Subject: Rules for Value type
Posted by ambyle on Thu, 08 Sep 2016 02:55:54 GMT

Are we allowed to implement value such that it can hold both the name and contents of a variable, or are we restricted to using one value for the name of the variable and another for its contents?

Subject: Re: Rules for Value type
Posted by tfmeads on Thu, 08 Sep 2016 03:09:01 GMT

He answered this in class, the reason why we can't do this is because it defeats the time/space saving benefits of a binary tree. How it's supposed to work is the value object saves the "key" in its sval pointer, when your program goes to fetch its real data, you're supposed to look for the node corresponding to the key in the binary tree holding all your variables.

That's how I understand it anyways, anybody feel free to correct me.

Subject: Re: Rules for Value type
Posted by ambyle on Thu, 08 Sep 2016 03:16:58 GMT

I guess I'm just not understanding why/how this saves time or space. If you could keep the contents of the variable with the name, you'd need one less value variable and you wouldn't have to do any searching to find the variable's contents. Finally, trying to organize the tree efficiently when a leaf is 'linked' to the node above it is very strange. Unless the nodes in the tree are supposed to contain a key and a value.

Subject: Re: Rules for Value type
Posted by BentHam on Thu, 08 Sep 2016 04:50:46 GMT

Yeah, I noticed that, too - having a variable key value essentially requires another data value within the same node, which definitely presents some annoyances as far as passing nodes and values is concerned.

I get the feeling that you're free to implement whatever members to your value class that you want - he wouldn't object to some creativity on your part, I don't think. I'm in too deep with my current implementation to change things up now, but I bet something like

typedef struct value {
    int type;
    int iVal;
    float rVal;
    char *sVal;
}
int isVar;
char *varName;
} value;

would make things a lot simpler, rather than saving the variable name in sVal and then having a separate value attached for your variable's value.

The only complication with this is that you would need a separate isVar member, rather than using your type variable to track if the value is a variable. This so that you can still know what data type your variable is, but check if it's a variable separately.

if (v->isVar)
    findNode(bst, varName);

That sort of thing.

Subject: Re: Rules for Value type
Posted by ambyle on Thu, 08 Sep 2016 05:02:05 GMT
View Forum Message <> Reply to Message

I was thinking about using a var struct inside of the value type. I'm already using unions so it wouldn't really take any more space. The only thing is I'm not sure if it's allowed.

Subject: Re: Rules for Value type
Posted by lusth on Thu, 08 Sep 2016 11:10:12 GMT
View Forum Message <> Reply to Message

As long as you are storing the variable in a bst, looking up that variable in the bst, and updating that value pointer after finding the variable in the bst, I'm OK with that.

I believe the conversation in class was about not using a bst to store variable values.