Klausur PROG-C FS16

Name:

Klasse:

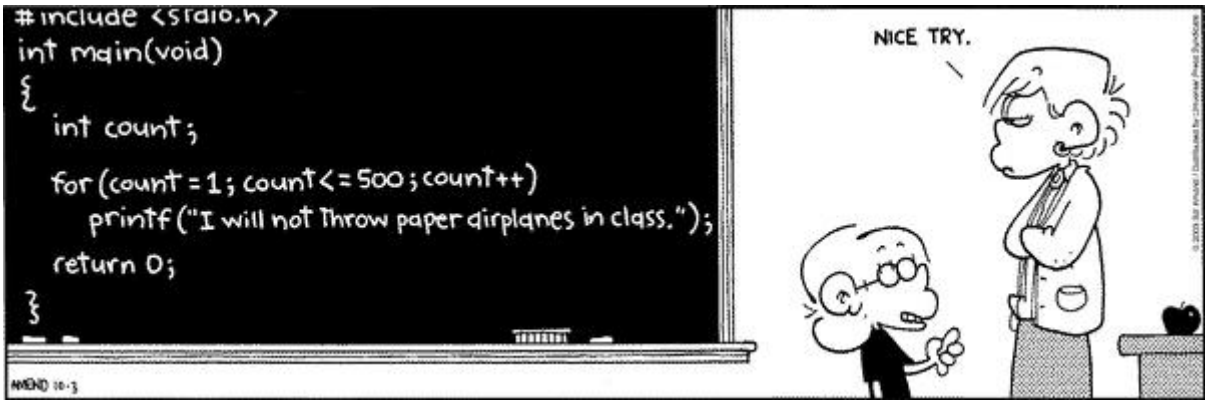
Zeit: 60 Minuten

Bedingungen:

- Die Aufgaben werden auf den ausgeteilten Blättern gelöst.
- Falls der Platz nicht ausreicht, benutzen Sie bitte die Rückseiten und bringen Sie bei der Aufgabe einen entsprechenden Verweis an.
- Blätter **nicht** auseinandernehmen!
- **Keine** elektronischen Hilfsmittel
- Wo immer möglich soll der Lösungsweg ersichtlich sein.
- Unredliches Verhalten hat die Note 1 zur Folge.

Maximale Punktzahl: 60

Erreichte Punktzahl: **Note:**



Answers

als "verborgener Text"

Aufgabe 1 : 10 Punkte **a) 2P b) 2P c) 2P d) 2P e) 2P**

Ergänzen Sie die Adressen und Inhalte des Arrays txt. Dabei werden die Werte des Arrays von einer Teilaufgabe zur nächsten überragen. Der Array txt werde an der Adresse 0x100 angelegt.

a) `char txt[3] = {'\0'};` 2 Punkte

Adresse [HEX]	00 00 01 00	**00 00 01 01**	**00 00 01 02**		
Inhalt [HEX]	**00**	**00**	**00**		

b) `txt[2] = 80;` 2 Punkte

Inhalt [HEX]	**00**	**00**	**50**		

c)
```
char *p = txt;
*p = *(p + 2) + 3;
```
2 Punkte

Inhalt [HEX]	**53**	**00**	**50**		

d) `*(p + 1) = (int)txt + 1;` 2 Punkte

Inhalt [HEZ]	**53**	**65**	**50**		

e) Inhalt von txt als char 2 Punkte

Inhalt [char]	**'S'**	**'e'**	**'P'**		

1 Punkt für korrekte Position
1 Punkt für korrekten Inhalt

Aufgabe 2 : 10 Punkte **a) 4P b) 1P c) 1P d) 4P**

Gegeben ist die Funktion:

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

void inverse(char * const);

int main(void){
  char c[] = "SEPFS16";
  char *p = c;
  (void) printf("\n%d", sizeof(c));
  (void) printf("\n%c", p[0]);
  (void) printf("\n%d", p[0] - p[2]);
  (void) printf("\n%s", p + p[0] - p[2]);
  inverse(p);
  (void) printf("\n%s", p);
  return EXIT_SUCCESS;
}
```

a) Was ist die Ausgabe von: 4 Punkte

