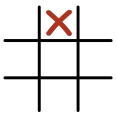
Programming 2 (SS 2023)
Saarland University
Faculty MI
Compiler Design Lab

Sample Solution 4
C

Prof. Dr. Sebastian Hack
Pascal Lauer, M. Sc.
Marcel Ullrich, B. Sc.

The calendar indicates which script chapters you should study in conjuction with each lecture. The exercises are designed to enhance your understanding of the lecture material and prepare you for the mini-tests and the final exam. Additional exercises can be found at the end of each chapter in the script.
The difficulty of an exercise on the sheet is determined by the number of annotated 'X' and 'O' marks in the tic-tac-toe field, with four levels (1-4) increasing by one mark per level.

Exercise 4.1: Mixed Bag



This exercise is there to help you get acquainted with important terms and concepts from the script.

1. Find some C program of your choice and determine all identifiers and keywords. Which different types can you find in the program?
2. What is understood by the term *block*? What are blocks used for?
3. How can we find out, how many bytes the base types `int`, `short`, `long` and `float` occupy in the virtual machine? Test your idea!
4. Why is `&x` not L-evaluable?

Solution

1. -
2. Blocks are a sequence of statements enclosed by curly braces. They open a new scope where variables can be declared. The advantage of blocks is that using them, one can use variables with the same name in many different functions without those variables influencing each other.
3. To determine the size of types, one uses `sizeof`.

```
printf("int: %lu\n", sizeof(int));
printf("short: %lu\n", sizeof(short));
printf("long: %lu\n", sizeof(long));
printf("float: %lu\n", sizeof(float));
```

Open in GitLab [1]

We use `lu` in `printf` to output a *long unsigned int*, because `sizeof` returns this type (`i` also works, but produces a warning).

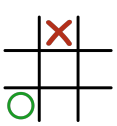
4. We assume that we have the following program:

```
int a;
&a = 1;
```

Open in GitLab [2]

The evaluation of `&a` results in the address of `a`. One would now try to overwrite this with 1, which doesn't make sense. The address describes a place in memory and is not just changable by simple overwriting.

[1] https://git.prog2.de/staff/exercise_sheets/-/blob/master/C/c-mixed-questions/code_0.c
[2] https://git.prog2.de/staff/exercise_sheets/-/blob/master/C/c-mixed-questions/code_1.java



Exercise 4.2: pointers everywhere

1. Konrad Klug and No Hau learned today what pointers in C are. Therefore they started right away and wrote their first programs in which they experimented with pointers. Unfortunately, syntax errors snuck into Konrad's first assignment block. Find out which is the incorrect assignment block and help Konrad by marking the incorrect places.

```c
int x = 20;
int y = 11;
int* px = &x;
int *py = &y;
```

Open in Browser Open in GitLab [3]

```c
int x = 20;
int y = 11;
*int px = &x;
int py* = &y;
```

Open in Browser Open in GitLab [4]

2. After Konrad has found his errors with your help, he continues to work on his program. Look at the following table with Konrad's program and complete it. First decide for each assignment whether it is valid or not. If the assignment is valid, indicate what state the variables are in after the assignment. Assume at the beginning of the task the valid assignment block from task part (a) and at the beginning of each line the state of the last valid one.

	assignment	x	y	px	py
0.	-	20	11	0x004	0x008
1.	`*px = 7;`	7	11	0x004	0x008
2.	`&px = &py;`			invalid	
3.	`px = &y;`				
4.	`&px = *x;`				
5.	`*px = 5;`				
6.	`*x = *py;`				
7.	`y = y + x;`				
8.	`*x = *y;`				
9.	`x = (px == py);`				
10.	`&x = py;`				
11.	`x = x + y;`				
12.	`px = &x;`				
13.	`x = *(px + 4);`				

3. Because No Hau has noticed that Konrad has even more difficulty with pointers than you do, he has designed the following quiz. Konrad thinks he can answer all the questions correctly. Can you do the same?

