KSP - Tutorium

Test 4-Korr
Africa‘s Geek

KSP

Donchi Fofack Donald

Test für die Klausurvorbereitung – KSP

Nachname:	
Vorname:	
Betreuer:	Donchi Fofack Donald

	Max. Punktzahl	erreicht
Aufgabe 1	17	
Aufgabe 2	20	
Aufgabe 3	20	
Aufgabe 4	10	
Aufgabe 5	35	
Gesamt	102	

Aufgabe 1: (17 Punkte)

I- Schreiben Sie C-Deklarationen für folgende Aussagen:

a) eine möglichst große ganze Zahl z
long z;

b) ein Feld a von 20 Zeichen
char a[20];

c) ein Feld b von 10 Zeigern auf ganze Zahlen
int *b[10];

d) eine Funktion f, die ein Feld von ganzen Zahlen ohne Vorzeichen und dessen Größe erwartet und einen Zeiger auf ein Element dieses Feldes zurückgibt

unsigned int *f(unsigned int c[], int groesse);

e) ein “Byte” (vorzeichenloses Zeichen) b
unsigned char b;

f) ein Feld a von 4 Zeigern auf vorzeichenlose ganze Zahlen
unsigned int *a[4];

g) eine Funktion f, die zwei ganze Zahlen erwartet, und einen Zeiger auf eine ganze Zahl zurückgibt.
Int *f(int a, int b);

h) einen Zeiger p auf eine Funktion, die einen Zeiger auf ein Zeichen erwartet und einen Zeiger auf ein Zeichen zurückgibt.
Char * (*p) (char *a);

II- Es sei vorgegeben:

```
int a[] = {1, 3, 5, 7, 2, 4, 6, 8};
int *p = &a[4];
int **q = &p;

int b[] = {10, 8, 15, 9, 1, 2, 12};
int *k = &b[3];
```

Geben Sie die Werte der folgenden Ausdrücke an:

a) a[3]+a[4] //9

b) *(p – 2) //5

c) *p + 2 //4

d) *(*q – 1) – 3 //4

e) b[5] //2

f) b[b[6] – b[0]] //15
g) *k-1 //8

h) *(k-1)//15

I) &b[6] – k //3

Aufgabe 2: (20 Punkte)

I-

Es sollen ganze Zahlen in einer doppelt verketteten Liste abgelegt werden. Zur Erinnerung: ein Knoten einer doppelt verketteten Liste hält jeweils einen Zeiger auf seinen Vorgänger und auf seinen Nachfolger (und speichert hier eine ganze Zahl).

a) Definieren Sie einen Typ "Node" zum Speichern eines Knotens einer solchen Liste.

```
typedef struct node
{
    struct node *next;
    struct node *prev;
    int value;

}Node;
```

b) Definieren Sie einen Zeiger p, der auf eine solche Liste zeigen kann.

```
Node *p = malloc(sizeof(Node));
```

II-

Gegeben ist Folgende Datenstruktur einer doppelt verketteten Liste:

```
typedef struct element {
        struct element * left;
        struct element * right;
        int value;
} ListElement;
```

left muss dabei immer auf das Element links in der Liste vom aktuell betrachteten Element zeigen (oder beim ersten Element auf NULL).
right muss immer auf das Element rechts neben dem aktuell betrachteten Element in der Liste zeigen (oder auf NULL beim letzten Element).

a. Implementieren Sie folgende Funkiton, die einen Wert in eine Liste einfügen soll. Der einzufügende Wert value soll anschließend an n-ter Stelle in der Liste stehen. Der Rest der Liste soll erhalten bleiben, wird also um eine Stelle nach rechts verlegt. Achten Sie darauf jedes neue Listen Element auf dem Heap anzulegen.

```
void insert_value(ListElement* list, int value, int n)
{
```

(Todo)

Aufgabe 3: Wissensfragen(20 Punkte)

A- Betrachten Sie die Bigint-Bibliothek unter Berücksichtigung des vorgegebenen Garbage Collection Verfahrens der Ninja VM. Mit welchen Problemen hätte eine Implementierung der Garbage Collection zu tun, würde die Schnittstelle der Bigint Bibliothek, anstelle der Big Integer Processor (bip) Variable, Übergabeparameter und Rückgabewerte vom Typ ObjRef für die Funktionsaufrufe definieren?

