Technische Universität Berlin **Fachgebiet für Neuronale Informationsverarbeitung**

Übungsblatt 10

Prof. Dr. Klaus Obermayer und Thomas Goerttler

C

dynamisches Dynamit

Verfügbar ab:	**14.06.2019**
Abgabe bis:	**23.06.2019**

Aufgabe 1: Alphabet

4 Punkte

Schreibt eine Funktion, welche die ersten `n` Buchstaben des Alphabets in eine neue Zeichenkette schreibt und diese zurückgibt. Die Funktion bekommt also den Parameter `unsigned int nChars` übergeben und muss den Speicherplatz für eine neue Zeichenkette dieser Länge allozieren. Für den Aufruf mit `nChars = 5` müsste demzufolge die Zeichenkette `"abcde"` erzeugt werden. Schreiben Sie nun noch eine main-Funktion, welche eine entsprechende Zahl vom Benutzer einliest, die Funktion aufruft und die erhaltene Zeichenkette ausgibt. Behandelt sämtliche möglichen auftretenden Zahlen, die der Nutzer eingeben könnte.

Musterlösung:

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

char* alphabet(unsigned int nChars)
{
   if(nChars > 26)
      nChars = 26;
   char *ret = calloc(nChars+1, sizeof(char));
   if(ret){
      char c = 'a';
      for(int i = 0; i < nChars; i++, c++){
         ret[i] = c;
      }
      ret[nChars] = '\0';
   }
   return ret;
}

int main()
{
   unsigned int nChars = 0;
   char *str = NULL;
   printf("Please enter number of alphabet chars to be printed: ");
   scanf("%u", &nChars);
```

```
25      str = alphabet(nChars);
26      if(str)
27          puts(str);
28      return 0;
29  }
```

Aufgabe 2: Studierende — 5 Punkte

Deklariert zunächst eine Struktur `struct student` und damit den Typ `Student` (typedef). Diese soll folgende Elemente enthalten:

- Einen Vornamen von maximal 20 Zeichen Länge
- Einen Nachnamen von maximal 20 Zeichen Länge
- Einen Array, in welchem die Punkte von 3 Hausaufgabenblättern gespeichert werden können (Nur ganzzahlige Punkte sind erforderlich)

Schreibt eine `main`-Funktion, welche zuerst ein Array unbestimmter Länge aus Studierenden erstellt.
Ruft nun für jeden Studenten eine (selbst zu schreibende!) Funktion `void student_einlesen(Student* p_stud, int n)` auf, welche das array auf n Einträge erweitert, dann die Daten vom Nutzer einliest und in die Struktur schreibt. Die Funktion soll solange erneut aufgerufen bis ein Vorname beginnend mit x eingegeben wurde.
Wenn alle Daten eingelesen wurden, ruft für jeden Studenten die (selbst zu schreibende...) Funktion `void student_ausgeben` auf, welche alle Daten der übergebenen Struktur im Terminal ausgibt. Diese Übergabe der Struktur soll by-value, also als Student erfolgen. Anstatt der Punkte jeder Aufgabe soll die Summe aller Punkte ausgegeben werden und ob das zur Zulassung zur Klausur reicht.

Beginnt die main() mit folgendem code und beantwortet die Frage in dem Kommentar:

```
1 int main()
2 {
3     unsigned int aktuelleAnzahl = 0;
4     Student* kurs = malloc(aktuelleAnzahl);
5     //Frage: Was bedeutet malloc hier und warum ist es notwendig?
6  ..
```

Musterlösung:

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  typedef struct student{
5      char vorname[21];
6      char nachname[21];
7      int punkte[3];
8  } Student;
9
10 void student_einlesen(Student p_stud[], unsigned int count)
11 {
12     p_stud = realloc(p_stud, count * sizeof(Student));
13     if(!p_stud) exit(-1);
14
```

```
15      printf("\nGebe den Vornamen ein: ");
16      scanf("%s", p_stud[count-1].vorname);
17      printf("Gebe nun den Nachnamen ein: ");
18      scanf("%s", p_stud[count-1].nachname);
19      printf("Gebe nun nacheinander die 3 Punkte ein:\n");
20      for(int i=0; i<3; i++)
21      {
22          printf("%i: ", i+1);
23          scanf("%i", &(p_stud[count-1].punkte[i]));
24      }
25  }
26
27  void student_ausgeben(Student s)
28  {
29      int sum = 0;
30      for(int i=0; i<3; i++)
31          sum += s.punkte[i];
32      printf("\nStudent: %s %s\n", s.vorname, s.nachname );
33      printf("erreichte Punkte: %i, zugelassen: %u \n", sum, (sum >= 30)
        );
34  }
35
36  int main()
37  {
38      unsigned int aktuelleAnzahl = 0;
39      Student* kurs = malloc(aktuelleAnzahl);
40      // weisst indirekt auch NULL an kurs zu:
41      // um realloc anwenden zu koennen muss kurs initialisiert sein.
42      // direkt NULL ist aber eine nur lokal(in main) gültige adresse
43      // bei der uebergabe dann quasi als call-by-value
44      // d.h. der reservierte speicher besteht dann auch nur in der
45      // einlesen-Funktion lokal dort.
46      // die mit malloc bestellten adressen/bloecke, auch wenn NULL,
47      // bleiben bis free() ueberall erhalten.
48
49      do
50      {
51          aktuelleAnzahl++;
52          student_einlesen(kurs, aktuelleAnzahl);
53      }while(kurs[aktuelleAnzahl-1].vorname[0] != '-'); //asciienter=10
54
55
56      for (int i=0; i<aktuelleAnzahl; i++)
57          student_ausgeben(kurs[i]);
58
59      free(kurs);
60  }
```

Aufgabe 3: Dynamisch — 3 Punkte

Führt eine Handsimulation für folgendes Programm durch – schreibt also eine Tabelle, aus der hervorgeht, welche Zeilen nacheinander ausgeführt werden, und welche Werte die Variablen nach Ausführung dieser Zeile jeweils haben. Achtet auf die Hinweise!

- Variablen, die zu diesem Zeitpunkt noch nicht existieren, werden mit - markiert

- Variablen, deren Wert undefiniert ist, werden mit `undef` markiert
- Die erste ausgeführte Zeile einer Funktion ist immer die öffnende Klammer {
- Die letzte ausgeführte Zeile einer Funktion ist immer die schließende Klammer }
- Beim Sprung in eine Funktion wird hinter die aktuelle Zeile in Klammern die Zeile, aus der der Sprung stattgefunden hat, geschrieben
- Die nächste Zeile, die nach einem `return`-Statement ausgeführt wird, ist die schließende Klammer der beinhaltenden Funktion! (Sprich: die Funktion wird danach verlassen ohne den auf das return folgenden Code zu beachten.)

Hinweis: die beiden Fragen unter der Tabelle bitte nicht vergessen.

```
1  #include <string.h>
2  #include <stdio.h>
3  #include <stdlib.h>
4
5  #define StringAdr char*
6
7  typedef char string[10];
8
9  int main()
10 {
11    string a = "Guten";
12    string b = "Tag";
13    StringAdr platz = 0;
14    int fuenfIntsPlatz = 5 * sizeof(int);
15    platz = (StringAdr) malloc ( fuenfIntsPlatz );
16    if(platz == 0) return -1;
17    StringAdr p = &a[0] + strlen(a);
18    platz = (StringAdr) realloc ( platz, 10000000000000 );
19    platz = (StringAdr) p;
20    strncpy(platz, b, strlen(b));
21 }
```

Die erste Speicheranforderung kann erfüllt werden, der allozierte Block liegt an Adresse 1380; die zweite Speicheranforderung wird abgelehnt (zu groß!).
Die strings a und b erhalten Startadressen 1000 und 2000.

