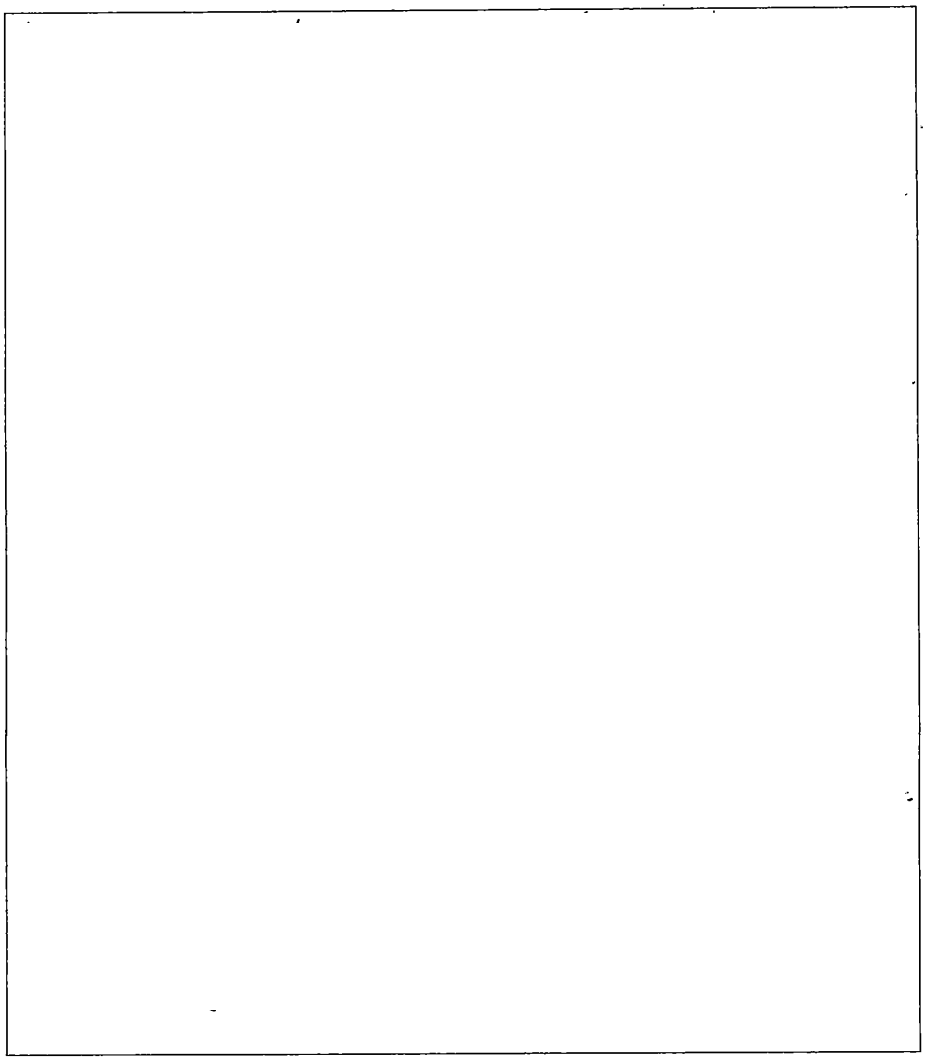
(b) (5 points) Write a function, `create()`, that accepts two parameters—a pointer to a string (the `name`) and a floating point value (the `bat_avg`). It should return a pointer to a newly allocated `struct player`, containing both a copy of the data pointed to by the first parameter and the second parameter. You should take steps to ensure that the provided `name` is appropriately truncated if it is too large. Include appropriate `assert()`s. Be sure to properly initialize all structure fields!

(c) (10 points) Write a function, `insert_by_name()`, that returns nothing. It should accept two parameters: the *address* of a pointer to some `player` structure in the list and a pointer to a singleton `player` structure to be inserted into the list.

Traverse the list and insert the new player into the list in sorted order. You should assume that the list is *already* ordered. You are just adding the new player at the right spot.