```
(void) printf("\n%d", sizeof(c));
```
1P : 8

```
(void) printf("\n%c", p[0]);
```
1P : S

```
(void) printf("\n%d", p[0] - p[2]);
```
1P : 3

```
(void) printf("\n%s", p + p[0] - p[2]);
```
1P : FS16

b) Ist dieser Ausdruck Pointerarithmetik ? 1 Punkt

```
(void) printf("\n%d", p[0] - p[2]);
```
1P : NO

c) Ist dieser Ausdruck Pointerarithmetik ? 1 Punkt

```
(void) printf("\n%s", p + p[0] - p[2]);
```
1P : YES

d) Schreiben Sie die Funktion inverse um c[ ] zu invertieren: 4 Punkte

```
1  void inverse(char * const pc) {
2    int i = 0, len = 0, temp = 0;
3    for(i = 0; *(pc + i); i++, len++);
4    for(i = 0; i < len/2; i++) {
5      temp = *(pc + i);
6      *(pc + i) = *(pc + len - i - 1);
7      *(pc + len - i - 1) = temp;
8    }
9
10
11
12 }
```

1P: Länge bestimmen
1P: Iteration zur Mitte
1P: Temp Variable
1P: korrekter Tausch

Aufgabe 3 : 10 Punkte **a) 3 P b) 3 P c) 3 P d) 1 P**

Schreiben Sie entsprechenden Source in ANSI C

a) Bestimmen Sie mithilfe des Modulo-Operators die zweitletzte Ziffer einer mindestens dreistelligen reellen Zahl. 3 Punkte

```
1  int z = 123;
2
3  z = (z % 100) - (z % 10)) / 10 ;
4
```

1P: modulo 100
1P: modulo 10
1P: Resultat korrekt

b) Zur exakten Festlegung der Schaltjahre dienen die folgenden Regeln:
- ist die Jahreszahl durch 4 teilbar, so ist das Jahr ein Schaltjahr.
Diese Regel hat allerdings eine Ausnahme:
- ist die Jahreszahl durch 100 teilbar, so ist das Jahr kein Schaltjahr.
Diese Ausnahme hat wiederum eine Ausnahme:
- ist die Jahreszahl durch 400 teilbar, so ist das Jahr doch ein Schaltjahr.
Erstellen Sie ein C Source Sequenz, die berechnet, ob eine Jahreszahl ein Schaltjahr bezeichnet oder nicht. 3 Punkte

```
1  int jahr = 2016;
2  if ( jahr % 4 == 0)
3    if( jahr % 100 == 0)
4      if( jahr % 400 == 0) printf ("yes");
5      else printf ("no");
6    else printf ("yes");
7  else printf ("no");;
8
9
```

1P: 1 Regel
1P: 2 Regel
1P: 3 Regel

c) Primzahlberechnung bis zur Zahl 100 nach dem Sieb-Verfahren:
Schreiben Sie Zahlen 2..100 in einen Array. Beginnend mit der kleinsten Zahl wird die Zahl als Primzahl auf dem Bildschirm ausgegeben und gleichzeitig alle Vielfachen dieser Zahl im Array auf 0 gesetzt. Dann wird die nächste Zahl, die nicht 0 ist, im Array entsprechend bearbeitet. 4 Punkte

```
1  int prim[100], i,j;
2  for(i = 0; i < 100; i++) prim[i] = i;
3  for(i = 2; i < 100; i++) {
4    if (prim[i]){
5      printf("%d ", prim[i]);
6      for (j = 2 * i; j < 100; j += i) {
7        prim[j] = 0;
8      }
9    }
10 }
11
12
```

1P: array initialisieren
1P: array iterieren; Start bei 2
1P: prim ausgeben
1P: Vielfache 0 setzten

Aufgabe 4 : 10 Punkte **a) 2 P b) 2 P c) 2 P d) 4 P e) 2 P**

Gegeben ist der folgende Programmcode:

```
1
2  int *pi1, *pi2, i;
3
4  pi1 = pi2 + i;
5  pi1 = i + pi2;
6  i = pi1 * pi2;
7  i = pi1 - pi2;
8  i = pi1 + pi2;
9
```

Erklären Sie die Bedeutung der Zeilen mit den folgenden Nummern:

a) Zeile 4: **`pi1 = pi2 + i;`** 2 Punkte

1P: pi2 ist ein Pointer, also dessen Inhalt eine Adresse
1P: dazu wird i addiert und als neue Adresse in pi2 gespeichert

b) Zeile 5: **`pi1 = i + pi2;`** 2 Punkte

1P: i ist eine integer Zahl, dazu wird eine Adresse pi2 addiert
1P: das Resultat wird als neue Adresse in pi2 gespeichert

c) Zeile 6: **`i = pi1 * pi2;`** 2 Punkte

1P: die beiden Inhalte von pi1 und pi2 (Adressen) werden summiert
1P: das macht keinen Sinn, resultiert in einem compile error
error: invalid operands to binary * (have 'int *' and 'int *'

d) Zeile 7: **`i = pi1 - pi2;`** 2 Punkte

1P: die beiden Inhalte von pi1 und pi2 (Adressen) werden subrtahiert
1P: resultiert im Offset als sizeof(int) der beiden Pointer

d) Zeile 8: **`i = pi1 + pi2;`** 2 Punkte

1P: Addieren von zwei Adresse
1P: das macht keinen Sinn, resultiert in einem compile error
error: invalid operands to binary + (have 'int *' and 'int *')

Aufgabe 5 : 10 Punkte ► nächste Seite

Aufgabe 6 : 10 Punkte ► übernächste Seite

"accordnig to a rseearcehr at cmabrigde uinversiyt, it deos'nt mtaetr in what odrer the ltetres in a word are, the only ipmortnat tihng is that the frist and last lteetr be at the rgiht palec. The rest can be a ttoal mess and you can sitll read it wtihuot porblme. this is bceasue the hmuan mind does not read eevry lteetr by istlef but the word as a wohel."

Typoglycemia

Vervollständigen Sie das Programm um diesen Text zu erzeugen.
Dabei gelten die folgenden beiden Regeln für die Länge len eines Wortes:

Regel 1: len = 5 ► Buchstaben 2 und 3 vertauschen

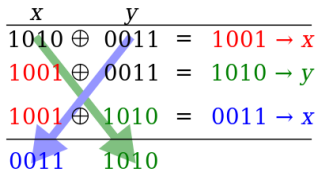
Regel 2: len > 5 ► Regel 1 und zusätzlich Buchstaben len-3 und len-2 vertauschen

Grammatik- sowie Satzzeichen dürfen Sie ignorieren!

Aufgabe 5 Erzeugen Sie den Text als Kopie (issue) des originalen Textes (origin). In der Kopie sind die Buchstaben entsprechend den beiden Regeln vertauscht. Vervollständigen Sie das Programm auf der **nächsten** Seite.

Aufgabe 6 Schreiben Sie die gleiche Logik (Regeln). Diesmal werten Sie den Text als Argumente aus der Kommandozeile aus. Verwenden Sie **alternative Kontrollstrukturen** zum Implementieren der beiden Regeln. Erzeugen Sie jeweils eine Kopie eines Kommandozeilen-Arguments (Wort) und geben Sie diese nach Bearbeitung (Regeln) auf der Konsole aus. Vervollständigen Sie das Programm auf der **übernächsten** Seite.

Nützliche Funktionen um diese Aufgabe anzugehen:



NAME	**SWAPCHAR** – vertauscht zwei char ► als Macro definiert
SYNOPSIS	`#define SWAPCHAR(x, y) ((x)^=(y),(y)^=(x),(x)^=(y))`
DESCRIPTON	The Macro SWAPCHAR swaps two characters using an algorithm that uses the XOR bitwise operation to swap values of distinct variables having the same data type without using a temporary variable.

NAME	**strtok** - split string into tokens
SYNOPSIS	`include <string.h>` `char *strtok(char *restrict s1, const char *restrict s2);`
DESCRIPTON	A sequence of calls to strtok() breaks the string pointed to by s1 into a sequence of tokens, each of which is delimited by a byte from the string pointed to by s2. The first call in the sequence has s1 as its first argument, and is followed by calls with a null pointer as their first argument. The separator string pointed to by s2 may be different from call to call.
RETURN VALUE	Upon successful completion, strtok() shall return a pointer to the first byte of a token. Otherwise, if there is no token, strtok() shall return a null pointer.

NAME	**strcat** - concatenate two strings
SYNOPSIS	`include <string.h>` `char *strcat(char *restrict s1, const char *restrict s2);`
DESCRIPTON	The strcat() function shall append a copy of the string pointed to by s2 (including the terminating NUL character) to the end of the string pointed to by s1. The initial byte of s2 overwrites the NUL character at the end of s1. If copying takes place between objects that overlap, the behavior is undefined.
RETURN VALUE	The strcat() function shall return s1; no return value is reserved to indicate an error.

Typoglycemia: Source zur Aufgabe 5