[3] https://git.prog2.de/staff/exercise_sheets/-/blob/master/C/basic-pointer-en/code_0.c
[4] https://git.prog2.de/staff/exercise_sheets/-/blob/master/C/basic-pointer-en/code_1.c

question 1: What is the type of the variables? Are all four declarations valid?

(a) `int* a, b;`

(b) `int *a, b;`

(c) `int *a, *b;`

(d)

```
1 int  x = 11;
2 int  *px = &x;
3 int* *a = &px;
```

Open in Browser Open in GitLab [5]

question 2: What outputs do the following expressions produce? Assume that `px` and `a` were defined analogously to 4. from question 1.

(a) `printf("%d", *px);`

(b) `printf("%d", *a);`

question 3: Are the following two expressions valid? If so, what do they evaluate to? Assume again that `px` and `a` were defined analogously to 4. from question 1.

(a) `x = (&px == (a + 4));`

(b) `x = (&(px + 4) == (a + 4));`

Solution

1. The 1st block is valid! In the second block, the last two lines are not valid assignments.

2.

	assignment	x	y	px	py
0.	-	20	11	0x004	0x008
1.	`*px = 7;`	7	11	0x004	0x008
2.	`&px = &py;`	invalid			
3.	`px = &y;`	7	11	0x008	0x008
4.	`&px = *x;`	invalid			
5.	`*px = 5;`	7	5	0x008	0x008
6.	`*x = *py;`	invalid			
7.	`y = y + x;`	7	12	0x008	0x008
8.	`*x = *y;`	invalid			
9.	`x = (px == py);`	1	12	0x008	0x008
10.	`&x = py;`	invalid			
11.	`x = x + y;`	13	12	0x008	0x008
12.	`px = &x;`	13	12	0x004	0x008
13.	`x = *(px + 4);`	undefined behavior			

3. **question 1:** All four declarations are valid.

   1. `a: int*, b: int`
   2. `a: int*, b: int`
   3. `a: int*, b: int*`
   4. `x: int , px: int* , a: int**`

   question 2:

   (a) `printf("%d", *px);` outputs 11

[5] `https://git.prog2.de/staff/exercise_sheets/-/blob/master/C/basic-pointer-en/code_2.c`

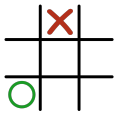
(b) `printf("%d", *a);` outputs the address where 11 is stored

question 3:

(a) `x = (&px == (a + 4));` valid, x evaluates to 0 since `a == &px` and thus `(a+4) != &px`.

(b) `x = (&(px + 4) == (a + 4));` is not a valid expression because `px+4` is not an L-value.

Exercise 4.3:



Determine the validity of each statement in C. If a statement is found to be invalid, mark it and continue examining the remaining code.

```
 1 int i, b;
 2 char c = 8;
 3 c = i;
 4 void x = 2;
 5 char* d;
 6 d = i;
 7 d = "Types␣are␣fun";
 8 i = d[0] + 2;
 9 d = (1==2);
10 int* x = 0;
11 char* y;
12 y = 1;
13 d = x | y;
14 void* e;
15 x = (int*) e;
16 int a = (b = 3) + 1;
17 int f = (int)("The␣End?"[0]);
```

Open in Browser Open in GitLab [6]

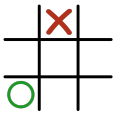
[6] https://git.prog2.de/staff/exercise_sheets/-/blob/master/C/typequiz/code_0.c