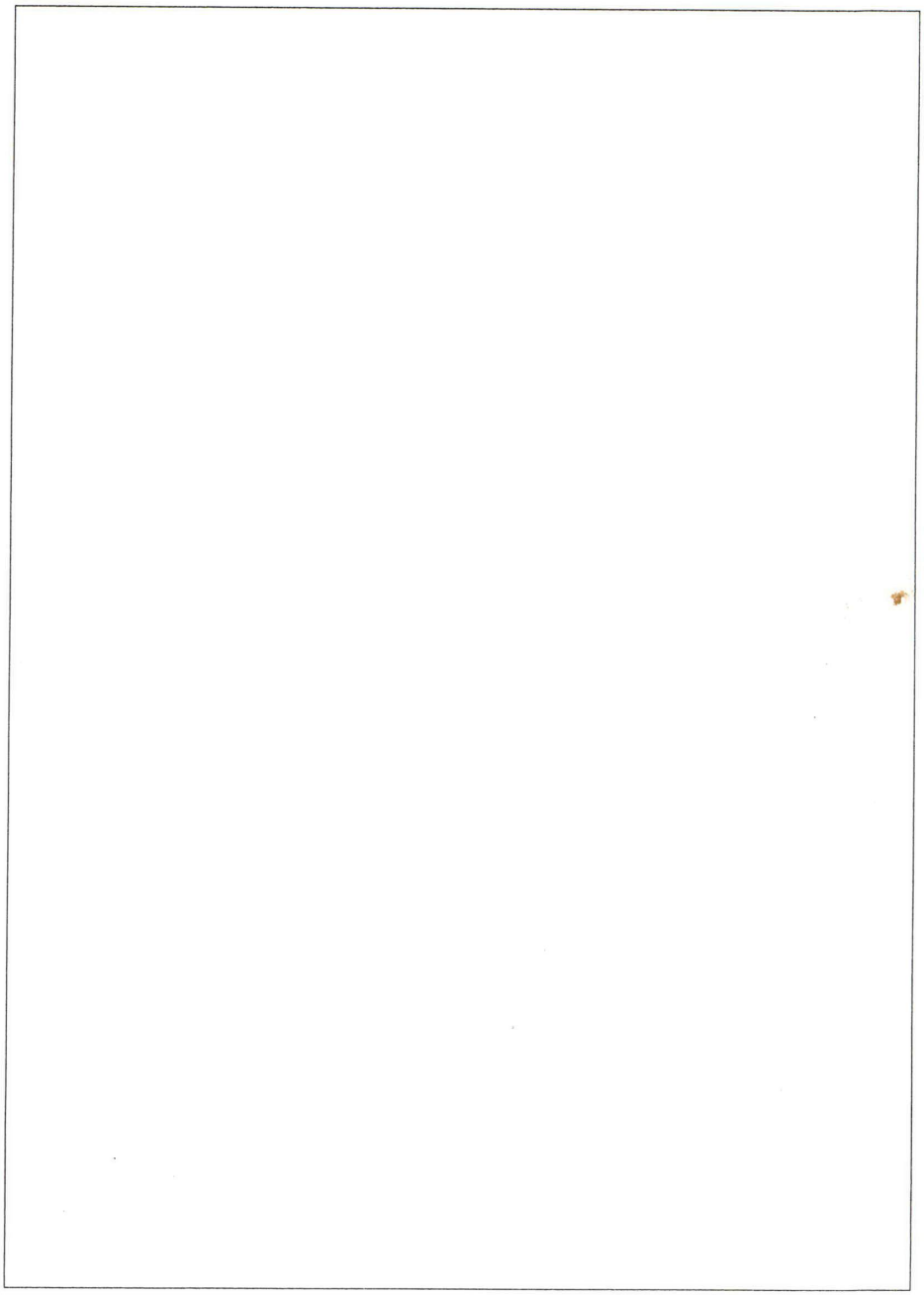
The order is increasing alphabetic order based on name. If two players have the same name, the new player should be inserted *after* the first matching player. Include appropriate `assert()`ion checks, including ones that ensure the new player is a singleton node.

It is not an error if the list is empty.

