

Cedric Aebi 17-103-235

Nicolas Müller 17-122-094

RA FS 18 Serie 1

Qiyang Hu, Givi Meishvili, Adrian Wälchli

Die erste Serie ist bis Dienstag, den 13. März 2018 um 15:00 Uhr zu lösen. Ihre Abgabe in ILIAS besteht aus zwei Teilen: Ein PDF mit den Lösungen zu den theoretischen Aufgaben und eine Zip-Datei mit allen Quellcodedateien des Programmierteils. Für Fragen steht im ILIAS jederzeit ein Forum zur Verfügung. Allfällige unlösbare Probleme sind uns so früh wie möglich mitzuteilen, wir werden gerne helfen. Viel Spass!

Theorieteil

Gesamtpunktzahl: 12 Punkte

1 String (1 Punkt)

Wie lang (in Bytes) ist der String "computer architecture"?

2 Array-Zugriff mit Pointern (1 Punkt)

Gegeben ist folgende `getAt` Funktion, welche ein bestimmtes Element aus einem `double`-Array zurückgibt:

```
1 double a[10];
2
3 double getAt(int i) {
4     return a[i];
5 }
```

Folgende Funktion macht genau das selbe, verwendet jedoch direkte Pointerarithmetik.

```
1 double getAt(double *a, int i) {
2     return *(a+i);
3 }
```

Geben Sie je ein Beispiel für eine äquivalente Funktion basierend auf einem `int`- und einem `short`-Array. Begründen Sie Ihre Antwort. Identifizieren und beheben Sie potenzielle Probleme der bereitgestellten Codefragmente.

3 Pointer-Typen (2 Punkte)

Gegeben sei der folgende Code:

```
long a = 1234567890; /* Hex: 499602d2 */
long b = 1000000000; /* Hex: 3b9aca00 */
```

Das führt zu folgendem Speicherinhalt (Ausschnitt):

Adresse Inhalt (Hex)

...	
bffff604	00
bffff605	ca
bffff606	9a
bffff607	3b
bffff608	d2
bffff609	02
bffff60a	96
bffff60b	49
...	

→

b

a

Hinweis: Byte-Reihenfolge ist Little-Endian. Das heisst das niedrigstwertige Byte steht an der tiefsten Speicher-Adresse, die weiteren Bytes folgen in aufsteigender Wertigkeit.

Geben Sie an, was der folgende Code ausgibt und warum (Sie können annehmen, dass mit GNU C gearbeitet wird, d.h. es gilt `sizeof(void)=1`):

```
1 void * p = &b;
2 printf("%x\n", p);
3 printf("%x\n", *(long*)p++);
4 printf("%x\n", *(char*)p++);
5 printf("%x\n", *(unsigned char*)p++);
6 printf("%x\n", p);
```

4 Parameterübergabe (2 Punkte)

Gegeben sei das folgende Programm. Geben Sie an, welchen Wert die Variablen `i` und `j` nach Programmdurchlauf haben.

```
1 int main () {
2     int i, j;
3     i = 103;
4     j = increment(&i);
5 }
6
7 int increment(int *x) {
8     return ++(*x);
9 }
```

Welchen Wert hätten die Variablen `i` und `j`, wenn die Funktion `increment` wie folgt definiert wäre?

```
*7 int increment(int *x) {
*8     return (*x)++;
*9 }
```

5 Pointer Arithmetik (2 Punkte)

Beschreiben Sie, was bei jedem `printf` im untenstehenden Programmstück ausgegeben wird.

```
1 short x[3] = {3, 2, 1};
2 short *px = x;
3 printf("%i %i\n", *x, *px);
4 px++;
5 printf("%i %i\n", *x, *px);
```

Beschreiben Sie, welche Ausgabe das folgende Programmstück zeigt. Welche möglichen Probleme auftreten können?

```
1 short x = 3;
2 short *px = &x;
3 *(px--) = 20;
4 *px = 21;
5 printf("%i %i\n", x, *px);
```

6 Structs und Unions (3 Punkte)

Erklären Sie, welchen Wert dieser Programmteil ausgibt:

```
1 struct {
2     char a[10];
3     char b;
4     char c;
5     short int d;
6 } myStruct;
```

```

7
8 union {
9     char a[12];
10    int b;
11    short int d[4];
12 } myUnion;
13
14 printf("%i", sizeof(myStruct));
15 printf("%i", sizeof(myUnion));

```

Zeichnen Sie die unterschiedlichen Speicheraufteilung der Variablen innerhalb der vorherigen Beispiele `myStruct` und `myUnion`:

7 Define (1 Punkt)

Erklären Sie wie `define` in C funktioniert. Welchen Wert folgendes Programmstück ausgibt?

```

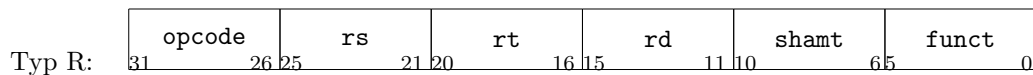
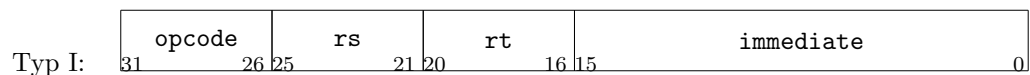
1 #define callA callB(13)
2
3 void callB(int a) {
4     printf("%i\n", a+2);
5 }
6
7 int main() {
8     callA;
9     return EXIT_SUCCESS;
10 }

```

Programmierteil

Die folgenden Aussagen beziehen sich auf die Datei `cSerie1.c`, diese kann von Ilias heruntergeladen werden. Ihre Aufgabe ist es, das gegebene Programmgerüst wie folgt zu vervollständigen:

- Laden Sie die Datei `cSerie1.c` von Ilias herunter und studieren diese aufmerksam. Versuchen Sie zu verstehen, was die bereits vorhandenen Teile bedeuten.
- Tragen Sie zuerst Ihren Namen sowie den Namen einer allfälligen Übungspartnerin oder eines allfälligen Übungspartners an der vorgesehenen Stelle in der Datei ein.
- Erstellen Sie drei Bitfelder namens `InstructionTypeI`, `InstructionTypeJ` und `InstructionTypeR`, die wie folgt aufgebaut sind:



Hinweise:

- http://publications.gbdirect.co.uk/c_book/chapter6/bitfields.html
 - Benutzen Sie `typedef`.
 - Dasjenige Element, das den niedrigwertigsten Bits entspricht (z.B. `funct` beim Typ R), soll an der ersten Stelle stehen.
- (d) Erstellen Sie eine `union` namens `Instruction` mit den folgenden drei Feldern:
- `i` vom Typ `InstructionTypeI`
 - `j` vom Typ `InstructionTypeJ`
 - `r` vom Typ `InstructionTypeR`

Hinweise:

- http://publications.gbdirect.co.uk/c_book/chapter6/unions.html
 - Benutzen Sie `typedef`.
- (e) Erstellen Sie eine Aufzählung namens `InstructionType` die folgende Elemente umfasst: `iType`, `jType`, `rType`, `specialType`

Hinweise:

- http://publications.gbdirect.co.uk/c_book/chapter6/enums.html
 - Benutzen Sie `typedef`.
- (f) Erstellen Sie eine Struktur namens `Operation`, die folgende Elemente enthält:
- einen String der Länge `OP_NAME_LENGTH` namens `name`
 - einen `InstructionType` namens `type`
 - einen Zeiger auf eine Funktion namens `operation`, die als einzigen Parameter einen Zeiger auf eine `Instruction` und leeren Rückgabewert hat

Hinweise:

- http://publications.gbdirect.co.uk/c_book/chapter6/structures.html
 - http://publications.gbdirect.co.uk/c_book/chapter5/function_pointers.html
 - Benutzen Sie `typedef`.
- (g) Erstellen Sie eine Struktur namens `Function`, die folgende Elemente enthält

- einen String der Länge `FUNC_NAME_LENGTH` namens `name`
- einen Zeiger auf eine Funktion namens `function`, die als einzigen Parameter einen Zeiger auf eine `Instruction` und leeren Rückgabewert hat

Hinweise analog vorheriger Teilaufgabe.

- (h) Implementieren Sie die Funktion `printInstruction`. Diese soll eine Instruktion formatiert ausgeben. Die Formatierung unterscheidet sich je nach Typ der zugehörigen Operation:

- iType**
- Name der Operation als linksausgerichteter String mit Feldbreite 4
 - `rt` und `rs` jeweils als vorzeichenbehaftete Ganzzahlen der Länge 2 mit führenden Nullen aufgefüllt
 - `immediate` als Hexadezimalzahl der Länge 4, ebenfalls mit führenden Nullen aufgefüllt und mit `0x` beginnend
 - Zeilenumbruch
- jType**
- Name der Operation als linksausgerichteter String mit Feldbreite 4
 - `address` als Hexadezimalzahl der Länge 8, ebenfalls mit führenden Nullen aufgefüllt und mit `0x` beginnend
 - Zeilenumbruch
- rType**
- Name der (zugehörigen) *Funktion* als linksausgerichteter String mit Feldbreite 4
 - `rd`, `rs` und `rt` jeweils als vorzeichenbehaftete Ganzzahlen der Länge 2 mit führenden Nullen aufgefüllt
 - `shamt` als Hexadezimalzahl der Länge 4, ebenfalls mit führenden Nullen aufgefüllt und mit `0x` beginnend
 - Zeilenumbruch
- specialType**
- Name der Operation als linksausgerichteter String mit Feldbreite 4
 - Zeilenumbruch