Zeile	platz	fuenfInts	p	a	b

Zwei weitere Fragen:

- was passiert eigentlich in Zeile 20 genau - wenn das geht: warum, wenn nicht: warum nicht?

- was passiert wenn man in Zeile 21 `free(platz);` einfügt und warum/warum nicht?

Musterlösung:

1.

Zeile	platz	fuenfInts	p	a	b
10	-	-	-	-	-
11				1000	
12					2000
13	0				
14		20			
15	1380				
16					
17			1005		
18	0				
19	1005				
20					
21	-	-	-	-	-

2. der Inhalt von b wird für Länge von b Zeichen nach a kopiert(da zeigt p noch hin), das geht gut weil a gerade lang genug ist (typedef .. [10]).
3. free() kann nur auf dynamisch erzeugte speicher angewendet werden, platz ist aber mit p (also adresse der mitte von char array a) überschrieben worden und a ist nicht dynamisch. absturz des programms folgt.

Aufgabe 4: Simcity — 3 Punkte

Führt eine Handsimulation für folgendes Programm durch – schreibt also eine Tabelle, aus der hervorgeht, welche Zeilen nacheinander ausgeführt werden, und welche Werte die Variablen nach Ausführung dieser Zeile jeweils haben. Achtet auf die Hinweise!

- Variablen, die zu diesem Zeitpunkt noch nicht existieren, werden mit - markiert
- Variablen, deren Wert undefiniert ist, werden mit `undef` markiert
- Tragt nur Variablenänderungen ein, leere Zellen werden als "unverändert" interpretiert.
- Die erste ausgeführte Zeile einer Funktion ist immer die öffnende Klammer {
- Die letzte ausgeführte Zeile einer Funktion ist immer die schließende Klammer }
- Beim Sprung in eine Funktion wird hinter die aktuelle Zeile in Klammern die Zeile, aus der der Sprung stattgefunden hat, geschrieben
- Die nächste Zeile, die nach einem `return`-Statement ausgeführt wird, ist die schließende Klammer der beinhaltenden Funktion! (Sprich: die Funktion wird danach verlassen ohne den auf das return folgenden Code zu beachten.)

```
1 #include <stdlib.h>
2
3 typedef double* DoublePointer;
4
5 int main()
6 {
7   char groesse = 3 * sizeof(double);
8   DoublePointer excel = (DoublePointer) calloc( 1, groesse );
9   if (excel == 0) return -1;
```

```
10   for(double* p = excel; p < (excel + 4); p += 2)
11   {
12     *p += 3;
13   }
14   double* excel2 = (DoublePointer) realloc( excel,  10000000000000 );
15   if (excel2)
16     free( excel2 );
17   else
18     free( excel );
19 }
```

Die erste Speicheranforderung kann erfüllt werden, der allozierte Block liegt an Adresse 1024; die zweite Speicheranforderung wird abgelehnt (zu groß!).

Folgender Beitrag auf des Programmierer's wichtigster Internetseite `StackOverflow` kann euch helfen, das Verhalten ab Zeile 14 richtig zu beurteilen:

https://stackoverflow.com/questions/1607004/does-realloc-free-the-former-buffer-if-it-fails

Zeile	groesse	excel	excel2	p	excel[0]	excel[1]	excel[2]	excel[3]

Musterlösung:

	Zeile	groesse	excel	excel2	p	excel[0]	excel[1]	excel[2]	excel[3]
	6	-	-	-	-	-	-	-	-
	7	24							
	8		1024			0.0	0.0	0.0	0.0
	9								
	10				1024				
	11								
	12					3.0			
	13				1040				
1.	10								
	11								
	12							3.0	
	13				1056				
	10				-				
	14			0					
	15								
	17								
	18					-	-	-	-
	19	-	-	-					