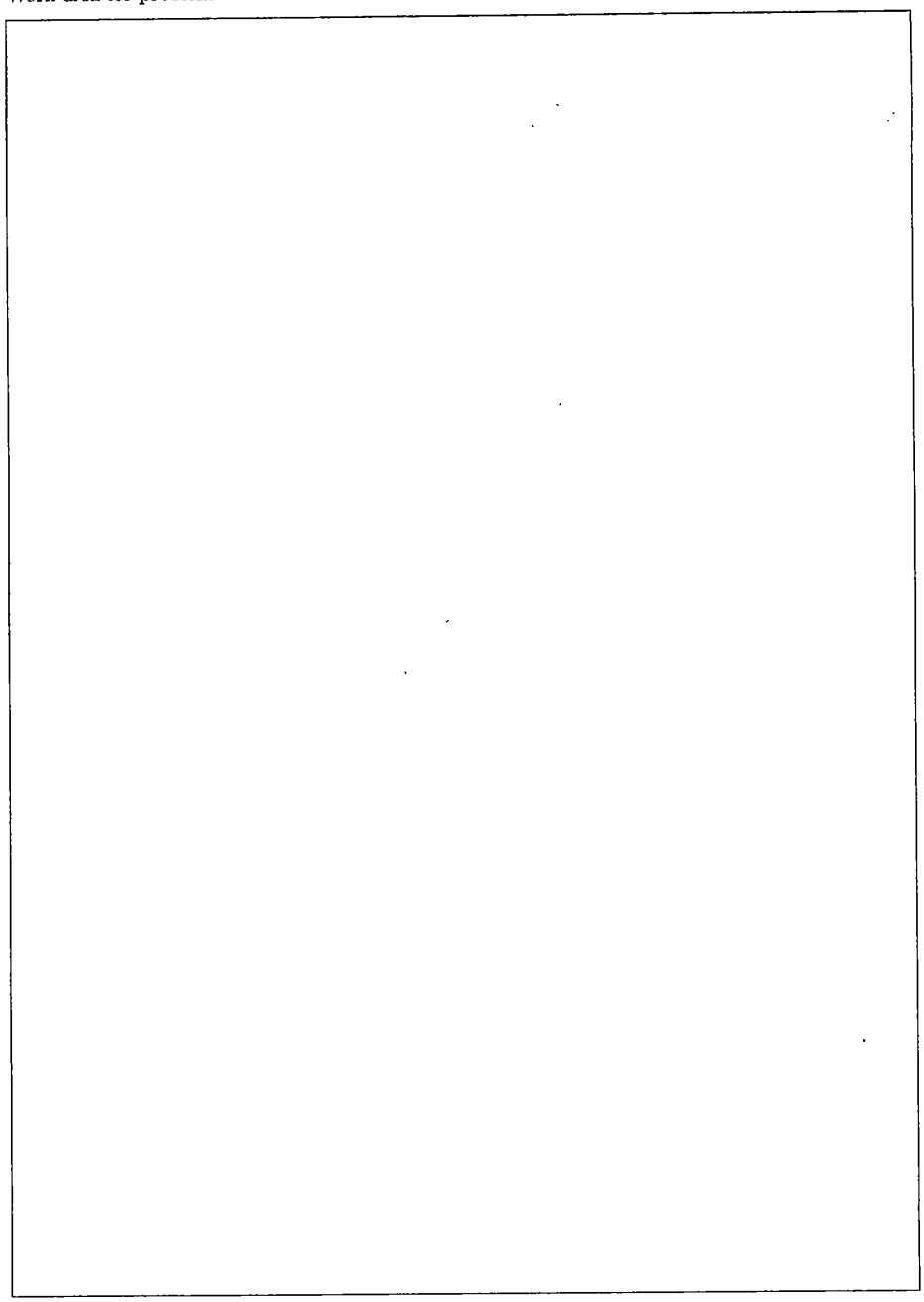


Turn over for additional space...

Work area for problem 2.c. continued...

