Studiengänge: Informatik, Flug- und Fahrzeuginformatik

Prüfung
Grundlagen der Programmierung 1

Prüfer: Regensburger
Prüfungsdauer: 90 Minuten
Hilfsmittel: keine

Studiengang	Dozent	Matrikelnummer	Semester	Raum	Platz

Aufgabe	1	2	3	4	Σ	Note
Punkte						

Bitte beachten:
Tragen Sie Ihre persönlichen Angaben auf dieses Deckblatt ein.
Schreiben Sie Ihre Antworten direkt in die dafür vorgesehenen freien Stellen des Angabetextes.
Geben Sie alle Blätter wieder ab, auch wenn einzelne Seiten nicht beschrieben sein sollten.
Alle Blätter der Angabe müssen bei der Abgabe wieder richtig sortiert und geheftet sein!

Viel Erfolg!

Aufgabe 1: (Programmverständnis und Syntax, ca. 25%)

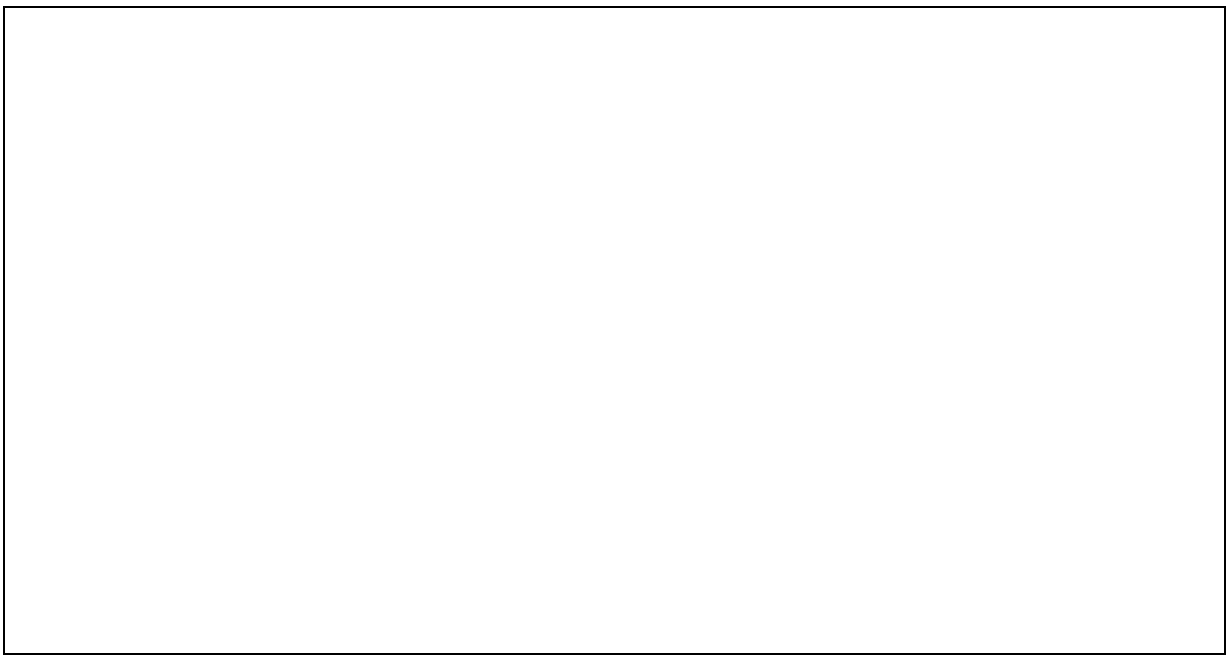
a) Geben Sie für die Funktionen f1 bis f5 jeweils den Funktionskopf (Prototyp) an. Die Typen lassen sich aus den Aufrufen der Funktionen in der Funktion `main` ermitteln.

```
struct elem {
    char* s;
    struct elem *next;
};

int main(void) {
    char c;
    int *s = f1(&c);
    struct elem n[] = {{"xyz",NULL},f2("def"),{"abc",NULL}};
    n[1].next = &n[0];
    char *p = f3(n[1].next)->s;
    f4(*p, &(n[2].next));
    n[0].s = f5(&(n[0].s),s);
    return 0;
}
```

```
**********************************************************************
f1:



**********************************************************************
f2:



**********************************************************************
f3:



**********************************************************************
f4:



**********************************************************************
f5:
```

b) Welche Ausgaben erzeugt das folgende Programm?

