

SEB C Testat Fragen

Montag, 24. Juni 2019 18:36

The screenshot shows a web-based test interface for a C program. At the top, there's a header bar with the SEB logo, the text "Sprache", and a user profile for "Alexander Trussell". Below the header, the title "Formativer Test MC-C FS2019" is displayed, followed by the subtitle "Formativer Test im C-Teil des Moduls C". A section for "Header- und SourceCode-Dateien (7 Punkte)" is present, with the instruction "Sie haben die folgende Antwort gegeben:". The main content area contains instructions for finding code related to a stack, mentioning the global variable `stackState`. It also specifies that code should be submitted in `stack.h` and `stack.c`, noting that the order within the header file does not matter. A warning about incorrect code being deducted is included. The code itself is a C program with a stack implementation:

```
#include <stdio.h>

#define SIZE 3

enum ErrorCode { READY, EMPTY, FULL };

enum ErrorCode stackState = READY;

int stack[SIZE];
int pos = 0;

void push(int e) {
    if (pos < SIZE) {
        stack[pos++] = e;
        stackState = READY;
    } else {
        stackState = FULL;
    }
}

void pop() {
    if (pos > 0) {
        pos--;
        stackState = READY;
    } else {
        stackState = EMPTY;
    }
}

int peek() {
    if (pos > 0) {
        return stack[pos - 1];
    } else {
        return -1;
    }
}

int isEmpty() {
    if (pos == 0) {
        return 1;
    } else {
        return 0;
    }
}

int isFull() {
    if (pos == SIZE) {
        return 1;
    } else {
        return 0;
    }
}
```

A note at the bottom right of the code area says "#include \"stack.h\"".

```
#include <stdio.h>

#define SIZE 3

enum ErrorCode { READY, EMPTY, FULL };

enum ErrorCode stackState = READY;

int stack[SIZE];
int pos = 0;

void push(int e) {
    if (pos < SIZE) {
        stack[pos++] = e;
        stackState = READY;
    } else {
        stackState = FULL;
    }
}

int pop(void) {
    int r = -1;

    if (pos > 0) {
        r = stack[pos--];
        stackState = READY;
    } else {
        stackState = EMPTY;
    }
    return r;
}

int main(int argc, char** argv) {
    ...
    r = pop() + pop();
    if (stackState == READY) {
        printf("Resultat = %d\n", r);
        push(r);
    } else {
        printf("Fehler: Zu wenig Operanden auf Stack\n");
    }
    ...
}
```

#include "stack.h"

Die bestmögliche Lösung lautet:

// stack.h (ziehen Sie die entsprechenden Codezeilen hinein)	passt zu	void push(int e);
// stack.h (ziehen Sie die entsprechenden Codezeilen hinein)	passt zu	int pop(void);
// stack.h (ziehen Sie die entsprechenden Codezeilen hinein)	passt zu	#define _STACK_H
// stack.h (ziehen Sie die entsprechenden Codezeilen hinein)	passt zu	#ifndef _STACK_H
// stack.h (ziehen Sie die entsprechenden Codezeilen hinein)	passt zu	#endif
// stack.h (ziehen Sie die entsprechenden Codezeilen hinein)	passt zu	enum ErrorCode { READY, EMPTY, FULL };
// stack.h (ziehen Sie die entsprechenden Codezeilen hinein)	passt zu	extern enum ErrorCode stackState;
// stack.c (ziehen Sie die entsprechenden Codezeilen hinein)	passt zu	void push(int e) { . . . }
// stack.c (ziehen Sie die entsprechenden Codezeilen hinein)	passt zu	int pop(void) { . . . }
// stack.c (ziehen Sie die entsprechenden Codezeilen hinein)	passt zu	#include "stack.h"
// stack.c (ziehen Sie die entsprechenden Codezeilen hinein)	passt zu	static int stack[SIZE];
// stack.c (ziehen Sie die entsprechenden Codezeilen hinein)	passt zu	static int pos = 0;
// stack.c (ziehen Sie die entsprechenden Codezeilen hinein)	passt zu	enum ErrorCode stackState = READY;
// stack.c (ziehen Sie die entsprechenden Codezeilen hinein)	passt zu	#define SIZE 3

Sie haben 4.5 von 7 möglichen Punkten erreicht.

Sprache - Alexander Trussell ([Anmelden](#))

Formativer Test MC-C FS2019

Formativer Test im C-Teil des Moduls C

Zeiger und Vektoren (8 Punkte)

Sie haben die folgende Antwort gegeben:

Vervollständigen Sie die Funktion `createAndFillArray()`, welche einen Vektor mit `n` int Werten kreiert und diesen mit den Werten `0..n-1` initialisiert. Die Funktion liefert `SUCCESS` zurück falls die Speicherallokation erfolgreich war. Sonst wird `FAILURE` zurück geliefert.
Unter ein Beispiel der Konsole-Ausgabe mit `n=3`:

console output:

```
0ter Wert: 0
1ter Wert: 1
2ter Wert: 2
```

Füllen Sie die untenstehenden Lücken so aus, dass dieses Programm richtig funktioniert. (Tipp: schreiben Sie mindestens einen korrekten Buchstaben/Zeichen und wählen Sie aus dem erscheinenden "Pull-down-Menu" aus).

```
/* Program-Code starts here*/
enum FailureCode {SUCCESS, FAILURE};

main() {
    int i, *a;
    if (createAndFillArray(3, &a) == SUCCESS) {
        for (i = 0; i < 3; i++) {
            printf("%dter Wert: %d\n", i, a[i]);
        }
    }
}
```