Hinweise:

- Die der Instruktion zugehörige Operation erhalten Sie mit `Operation o = operations[i->i.opcode]` analog für Funktionen
 - http://publications.gbdirect.co.uk/c_book/chapter3/flow_control.html#section-5
 - http://publications.gbdirect.co.uk/c_book/chapter9/formatted_io.html
 - Als Beispiel liegt der gewünschte Output des fertigen Programmes in der Datei `outputc1` bei. Stellen Sie sicher, dass Ihr Programm diesen Output erzeugt.
- (i) Stellen Sie sicher, dass Ihr Programm mit dem Befehl `gcc -ansi -pedantic -Wall -o cSerie1 cSerie1.c` ohne Fehler und Warnungen kompiliert. Dies ist eine notwendige Voraussetzung, damit der Programmierteil als erfüllt gilt.
- (j) Benennen Sie die Datei `<nachname>.c` (wobei `<nachname>` natürlich durch Ihren Nachnamen zu ersetzen ist).
- (k) Geben Sie die Datei elektronisch durch Hochladen in Ilias ab.

Theorie Teil

1) 22. 20 characters 1 "empty character" (space)
1 "0" at the end. 


2) short b[10];

```
int getAt(int i) {  
    return b[i];  
}
```

```
}  
int getAt(short *b, int i) {  
    return *(b+i);  
}
```

Begründung: Alles "äquivalent" zu int.

Behebung der Fehler: Grundsätzlich kein Array-Grenzen checking

```
double getAt(int i) {  
    if (i < 0 || i > (sizeof(a) / sizeof(double))) {  
        printf("Array out of bounds\n");  
        exit(1);   
    }  
    return a[i];  
}
```

(Funktion 2 mit pointer ist analog)

3) Ausgabe: 1020 ; physikalische Speicheradresse von b
 (alle in ; da in p 8b abgespeichert ist
 Hexadezimal 3b9aca00 ; p gecasted in long *. Ausgabe ist b
 konvertiert) ; in Hexadezimal, da man den pointer
 : *p ausgibt, also worauf p zeigt.
 ffffffffca ; p um eins grösser. Und nun char *
 ; p zeigt auf Speicheradresse
 bffff605 also ca.
 9a ; p + 1 (bffff606). Zeigt auf 9a. Aus-
 gabe in unsigned. Deswegen die
 "f" nicht.
 1023 ; p + 1 (bffff607). Physikalische Adresse
 von 3b.

4) i = 104 / j = 104
 i = 104 / j = 103 (Variante 2)

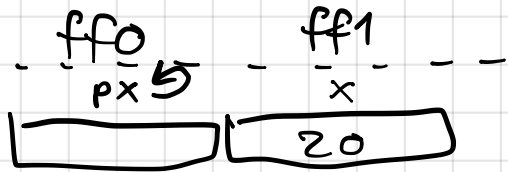
5) Erstes printf. Gibt für x x[0] also 3 aus, da
 das Array x[] ja ein pointer ist. Und für *px
 auch 3 aus. px zeigt auf, welches auf x[0] zeigt.
 ("3 3")

Zweites printf. Für *x wieder x[0] also 3. px
 wurde inkrementiert. Zeigt also auf die nächste
 Speicheradresse *(x+1) also x[1] 2.
 ("3 2")

Skizze: Initialisierung



Ausgabe: 20 21



Probleme: ungewollte Veränderungen von anderen Variablen, wenn man nicht aufpasst mit Speicheradressen. Z.B. $*(\text{px}--)$ ist nicht $*(--\text{px})$. Oder wenn px auf

nichts zeigt.

6) myStruct gibt 14 aus. Da $\text{sizeof}(\text{char}) = 1$.

Somit $10 \cdot 1 + 1 + 1 + \text{sizeof}(\text{short int}) = 2 = 14$.

myUnion gibt 12 aus. Größtes benötigtes Element

char a[12] = 12 · 1 = 12

7) Mit define kann man in C Makros definieren. (Häufig verwendet für Konstanten).

Syntax: `#define KONSTANTEN-NAME value.`

Ausgabe: 15.