Solution

Line	Valid?	Explanation
1	yes	Variable declaration. Variable has complete type.
2	yes	Variable declaration with initialization. Variable has complete type. Type `int` (for the constant 8) is implicitly convertible to `char`.
3	yes	Assignment. Type `int` is implicitly convertible to `char`.
4	no	`void` is an incomplete type. A variable of incomplete type may not be declared.
5	yes	Variable declaration. Variable has complete type `char*`.
6	no	Assignment. Type `int` is not implicitly convertible to `char*`. Conversion requires an explicit cast (see also C standard 6.5.4p3)
7	yes	Assignment. String literal has type `char[24]`. Type is implicitly converted to `char*` (see also C standard 6.3.2.1p3), which is the type of the variable that is assigned to.
8	yes	Right hand side expression has type `int`, because we add up a `char` (`d[0]`) and an `int` (2), which are convertable.
9	no	Right hand side expression has type `int`, which is not implicitly convertible to `char*`.
10	yes	Variable declaration with complete type. Initialization is valid since `0` is the null pointer constant with type convertible to `int*`.
11	yes	Variable declaration with complete type.
12	no	Assignment: Conversion from `int` to `char*` requires explicit cast.
13	no	Expression has type `int`. Expression is well-typed but conversion from `int` to `char*` requires explicit cast.
14	yes	Variable declaration with complete type. Every pointer type is a complete type, even `void*`.
15	yes	Assignment: `void*` is implicitly convertible into any other pointer type (here: `int*`). The cast is not even needed.

Line	Valid?	Explanation
16	yes	Variable declaration with complete type `int`. The assignment `b=3` is well-typed since both `b` and 3 have type `int`. It has the type of the left hand side `int`. Hence, the initializer expression also has type `int`, which coincides with the type of the declared variable.
17	yes	Declaration with complete type. String literal is implicitly converted to type `char*`. Casted expression therefore has type `char` and is casted to `int`, which is valid. Initialization is valid since the righthand side also has type `int`.

Exercise 4.4:



Take a look at the following C function:

```
1 void foo() {
2   int *w;
3   int **y;
```

```
 4   int x[2];
 5   int z;
 6   z = 13;
 7   w = x;
 8   y = &w;
 9   *x = z;
10   *(x+1) = 42;
11   z = *w;
12 }
```

Open in Browser Open in GitLab [7]

1. For every expression and subexpressions, write down whether L-evaluation or R-evaluation occurs.
2. List all the expressions that can be L-evaluated. Additionally, identify which expressions can also be R-evaluated?
3. Complete the following execution trace. For every line, shortly describe the L- or R-evaluation.

Line	`w`	`x[0]`	`x[1]`	`y`	`z`	
2						
3	?					Initialization of w
4	?			?		Initialization of y
5	?	?	?	?		Initialization of x
6	?	?	?	?	?	Initialization of z
7	?	?	?	?	13	L-evaluation of variable z gives the address of the container. The R-evaluation of the constant 13 gives the value 13. Therefore, 13 is written into the container of z.
8	`&x[0]`	?	?	?	13	...
⋮	⋮	⋮	⋮	⋮	⋮	⋮

Solution

1.
   - `z = 13`: `z` is L-evaluated and 13 is R-evaluated.
   - `w = x`: `w` is L-evaluated and `x` is R-evaluated.
   - `y = &w`: `y` is L-evaluated, `&w` is R-evaluated and `w` is L-evaluated.
   - `*x = z`: `*x` is L-evaluated, `x` is R-evaluated and `z` is R-evaluated
   - `*(x+1) = 42`: `*(x+1)` is L-evaluated, `x` is R-evaluated, `x+1` is R-evaluated and `42` is R-evaluated.
   - `z = *w`: `z` is L-evaluated and 13 is R-evaluated, `*w` is R-evaluated and `w` is R-evaluated.
2. `w, y, z, *w, *x, *(x+1)` are all L-evaluable expression that occur in the program. All expressions are R-evaluable. `x` is an array an therefore not L-evaluable anywhere in the program. Arrays behave a little weird in C. Contrary to popular belief, arrays are not just a pointer to the first element. Array values are L-evaluable, but will almost always be converted to a pointer to the first element when used in the program, which is not L-evaluable anymore. Compare C11 Standard Section 6.3.2.1 (3).

[7] `https://git.prog2.de/staff/exercise_sheets/-/blob/master/C/l-r-evaluation/code_0.c`

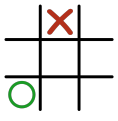
3.