Elements of the vector @param pa Reference of the vector to create and fill /

```

/* Program-Code starts here */

enum FailureCode {SUCCESS, FAILURE};

main() {
    int i, *a;

    if (createAndfillArray(3, &a) == SUCCESS) {
        for (i = 0; i < 3; i++) {
            printf("%dter Wert: %d\n", i, a[i]);
        }
    }

    /* Creates and fills a vector, @param n Number of elements of the vector @param pa Reference of the vector to create and fill */
    //Funktions-Kopf von createAndfillArray( )
}

enum FailureCode createA
{
    int i;

    *pa = malloc(n * sizeof(int)) // dynamische Memory Allokation

    if(*pa) //falls Memory Zeile vorher allokiert werden konnte - sonst Fehlerbehandlung -> else

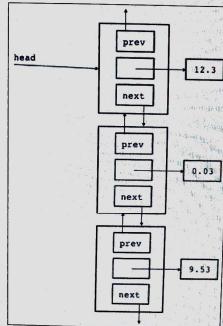
    {
        for (i = 0; i < n; i++) {
            *(pa + i) = i; //Werte zuweisen
        }
        return SUCCESS;
    } else {
        return FAILURE;
    }
}

```

Zur Speicherung von float-Werte wird eine doppelt verkettete Liste verwendet wie im Bild unten als Beispiel ersichtlich. Ein Knoten in dieser Liste hat einen Zeiger auf den float-Wert sowie Zeiger auf den Knoten vorher (*prev*) und auf jenen nachher (*next*).

1. Definieren Sie einen neuen Datentyp *NodePr_t* für einen Zeiger auf einen Knoten. (2 Punkte)
2. Definieren Sie einen neuen Datentyp *NodePr_l* für den Knoten. Die Struktur beinhaltet die beiden Zeiger *prev* und *next* sowie einen Zeiger *fP* auf einen float-Wert. Verwenden Sie den unter 1.) definierten neuen Datentyp *NodePr_t*. (2 Punkte)
3. Allocieren Sie *dynamisch* einen neuen Knoten und weisen Sie diesen Knoten der neuen deklarierten Zeiger-Variable *Anw* zu. Verwenden Sie die unter 2.) definierten Datentypen. (2 Punkt)
4. Wie im Bild unten werden Sie Ihrem ersten Knoten mit dem Zeiger *head* aus 3.) den Wert 12.3 zuweisen.

Achtung: Achten Sie auf den Speicherbereich, welchen Sie beschreiben. (2 Punkt)



Union und Bitfelder (8 Punkte)

Sie haben die folgende Antwort gegeben:

Schreiben Sie ein Programm, dass die Dezimalwerte von Sign, Exponent und Mantissa einer Gleitkommazahl ausgibt.

Den Aufbau einer IEEE754-Gleitkommazahl können Sie untenstehendem Beispiel entnehmen. Die dargestellten Dezimalwerte 0 für Sign, 128 für Exponent und 65536 für Mantisse erzielen den Wert 12.3.

Sign	Exponent	Mantissa
Value: +1	21	
Encoded as: 0	128	1.0078125 65536
Binary:		

1. Definieren Sie zuerst **einen neuen Type** für ein Bitfeld mit den Komponenten sign, exponent und mantissa mit den Größenangaben vom Bild. (2 Punkte)
2. Definieren Sie **eine Union** mit den zwei Alternativen Datentypen float und dem unter 1.) definierte Bitfeld. (2 Punkte)
3. Definieren Sie eine Variable mit der Bezeichnung *value* dieser Union. (1 Punkt)
4. Wiesen Sie der float-Alternative der Variable *value* den Wert 2015625f zu. (1 Punkt)
5. Geben Sie die drei Werte der Bitfield-Alternative der Variablen *value* auf die Konsole mit printf(..) aus. (2 Punkte)

Hinweise:

- Markieren Sie Ihre Teillösungen mit den Bezeichnungen 1. bis 5.
- Diese Aufgabe für 8 Punkte wird manuell korrigiert.

```

typedef bitfield struct {
    int Sign : 1;
    int Exponent : 8;
    int Mantissa : 23;
} bitfield_t;

union Value {
    float value;
    bitfield_t;
}
```

Ausdrücke (4 Punkte)

Sie haben die folgende Antwort gegeben:

Bestimmen Sie die Ausgabe folgender C-Code Zeilen auf der Konsole. (Lücke rechts davon die Ausgabe hinschreiben, Gross-/Kleinbuchstaben wird unterschieden)
***** ab hier der C-Code, Variablen wie folgt initialisiert *****

```
float af = 13.403;  
unsigned int b = 9;  
char* pText = "MC-c";
```

printf("%.2f\n", af+1.0); 14.40

printf("%c\n", *(pText+1)); C

printf("%d\n", b << 2); 4

printf("%s\n", ((af < 13) || (b < 9)) ? "YES" : "NO"); YES



Die bestmögliche Lösung lautet:

printf("%.2f\n", af+1.0); 14.40 oder 14,40

printf("%c\n", *(pText+1)); C

printf("%d\n", b << 2); 36

printf("%s\n", ((af < 13) || (b < 9)) ? "YES" : "NO"); NO

Sie haben 2 von 4 möglichen Punkten erreicht.