C Exam

A. Nonymous R. E. Dacted

16. Januar 2020

There are 80 *points in total but the time (*120 *minutes) is insufficient to solve all exercises. Chose what exercises you want to solve and try to maximise your points! You need* 30 *points to pass.*

Good luck!

Exercise 1: Programming: The Sieve of Eratosthenes (30 **P)**
Write a program that takes a number as a command line argument and returns all primes up to and including that number. Use the *Sieve of Eratosthenes* to implement this functionality. The Sieve of Eratosthenes computes all prime numbers from 2 up to a number n as follows:

1. Write down all integers from 2 to n.
2. Take the smallest integer on the list that has not been struck out and print out that it is prime.
3. Strike out that number and all its multiples.
4. Repeat steps 2–3 until all numbers have been struck out.

Take care of good error handling. Print each prime on its own line.

Listing 1: Sample execution and output

```
$ ./sieve 23
2
3
5
7
11
13
17
19
23
```

Exercise 2: Calling the C Compiler $(3+2$ **P)**

a) Give options for the C compiler to run just the preprocessor, to generate assembly files, and to generate object files.

b) Give a compiler invocation that createst an executable file `test_program` from the source file `test.c` using the math library `-lm`.

Exercise 3: Syntax $(3+4$ **P)**

a) Briefly describe the difference between `for`, `while` and `do ... while` loops.

b) Breifly describe the jump statemenets `goto`, `break`, `return`, and `continue`.

Exercise 4: Storage Classes $(3+3+3$ **P)**

a) Briefly describe the difference between **automatic**, **static**, and **dynamic** objects.

b) For each storage class, name a use case where using an object of that storage class is useful.

c) For each storage class, explain how to obtain a pointer to an object of that storage class.

Exercise 5: Operator Precedence $(2+2+2+2$ **P)**
The C language has the following operators:

```
1 ! != % %= & && &= () (type) * *= + += ++ , - -- -= -> . /
2 /= < << <<= <= = == > >> >>= >= ?: [] ^ ^= sizeof | || |=
```

Add explicit parentheses to the following statements to highlight the precedence of the operators used within. For example:

```
1 a = (((b / c) + (d * e)));
```

Additionally, give the value of *all variables* after execution. Before execution, the variables have values

```
1   int a = 0, b = 1, c = 2, d = 3;
```

Listing 2: Expressions

```
// (a)
a = b++ * c + d / c * c;

// (b)
a = a == b || c == --d && b = d--;

// (c)
a = a && b == c | 0xF << d - 1;

// (d)
a = d -= b, d - b + a;
```

Exercise 6: Code Review $(2 + 2 + 2$ **P)**
Find errors in the program text below. Mark up to 3 problematic sections and explain for each why the code is wrong.

```
1  void bar()
2  {
3    return (42);
4  }
5
6  void main(int argc, char **argv)
7  {
8    int a;
9    if (argv[1] == "foo") {
10     a = atoi(argv[2]);
11   } else if (argv[1] == "bar") {
12     a = bar();
13   }
14
15   printf(b/a);
16 }
```

Exercise 7: Reading Declarators $(2 + 2$ **P)**
For the following function pointer `foo`, give:

- the types of the arguments and
- the type of the return value
- den Typen des/der Argumente und

```
1 void (*foo(size_t n, size_t (*bar)(size_t, void *)))(void);
```

Exercise 8: Memory Layout: Linked List (5 P)
Visualise the memory regions and Pointers used in this program and the point of the comment.

```
1  struct linked_list {
2      struct linked_list *next;
3      int element;
4  };
5
6  struct linked_list _head = {
7      .next = NULL,
8  };
9  struct linked_list *const HEAD = &_head;
10
11 int main(void) {
12     struct linked_list *a = malloc(sizeof (a));
```

```
13      if (!a) {
14          return (1);
15      }
16      a->element = 42;
17      a->next = NULL;
18      HEAD->next = a;
19
20      // insightful comment
21
22      free(a);
23  }
```

Exercise 9: Documentation (2 + 2 **P**)
Imagine you try to understand a C program. In that C program you find a call to the unknown function `mkstemp`. How could you find out what this function does? Name two approaches.

Exercise 10: External Linkage (2 **P**)
You want to use a variable `a` defined in `a.c` in file `b.c`. Add appropriate declarations and definitions to both files to allow this.

Listing 3: File a.c

```
1
2
3
4  int foo(void)
5  {
6      a += 1;
7  }
```

Listing 4: File b.c

```
1
2
3  int foo(void);
4
5  int main(void)
6  {
7      a = 42;
8      foo();
9      printf("%d\n", a);
10 }
```