Line	w	x[0]	x[1]	y	z	
2						
3	?					Initialization of w
4	?			?		Initialization of y
5	?	?	?	?		Initialization of x
6	?	?	?	?	?	Initialization of z
7	?	?	?	?	13	L-evaluation of variable z gives the address of the container. The R-evaluation of the constant 13 gives the value 13. Therefore, 13 is written into the container of z.
8	`&x[0]`	?	?	?	13	The L-evaluation of the variable w gives the address of the container of w. The R-evaluation of x gives an array value, which is converted to the address of the first element of the array x is bound to. The address is written into the container of w.
9	`&x[0]`	?	?	`&w`	13	The L-evaluation of the variable y gives the address of the container of y. The expression is L-evaluable: The R-evaluation of the expression `&` gives the L-evaluation of w, which is the address of the container of w.
10	`&x[0]`	13	?	`&w`	13	The L-evaluation of the expression `*x` is the result of the R-evaluation of x, which is the base address of the corresponding array after conversion. The R-evaluation of z gives the content of the container which one receives when L-evaluating z. This is the value 13. Hence, the value 13 is written into the first container of the array that x is bound to.

Line	w	x[0]	x[1]	y	z	
11	`&x[0]`	13	42	`&w`	13	The L-evaluation of the expression `*(x+1)` is the result of the R-evaluation of `x+1`, which is the base address of the array x is bound to plus `sizeof(int)`. This results in the address of the second container of the array. The R-evaluation of the constant 42 yields the value 42. Hence, the value 42 is written into the second container of the array x is bound to.
12	`&x[0]`	13	42	`&w`	13	The L-evaluation of z gives the address of the container of z. The expression `*w` is L-evaluable and gives the R-evaluation of w, which is the contained value in the container of w, i.e. the base address of the array x is bound to. The R-evaluation of a L-evaluable expression reads the content of the container, of which the L-evaluation gives the address. Hence, the content of the first container that belongs to the array x is bound to is written into the container of z.

Exercise 4.5:

Calculate by hand which output the following program prints. Only use a computer to validate your solution.



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

```c

int main(int argc, char* argv[]) {
    int aa[6] = { 50, 60, 70, 80, 90, 100 };
    int bb[6] = { -2, 2, 0, 1, -1, 3 };

    int *a, **b, *c, **d, *e;

    c = & bb[5]; //9
    d = &c; //10
    a = bb; //11
    b = &a; //12

    for (e = *b; e <= *d; e++) { //142
        *e = *(aa + 3 - *e);
    }

    printf("%d␣%d␣%d␣%d␣%d␣%d\n", bb[0], bb[1], bb[2], bb[3], bb[4], bb[5]);

    return 0;
}
```

Open in Browser Open in GitLab [8]

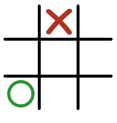
Solution

Output of the program:

Step	Line	a	b	c	d	e	aa	bb
1	4	-	-	-	-	-	-	-
2	5	-	-	-	-	-	[50, 60, 70, 80, 90, 100]	-
3	7	-	-	-	-	-	[50, 60, 70, 80, 90, 100]	[−2, 2, 0, 1, −1, 3]
4	9	?	?	?	?	?	[50, 60, 70, 80, 90, 100]	[−2, 2, 0, 1, −1, 3]
5	10	?	?	&bb[5]	?	?	[50, 60, 70, 80, 90, 100]	[−2, 2, 0, 1, −1, 3]
6	11	?	?	&bb[5]	&c	?	[50, 60, 70, 80, 90, 100]	[−2, 2, 0, 1, −1, 3]
7	12	&bb[0]	?	&bb[5]	&c	?	[50, 60, 70, 80, 90, 100]	[−2, 2, 0, 1, −1, 3]
8	14	&bb[0]	&a	&bb[5]	&c	?	[50, 60, 70, 80, 90, 100]	[−2, 2, 0, 1, −1, 3]
9	15	&bb[0]	&a	&bb[5]	&c	&bb[0]	[50, 60, 70, 80, 90, 100]	[−2, 2, 0, 1, −1, 3]
10	14	&bb[0]	&a	&bb[5]	&c	&bb[0]	[50, 60, 70, 80, 90, 100]	[100, 2, 0, 1, −1, 3]
11	15	&bb[0]	&a	&bb[5]	&c	&bb[1]	[50, 60, 70, 80, 90, 100]	[100, 2, 0, 1, −1, 3]
12	14	&bb[0]	&a	&bb[5]	&c	&bb[1]	[50, 60, 70, 80, 90, 100]	[100, 60, 0, 1, −1, 3]
13	15	&bb[0]	&a	&bb[5]	&c	&bb[2]	[50, 60, 70, 80, 90, 100]	[100, 60, 0, 1, −1, 3]
14	14	&bb[0]	&a	&bb[5]	&c	&bb[2]	[50, 60, 70, 80, 90, 100]	[100, 60, 80, 1, −1, 3]
15	15	&bb[0]	&a	&bb[5]	&c	&bb[3]	[50, 60, 70, 80, 90, 100]	[100, 60, 80, 1, −1, 3]
16	14	&bb[0]	&a	&bb[5]	&c	&bb[3]	[50, 60, 70, 80, 90, 100]	[100, 60, 80, 70, −1, 3]
17	15	&bb[0]	&a	&bb[5]	&c	&bb[4]	[50, 60, 70, 80, 90, 100]	[100, 60, 80, 70, −1, 3]
18	14	&bb[0]	&a	&bb[5]	&c	&bb[4]	[50, 60, 70, 80, 90, 100]	[100, 60, 80, 70, 90, 3]
19	15	&bb[0]	&a	&bb[5]	&c	&bb[5]	[50, 60, 70, 80, 90, 100]	[100, 60, 80, 70, 90, 3]
20	14	&bb[0]	&a	&bb[5]	&c	&bb[5]	[50, 60, 70, 80, 90, 100]	[100, 60, 80, 70, 90, 50]
21	18	&bb[0]	&a	&bb[5]	&c	&bb[6]	[50, 60, 70, 80, 90, 100]	[100, 60, 80, 70, 90, 50]
22	20	&bb[0]	&a	&bb[5]	&c	&bb[6]	[50, 60, 70, 80, 90, 100]	[100, 60, 80, 70, 90, 50]

Exercise 4.6:

Try to comprehend every single statement. Does this program produce the output “56-42-13”?



[8] https://git.prog2.de/staff/exercise_sheets/-/blob/master/C/pointergolf/code_0.c

```c
#include <stdio.h>

int main(int argc, char* argv[]) {
    int x = 56;
    int *ap, *bp;
    int **app, **bpp;

    int r[3];

    r[0] = 1;
    r[1] = 2;
    r[2] = 3;

    ap = &r[1];
    bp = &r[2];

    app = &ap;
    bpp = &bp;

    *ap = 43;
    *(*(bpp)) = 13;
    ap++;
    bp -= 2;
    *(bp + 1) = 17;
    ap = &x;
    app = &bp;
    *bp = *ap;
    bpp = &ap;
    ap = &r[2];
    *((*bpp) - 1) = 42;

    printf("%d-%d-%d\n", r[0], r[1], r[2]);
}
```

Open in Browser Open in GitLab [9]

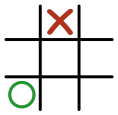
Solution

Output of the program: “56-42-13”.

[9] https://git.prog2.de/staff/exercise_sheets/-/blob/master/C/pointerchase/code_0.c

Step	Line	x	ap	bp	app	bpp	r[0]	r[1]	r[2]
1	4	–	–	–	–	–	–	–	–
2	5	56	–	–	–	–	–	–	–
3	6	56	?	?	–	–	–	–	–
4	8	56	?	?	?	?	–	–	–
5	10	56	?	?	?	?	?	?	?
6	11	56	?	?	?	?	1	?	?
7	12	56	?	?	?	?	1	2	?
8	14	56	?	?	?	?	1	2	3
9	15	56	&r[1]	?	?	?	1	2	3
10	17	56	&r[1]	&r[2]	?	?	1	2	3
11	18	56	&r[1]	&r[2]	&ap	?	1	2	3
12	20	56	&r[1]	&r[2]	&ap	&bp	1	2	3
13	21	56	&r[1]	&r[2]	&ap	&bp	1	43	3
14	22	56	&r[1]	&r[2]	&ap	&bp	1	43	13
15	23	56	&r[2]	&r[2]	&ap	&bp	1	43	13
16	24	56	&r[2]	&r[0]	&ap	&bp	1	43	13
17	25	56	&r[2]	&r[0]	&ap	&bp	1	17	13
18	26	56	&x	&r[0]	&ap	&bp	1	17	13
19	27	56	&x	&r[0]	&bp	&bp	1	17	13
20	28	56	&x	&r[0]	&bp	&bp	56	17	13
21	29	56	&x	&r[0]	&bp	&ap	56	17	13
22	30	56	&r[2]	&r[0]	&bp	&ap	56	17	13
23	32	56	&r[2]	&r[0]	&bp	&ap	56	42	13