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

struct board { int **cells; } bin;

int main() {
    int c,i,j, num=5;
    struct board *b = &bin;

    b->cells = malloc(sizeof(int*) * num);
    for(c=0; c < num; c++){
        b->cells[c] = calloc(num, sizeof(int));
    }

    for(i=0; i < num; i++) {
        for (j=0; j < i+1; j++) {
            if (j == 0 || j == i) {
                b->cells[i][j] = 1;
            } else {
                b->cells[i][j] =
                    b->cells[i-1][j-1] + b->cells[i-1][j];
            }
        }
    }

    for(i=0; i < num; i++) {
        for (j=0; j < num; j++) {
            printf("%3d", b->cells[i][j]);
        }
        printf("\n");
    }
    exit(EXIT_SUCCESS);
}
```



Aufgabe 2: (Algorithmenentwurf, ca. 25 %)

Schreiben Sie eine Prozedur `void drucke(int n)`, welche das Bild des griechischen Buchstabens Λ (großes Lambda) aus Sternchen zusammengesetzt auf dem Standardausgabekanal (Bildschirm) ausgibt.
Der Buchstabe ist gemäß Parameter n wie nachfolgend gezeigt zu skalieren:

Aufgabe 3: (Zeichenketten, ca. 25 %)

Wir implementieren eine Variante des bekannten Kinderspiels *B-Sprache*, bei dem in einem Satz alle Vokale v (a,e,i,o,u) ersetzt werden durch die Zeichenfolge v**b**v. Der Vokal v wird also verdoppelt und zwischen den Vokalen der Buchstabe ‚b' eingefügt. Aus dem Satz „kannst Du das auch" wird demnach „kabannst dubu dabas abaubuch".

Der Einfachheit halber sollen bei dieser Aufgabe nur Kleinbuchstaben zugelassen und keine Umlaute (ä,ö,ü) erlaubt sein.

Gegeben Sei folgendes Hauptprogramm:

```
int main(void){
    char* saetze[] = {
        "das ist die b-sprache.", "kannst du die auch?",
        "die kann doch jedes kind!", NULL,
    };

    int i = 0;
    while (saetze[i] != NULL) {
        char * trans;
        trans = nachB(saetze[i]);
        printf("%s\n", trans);
        free(trans);
        i++;
    }
    exit(EXIT_SUCCESS);
}
```

Bei Ausführung erzeugt das Programm folgende Ausgabe:

```
dabas ibist dibiebe b-sprabachebe.
kabannst dubu dibiebe abaubuch?
dibiebe kabann doboch jebedebes kibind!
```

Teilaufgabe a)
Schreiben Sie eine Funktion

```
bool istVokal(char c)
```

die `true` zurückgibt, wenn das Zeichen `c` ein Vokal ist, `false` sonst.

Teilaufgabe b)
Schreiben Sie eine Funktion

```
int zaehleVokale(char* s)
```

die die Anzahl der Vokale berechnet, die in der Zeichenkette `s` enthalten sind.
Sie dürfen annehmen, dass `s != NULL` gilt.

Teilaufgabe c)
Schreiben Sie die in `main()` verwendete Funktion