Beispielhaft gezeigt also: ObjRef result = bigAdd(a, b);
Beschreiben Sie ausführlich.

B- Beschreiben Sie die drei Phasen des Stop & Copy Garbage Collection Verfahrens.
(korrektur: siehe woche10.pdf)
C- Welche Aufgabe hat der Stack Pointer?
(korrektur: siehe woche10.pdf)
D- Wozu dient in C der tag bei einem Struct oder einer Union?

(korrektur: siehe woche10.pdf)

Aufgabe 4: (10 Punkte)

Programmierer Schlaufuchs muss häufig das Produkt aus zwei Faktoren berechnen. Er definiert dazu einen Makro:

```
#define MUL(a,b) a*b
```

a) Zu welchem Ausdruck wird der Aufruf MUL(2,3) expandiert und was ist dessen Wert?

b) Zu welchem Ausdruck wird der Aufruf MUL(5-3,1+2) expandiert und was ist dessen Wert?

c) Helfen Sie dem armen Schlaufuchs mit einer korrekten Definition des Makros aus der Patsche!

a) Es wird expandiert zu: 2*3 und der Wert ist 6
b) Es wird expandiert zu: 5-3*1+2 und der Wert ist 5-3+2=4
c) #define MUL(a,b) ((a)*(b))

Aufgabe 5: (35 Punkte)

Schreiben Sie in Ninja Assembler Folgendes Programm:

A- Schreiben Sie eine Funktion mit dem Namen print_n_times, die zwei Zahlen als Parameter entgegen nimmt (n und p) und eine Zahl zurück gibt. (10 Punkte)

Die Funktion schreibt n Zeilen mit der Zahl p in die Standardausgabe. Der Rückgabewert entspricht der Summe beider Parameter. Im Fall, dass n negativ ist, wird nichts ausgegeben und der Rückgabewert ist -1.

Implementieren Sie außerdem einen Funktionsaufruf in Ninja Assembler. Dabei rufen Sie die Funktion mit den Parametern n=3 und p=25 auf und schreiben Sie den Rückgabewert in die Standardausgabe.

Die Ausgabe sollte also folgendermaßen aussehen:
25
25
25
28

(Korrektur: siehe alte Klausur)
B- schreieben Sie bitte funk1, funk2, funk3, main und funk4 in assembler.

```
void funk1(){
    writeInteger(12);
    writeCharacter('\n');
}

void funk2(Integer a, Integer b){
    writeInteger(a%b);
    writeCharacter('\n');
}

Integer funk3(){
    return 2021;
}

Integer funk4(Integer a, Integer b){
    local Integer c;
    c= a / b;
    return c + 10;
}

void main(){
    local Integer a;
    local Integer b;
    funk1();
    funk2(1,2);
    a = funk3();
    b = funk4(1,2);
    writeInteger(b%a);
    writeCharacter('\n');
}
```

Korrektur:

```
//
// void funk1()
//
_funk1:
    asf 0
    pushc   12
    wrint
    pushc   10
    wrchr
__0:
    rsf
    ret
```

```
//
// void funk2(Integer, Integer)
//
_funk2:
    asf 0
    pushl   -4
    pushl   -3
    mod
    wrint
    pushc   10
    wrchr
__1:
    rsf
    ret
```

```
//
// Integer funk3()
//
_funk3:
    asf 0
    pushc   2021
    popr
    jmp __2
__2:
    rsf
    ret
```

```
_main:
    asf 2
    call    _funk1
    pushc   1
    pushc   2
    call    _funk2
    drop    2
    call    _funk3
    pushr
    popl    0
    pushc   1
    pushc   2
    call    _funk4
    drop    2
    pushr
    popl    1
    pushl   1
    pushl   0
    mod
    wrint
    pushc   10
    wrchr
__4:
    rsf
    ret
```

```
//
// Integer funk4(Integer, Integer)
//
_funk4:
    asf 1
    pushl   -4
    pushl   -3
    div
    popl    0
    pushl   0
    pushc   10
    add
    popr
    jmp __3
__3:
    rsf
    ret
```