Exercise 4.7:



Write a C program which can be used as a basis to implement a calendar. Your program shall:

1. Declare a `struct` in which an event can be stored. An event consists of a title and a date (for the sake of simplicity, we ignore start and end times here).
2. Declare a `struct` representing a date. A date consists of a day, month and year.
3. Add the option to read appointments via the console. A date is always read in the order: day, month, year.
4. Be able to print a date to the console.
5. *Challenge*: Organize all events in a data structure of your choice and print them all in chronological order.
6. *Challenge*: Write all events stored in the data structure to a file and read all events saved in the file at the beginning of the program.

Solution

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

// b.
typedef struct {
    int year;
    int month;
    int day;
} Date;

```

```c
// a.
typedef struct {
    Date *date;
    char *title;
} Event;

Date* init_Date(int day, int month, int year) {
    Date *date = malloc(sizeof(Date));
    date->year = year;
    date->month = month;
    date->day = day;
    return date;
}

Event* init_Event(char *title) {
    Event *event = malloc(sizeof(Event));
    event->title = title;
    return event;
}

void destroy_Event(Event *event) {
    free(event->date);
    free(event->title);
    free(event);
}

// d. (pass stdout to print to console)
void print_Event(Event *event, FILE* fp) {
    fprintf(fp, "%02d.%02d.%04d %s\n", event->date->day,
            event->date->month, event->date->year, event->title);
}

// c. (pass stdin to read from console)
Event *read_Event(FILE* fp) {
    int day, month, year;
    Event *event = init_Event(malloc(sizeof(char) * 256));
    int i = fscanf(fp, "%d.%d.%d %255[^\n]s", &day, &month, &year, event->title);
    if (i != 4) {
        free(event->title);
        free(event);
        return NULL;
    };
    event->date = init_Date(day, month, year);
    return event;
}

//Begin Challenge 1

typedef struct list {
    Event* event;
    struct list* next;
} List;

List* read_Events(List* old_list, FILE* fp) {
    int i;
    if (fp == stdin) {
        printf("How many events? ");
        if (scanf("%d", &i) != 1) {
            return NULL;
        }
    } else {
```

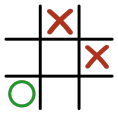
```
73          i = 0;
74          int c;
75          while ((c = fgetc(fp)) != EOF) {
76              if (c == '\n') i++;  // #events
77          }
78          rewind(fp);  // need to reset position in file
79      }
80      List* list = NULL;
81      List **tail = &list;
82      for (; i > 0; i--) {
83          *tail = malloc(sizeof(List));
84          (*tail)->event = read_Event(fp);
85          (*tail)->next = NULL;
86          if ((*tail)->event == NULL)
87              return NULL;
88          tail = &(*tail)->next;
89      }
90      if (old_list != NULL) {
91          *tail = old_list;
92      }
93      return list;
94  }
95
96  int compare_Date(Date* a, Date* b) {
97      if (a->year < b->year) {
98          return -1;
99      }
100     if (a->year > b->year) {
101         return 1;
102     }
103     if (a->month < b->month) {
104         return -1;
105     }
106     if (a->month > b->month) {
107         return 1;
108     }
109     if (a->day < b->day) {
110         return -1;
111     }
112     return a->day > b->day;
113 }
114
115 //clever pointer-to-pointer magic that properly unlinks
116 Event *unlink_max(List** delete_entry) {
117     List *list = *delete_entry;
118     if (!list) return NULL;
119
120     Event* max = list->event;
121     List* last = list;
122     list = list->next;
123     while (list) {
124         if (compare_Date(list->event->date, max->date) > 0) {
125             max = list->event;
126             delete_entry = &last->next;
127         }
128         last = list;
129         list = list->next;
130     }
131     if (max != NULL) {
132         List* tofree = *delete_entry;
133         (*delete_entry) = (*delete_entry)->next;
```