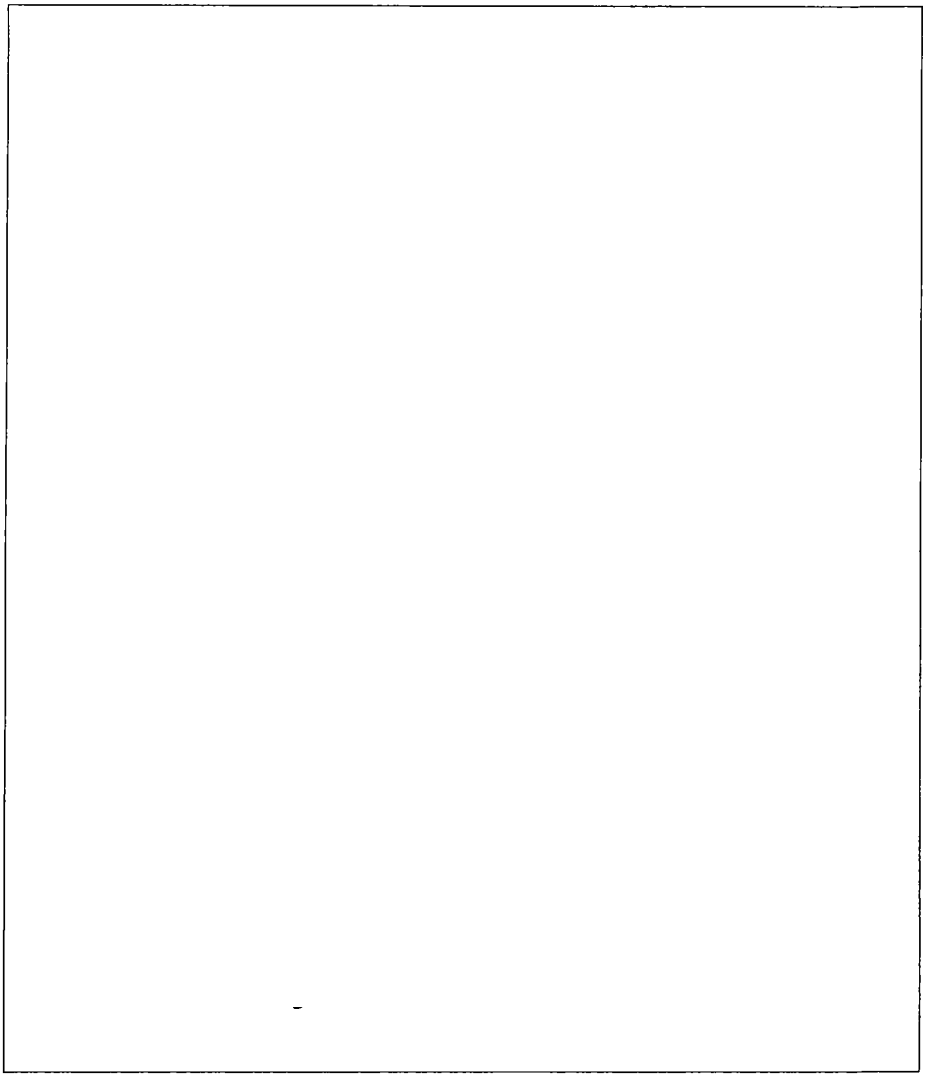


Work area for problem 2.c. continued...