```
1  #include <stdlib.h>
2  #include <stdio.h>
3  #include <string.h>
4  #define SWAPCHAR(x, y) ((x)^=(y),(y)^=(x),(x)^=(y))
5  char origin[] = "according to a researcher at cambridge ...";
6
7  int main(void){
8    (void) printf("\n%s", origin);
9    char *word = strtok(origin, " ");
10   char *issue = NULL;
11   issue = (char*)malloc(sizeof(origin) / sizeof(char) + 1);
12   if (issue == NULL) return EXIT_FAILURE;
13   strcpy(issue, "");
14   int len = 0;
15   while (word != NULL) {
16     len = strlen(word);
17     switch (len) {
18       case 0 ... 3: break;
19       default: SWAPCHAR(word[len - 2], word[len - 3]);
20       case 5: SWAPCHAR(word[1], word[2]);
21     }
22     strcat(issue, word);
23     strcat(issue, " ");
24     word = strtok(NULL, " ");
25   }
26   if (issue) free((void *)issue);
27
28
29 1P: korrekte Verwendung von malloc inkl. Anzahl Byte
30 1P: korrekter cast nach malloc
31 1P: Adress-Prüfung nach malloc
32 1P: issue initialisieren
33 1P: Iteration über Tokens
34 1P: Länge des aktuellen Tokens
35 1P: Regel 1
36 1P: Regel 2
37 1P: issue zusammensetzen inkl Leerzeichen zwischen Wörter
38 1P: free
39
40   (void) printf("\n%s", issue);
41   return EXIT_SUCCESS;
42 }
43
```

Typoglycemia: Source zur Aufgabe 6

```
1  #include <stdlib.h>
2  #include <stdio.h>
3  #include <string.h>
4
5  #define SWAPCHAR(x, y) ((x)^=(y),(y)^=(x),(x)^=(y))
6
7  int main(int argc, char **argv){
8    if (argc < 2) return EXIT_FAILURE;
9    int len = 0;
10   char *word;
11   argv++;
12   while (*argv){
13     word = *argv;
14     len = strlen(word);
15     if (len > 4) SWAPCHAR(word[1], word[2]);
16     if (len > 5) SWAPCHAR(word[len - 2], word[len - 3]);
17     printf("%s ", word);
18     argv++;
19   }
20
21
22
23
24
25
26
27
28
29 1P: include <string.h>
30 1P: korrekte main Argumente
31 1P: Prüfung ob Argumente vorhanden (argc)
32 1P: Iteration über argv
33 1P: Alternative Kontrollstruktur (unterschiedlich zu Aufgabe 5)
34 1P: Länge des aktuellen Wortes
35 1P: Regel 1
36 1P: Regel 2
37 1P: Wort ausgeben
38 1P: nächster Iterationsschritt (argv incrementieren)
39
40
41   return EXIT_SUCCESS;
42 }
43
```