Dynamische Speicherverwaltung - Übungsaufgaben

1. (a) Was muss in einem Programm getan werden, damit die vier Grundbefehle zur dynamischen Speicherverwaltung zur Verfügung stehen?

   (b) Angenommen die folgenden Codezeilen stehen in einer Methode, in welchen Teilen des für das Programm zugeteilten Speichers werden die Daten gespeichert?

   i.) `const float pi = 3.14159;`

   ii.) `int number = 3;`

   iii.) `*int = malloc(sizeof(int));`

2. In dem folgenden Programm wird ein Array mit zufälligen Zahlen gefüllt und dann ausgegeben:

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main(int argc, char** argv)
{
    //Einlesen der Parameter
    int wieOft = 1;
    if(argc > 1)
    {
        sscanf(argv[1], "%d", &wieOft);
    }

    srand(time(NULL));  //Wird fuer Zufallszahlen gebraucht

    for(int i=0; i<wieOft; ++i)
    {
        short numbers[10];

        //Array wird beschrieben
        for(int j = 0; j<10; ++j)
        {
            numbers[j] = rand();
        }

        printf("Die %d .Zahlen sind: ", (i+1));

        //Ausgabe der Zahlen
        for(int j = 0; j<10; ++j)
        {
            printf("%d ", numbers[j]);
        }

        printf("\n");
    }
    return 0;
}
```

(a) In Zeile 18 wird ein Array angelegt:

```
short numbers[10];
```

Schreiben Sie diese Zeile so um, dass das Array zur Laufzeit dynamisch erzeugt wird.

(b) Jetzt kann es allerdings zu Problemen kommen. Was könnte passieren, wenn das Programm mit einer sehr hohen Zahl aufgerufen wird? Wie muss der Code ergänzt werden, um das zu vermeiden?

3. Ein beliebtes Anwendungsgebiet von dynamischer Speichernutzung sind wachsende Strukturen. In dieser Aufgabe geht es darum, einen Stack nach dem Prinzip der verketteten Liste zu implementieren. Auf der nächsten Seite ist der Quellcode eines Programms, das dies tut. Es ist allerdings noch nicht ganz vollständig. Die für den Stack interessanten Teile sind diese:

```
1  struct Link
2  {   //Ein Kettenglied
3      struct Link* nextLink;
4      int content;
5  };
6
7  struct LinkedStack
8  {   //Der Stack an sich
9      struct Link* tos;
10     unsigned short size;
11 };
12
13 struct LinkedStack stack = {NULL, 0}
14
15 //Legt ein neues Element auf den Stack
16 void push(int element)
17 {
18     //Muss noch implementiert werden
19 }
20
21 //Gibt das oberste Element zurueck
22 int peek()
23 {
24     if(stack.tos==NULL)
25     {
26         printf("Fehler: Stack ist leer\n");
27         return 0;
28     }
29     return stack.tos->content;
30 }
31
32 //Entfernt das Oberste Element und gibt es zurueck
33 int pop()
34 {
35     //Muss noch implementiert werden
36 }
```

(a) Implementieren Sie die Methode `void push(int element)`, sodass sie ihre Spezifikation erfüllt.

(b) Implementieren Sie die Methode `int pop()`, sodass sie ihre Spezifikation erfüllt. Vergessen Sie dabei nicht, den nicht mehr benutzten Speicher freizugeben.

Auf den folgenden Seiten ist der vollständige Quellcode des Programms. Sie können damit überprüfen, ob ihre Implementation funktioniert.

stack.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

/*Dieses Programm implementiert einen Stack nach dem Prinzip der verketteten Liste.*/

struct Link
{   //Ein Kettenglied
    struct Link* nextLink;
    int content;
};

struct LinkedStack
{   //Der Stack an sich
    struct Link* tos;
    unsigned short size;
};

struct LinkedStack stack = {NULL, 0};

//Initialisiert den Stack
void initStack();

//Legt ein neues Element auf den Stack
void push(int element);

//Gibt das oberste Element zurueck
int peek();

//Entfernt das oberste Element und gibt es zurueck
int pop();

//Leert den Stack und gibt den Speicher frei
void deleteAll();

//Liest einen Integer vom Benutzer ein
int getInput();


int main()
{
    //Benutzerinformationen
    printf("Was soll getan werden?\n 0: Beenden\n 1: Element hinzufuegen\n");
    printf(" 2: Peek \n 3: Pop \n 4: Speicher ausgeben\n 5: Stack leeren\n");

    int input = 0;         //Für Benutzerinputs
    bool running = true;   //Wurde das Programm schon beendet?

    //Main Loop
    do  //while(running)
    {
        printf("Eingabe: ");
        input=getInput();

        switch(input)
        {
            case 0:
                running = false;
                break;
            case 1:
                printf("Fuege diesen Integer hinzu: ");
                push(getInput());
                break;
            case 2:
                printf("Das Oberste Element ist %d. \n", peek());
                break;
            case 3:
                printf("Es wurde %d entfernt. \n", pop());
                break;
```

```
70              case 4:
71                  int mem = stack.size * sizeof(struct Link);
72                  printf("Der Stack benutzt momentan %d Byte.\n", mem);
73                  break;
74              case 5:
75                  deleteAll();
76                  break;
77              default:
78                  printf("Ungueltige Eingabe. Bitte etwas anderes eingeben.\n");
79          }
80          input=0;
81          printf("\n");
82      }
83      while(running);
84
85      deleteAll();
86      printf("Programm wird beendet.\n");
87
88      return(0);
89  }
90
91  void push(int element)
92  {
93      //Muss noch implementiert werden
94  }
95
96  int peek()
97  {
98      if(stack.tos==NULL)
99      {
100         printf("Fehler: Stack ist leer\n");
101         return 0;
102     }
103     return stack.tos->content;
104 }
105
106 int pop()
107 {
108     //Muss noch implementiert werden
109 }
110
111 void deleteAll()
112 {
113     struct Link* current = stack.tos;
114     while(current != NULL)
115     {
116         struct Link* next = current->nextLink;
117         free(current);
118         current = next;
119         stack.size--;
120     }
121     stack.tos = NULL;
122     printf("Der Stack wurde geleert.\n");
123 }
124
125 int getInput()
126 {
127     int input = 0;
128     char buff[256];
129     fgets(buff, 256, stdin);
130     buff[11] = 0x00;    //die ersten 10 Zeichen
131     sscanf(buff, "%d", &input);
132     return input;
133 }
```