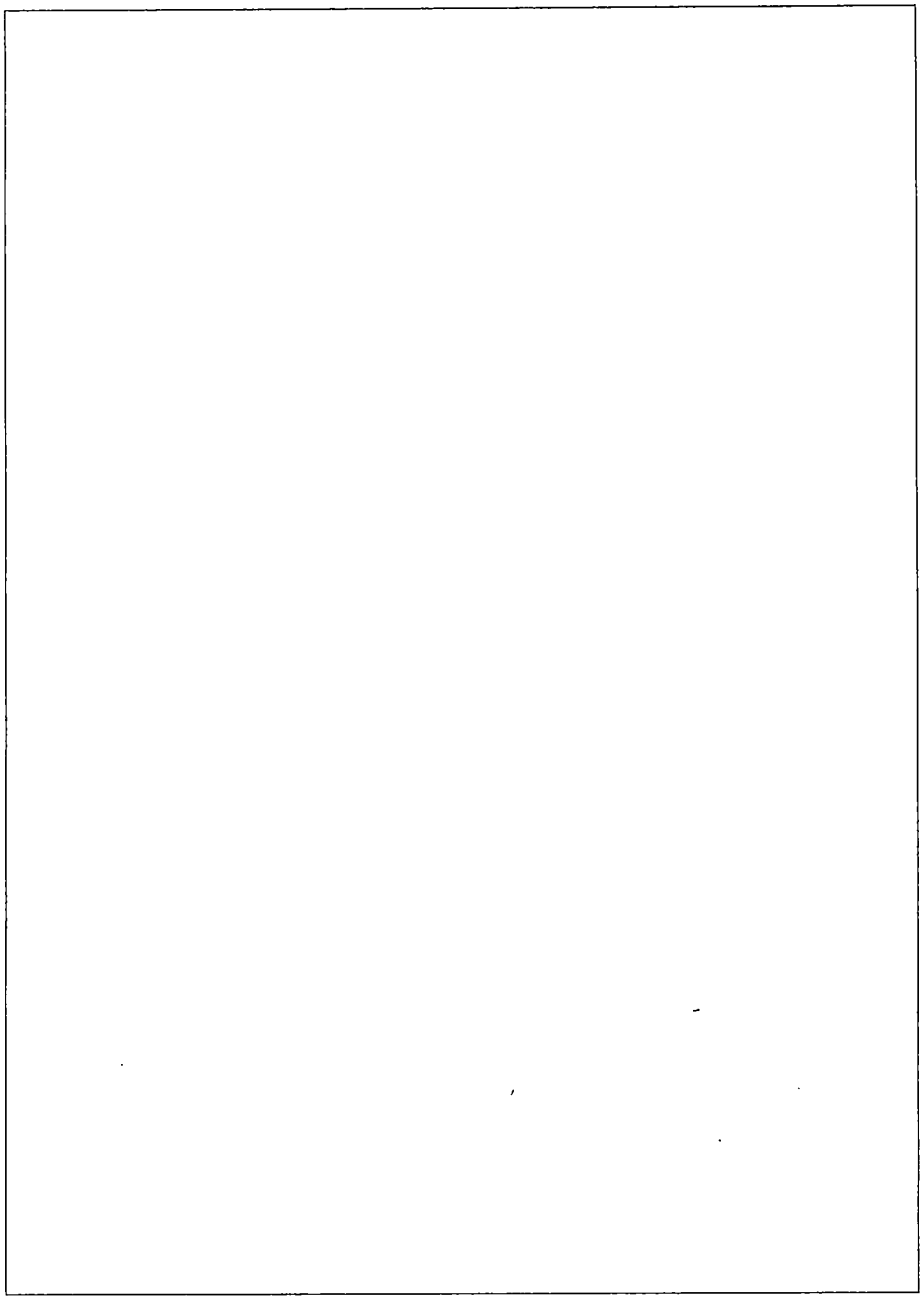
(d) (15 points) Write a function, `remove_players()`, that accepts two parameters—the *address* of a pointer to some `player` structure and a pointer to a `player` structure. Search the list referred to by the first parameter for all elements that match the `name`, `bat_avg`, or both from the second parameter. Remove all matches, deallocating the associated memory. Return the number of structures removed from the list (an integer).

This function should maintain the integrity of the list. The pointer that the first parameter points to should also be maintained appropriately. It is not an error for the list to be empty. Add appropriate `assert()`ion checks.