```
char* nachB(char* orig)
```

die die Zeichenkette `orig` in die B-Sprache übersetzt und als Ergebnis zurückliefert.

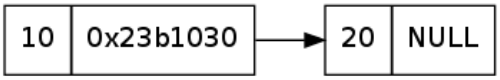
Hinweise und Vorgaben:

- Sie dürfen annehmen, dass `orig != NULL` gilt.
- Die Zeichenkette `orig` darf durch die Funktion `nachB` nicht verändert werden!
- Der zur Speicherung der neuen Zeichenkette benötigte Speicher soll in der Funktion `nachB` alloziert werden. Hierbei dürfen Sie annehmen, dass die Allokation des Speichers durch `malloc()` zu keinem Fehler führt.
- Es soll nur gerade so viel Speicher alloziert werden, damit die in die B-Sprache übersetzte neue Zeichenkette darin Platz findet.
- Verwenden Sie die in den Teilaufgaben a) und b) implementierten Funktionen.

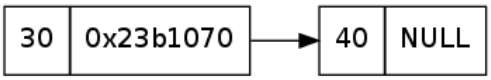
Aufgabe 4: (Einfach verkettete Listen, ca. 25 %)

Im nachstehenden Programm werden zunächst mittels `list_insert_end_proc()` zwei verkettete Listen a und b erzeugt.

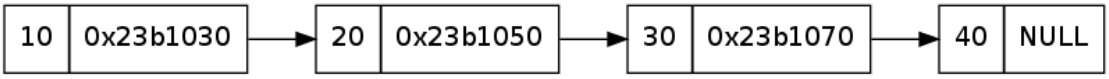
Liste a:



Liste b:



Danach wird durch Aufruf der Funktion `list_append()` die Liste b hinten an die Liste a angehängt.



Zum Schluss wird der dynamisch allozierte Speicherplatz wieder freigegeben.

```
typedef struct node {
  int          data;
  struct node* next;
} node_t;


int main(){
  node_t *anchor_a = NULL;
  node_t *anchor_b = NULL;

  list_insert_end_proc(&anchor_a,10);
  list_insert_end_proc(&anchor_a,20);
  list_insert_end_proc(&anchor_b,30);
  list_insert_end_proc(&anchor_b,40);

  anchor_a = list_append(anchor_a, anchor_b);

  list_free(anchor_a);

  // Aufgabe d)
  // list_free(anchor_b);

  return EXIT_SUCCESS;
}
```

In den nachfolgenden Teilaufgaben müssen sie einige der `in main()` verwendeten Funktionen sowie eine weitere Hilfsfunktion zum Erzeugen von Listenelementen implementieren.

Hinweise:
- Sie müssen keine Fehler bei der Allokation von Speicher behandeln.

- Folgende Funktion aus der Bibliothek `stdlib` ist unter Umständen hilfreich:

```
   void *malloc(size_t size);
```

Teilaufgabe a) Speicherallokation
Schreiben Sie eine Funktion `node_t* list_create_node(int data)` welche ein Listenelement vom Typ `node_t` im Heap-Speicher erzeugt, die Komponente `data` mit dem Wert des gleichnamigen Parameters belegt und die Komponente `next` mit einem NULL-Zeiger initialisiert. Als Ergebnis wird ein Zeiger auf das neue Listenelement zurückgegeben.

Teilaufgabe b) Iteration, prozedurale Schnittstelle
Schreiben Sie eine **iterative** Prozedur
`void list_insert_end_proc(node_t** panchor, int data)`
welche ein neues Listenelement hinten an die per Referenz übergebene Liste anfügt und dessen Komponente `data` mit dem Wert des gleichnamigen Parameters belegt.

Teilaufgabe c) Rekursion
Schreiben Sie eine **rekursive** Funktion

```
node_t*  list_append(node_t *anchor_a, node_t* anchor_b)
```

welche die am Zeiger `anchor_b` hängende Liste hinten an die Liste hängt, die über `anchor_a` erreichbar ist.
Als Funktionsresultat soll ein Zeiger auf die verlängerte Liste zurückgegeben werden.

Teilaufgabe d) Verständnisfrage
Darf man am Ende der Funktion `main()` direkt vor der `return`-Anweisung folgenden weiteren Aufruf tätigen?

```
list_free(anchor_b);
```

Begründen Sie Ihre Antwort!

(Ende des Aufgabentextes) Viel Erfolg!