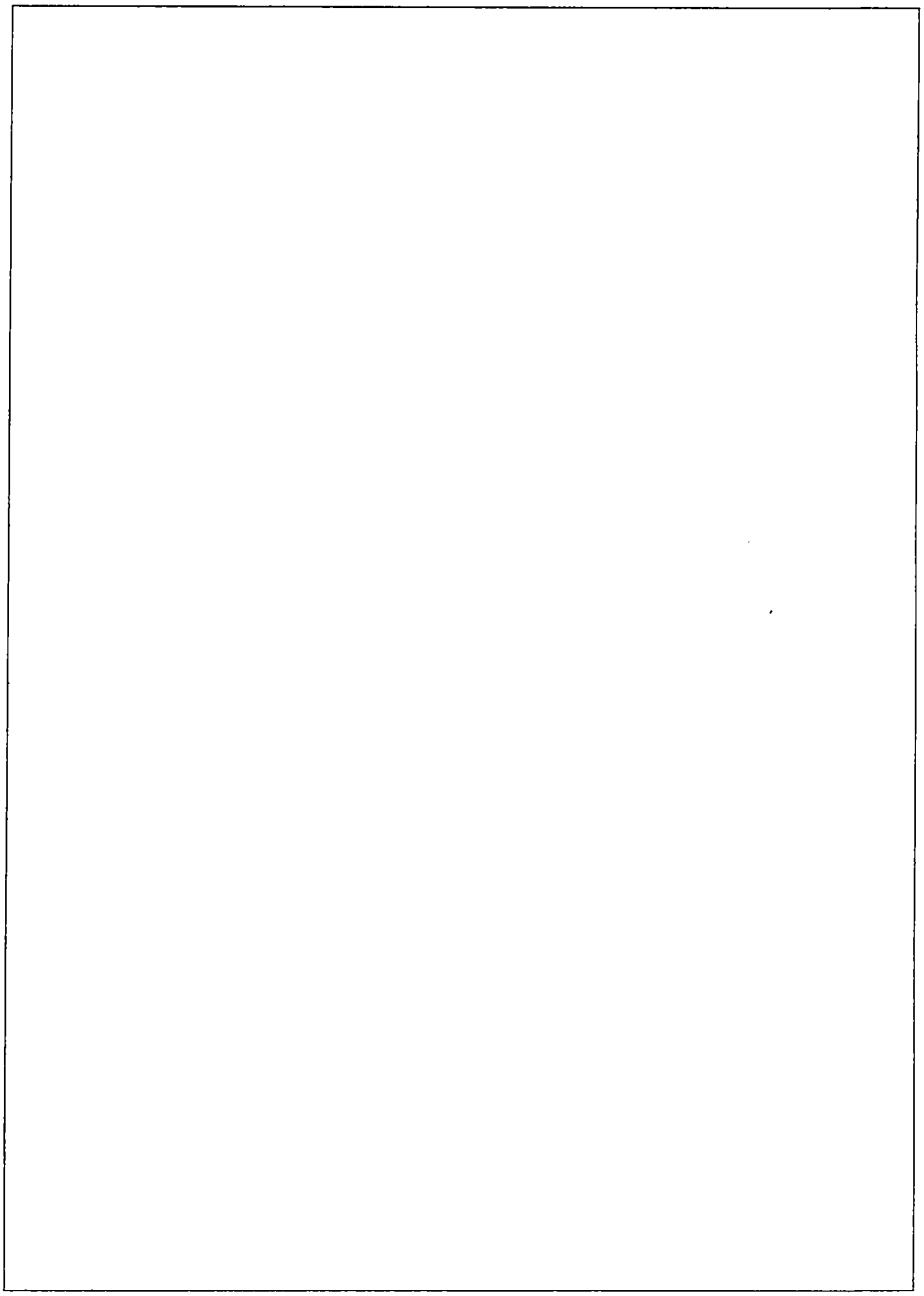


Addition space on next page...

Work area for problem 2.d. continued...



Work area for problem 2.d. continued...

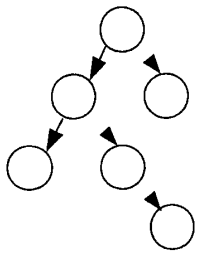


3. (35 points) The following questions deal with structures that are dynamically allocated and form a binary tree.

   (a) (5 points) Declare a structure, `tree_node`, which would be a valid node in a binary tree containing in this order: a pointer to a character (the `name`), a pointer to the node's left child, and a pointer to the node's right child. You may also declare your own type for this structure if you wish to slightly simplify the remaining questions.

(b) (10 points) Write a function, `create_node()`, which accepts a single parameter—a string (the name). Return a pointer to a newly allocated `struct tree_node`, containing a copy of the data pointed to by the first parameter. Be sure to properly initialize all structure fields!

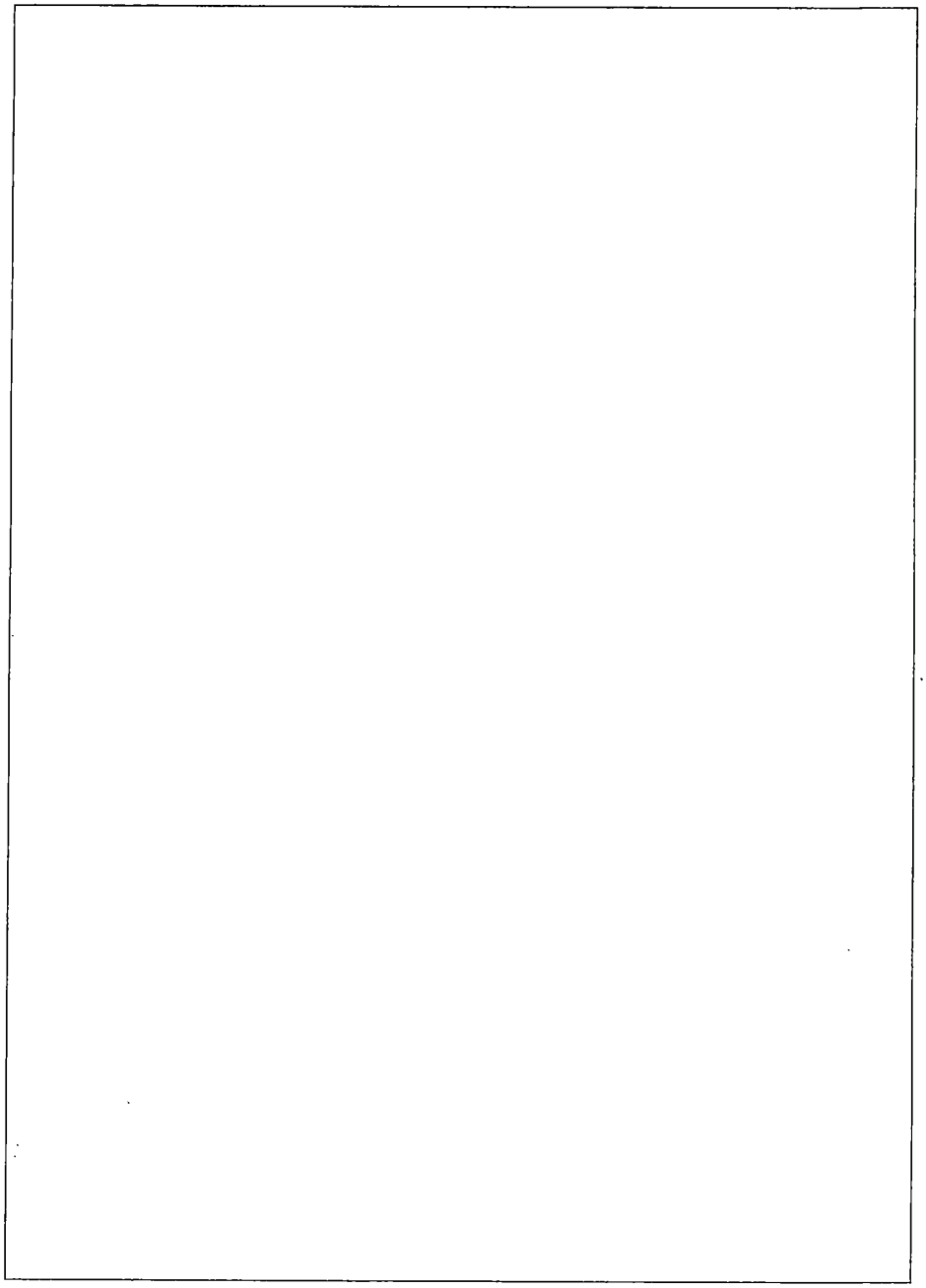
(c) (10 points) Define a *recursive* function, `tree_height()`, which accepts a single parameter—a pointer to the root of a preexisting tree. This function should return an integer—the height of the tree.



The height of a binary tree is the largest number of edges in a path from the root node to a leaf node. The tree above, for instance, has a height of 3. A root node by itself would have a height of zero.

Additional space on next page...

Work area for problem 3.d...



(d) (10 points) Define a *recursive* function, `do_tree_op()`, which accepts two parameters—a pointer to the root of a preexisting tree and a pointer to a function that returns an integer and accepts one parameter, a pointer to a node. Traverse the tree in postfix order, invoking the function pointed to by the second parameter on the node.

Assume that the root is valid. Return the sum of all integer values returned by the function pointed to by the second parameter.