```
134         free(tofree);
135     }
136     return max;
137 }
138
139 List* bubblesort(List* in) {
140     List* newlist = NULL;
141     while (in) {
142         Event* event = unlink_max(&in);
143         List* next = newlist;
144         newlist = malloc(sizeof(List));
145         newlist->next = next;
146         newlist->event = event;
147     }
148     return newlist;
149 }
150
151 void print_List(List* list, FILE* fp) {
152     for (;list;list = list->next) {
153         Event* event = list->event;
154         print_Event(event, fp);
155     }
156 }
157
158 void destroy_List(List* list) {
159     List* next;
160     while (list) {
161         next = list->next;
162         destroy_Event(list->event);
163         free(list);
164         list = next;
165     }
166 }
167
168 //End Challenge 1
169
170 //Begin Challenge 2
171
172 /* we only define the name of the data file here, for the other parts see:
173  * - read_Events for reading the events from the file
174  * - print_List for writing the appointments to the file
175  * Note: read_Events can also read from the console
176  */
177 #define FILENAME "Events.txt"
178
179 //End Challenge 2
180
181 int main() {
182     // read stored appointments in the beginning
183     FILE *fp = fopen(FILENAME, "a+");
184     List* list = read_Events(NULL, fp);
185     fclose(fp);
186
187     list = read_Events(list, stdin); // read console
188     if (!list) {
189         fprintf(stderr, "Error␣while␣reading!\n");
190         return 1;
191     }
192     list = bubblesort(list);
193     print_List(list, stdout);
194
```

```
195     // write all appointments in the end, creating the file if necessary
196     fp = fopen(FILENAME, "w");
197     print_List(list, fp);
198     fclose(fp);
199
200     destroy_List(list);
201     return 0;
202 }
```

Open in Browser Open in GitLab [10]

Exercise 4.8: Genealogy



No Hau wants to create a family tree of her family and their friends. She has created a file in which she has listed the children of each person. To save herself work, she limits herself to the oldest child of each family. Since in her family and their circle of friends every first name is unique, she choses the following file format for her list:

```
famtree example {
    Robert Child: Sandra;
    Hung Child: Gustav;
    Gustav Child: Marvin;
    Sandra Child: Marvin;
}
```

Write a graph structure for No Hau that represents the family tree. You do not need to implement how the file is parsed but may assume that the function `parse_family_tree`, which translates a file into its graph structure, exists.

[10] https://git.prog2.de/staff/exercise_sheets/-/blob/master/C/structs/code_0.c

Solution

```h
#ifndef PAGERANK_PARSER_H
#define PAGERANK_PARSER_H

#include "graph.h"

graph_t *parse_graph (const char *);

#endif
```

parser.h

Open in GitLab [a]

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

#include "parser.h"

graph_t *parse_graph (const char *filename) {
    FILE *file = fopen (filename, "r");
    if (!file) {
        fprintf (stderr, "Given Filename matches no file!\n");
        exit (1);
    }

    char s_name[256];
    char t_name[256];

    int c = fscanf (file, "famtree %s {\n", &s_name[0]);
    graph_t *graph = init_graph (s_name);

    memset(s_name, 0, 256);
    memset(s_name, 0, 256);
    while (1) {
        if ((c = fscanf (file, "%s Child: %s;\n", &s_name[0], &t_name[0])) == EOF) {
            fprintf (stderr, "Error while reading file...\n");
            exit (1);
        }
        if (s_name[0] == '}') break;
        node_t *s_node = get_node_from_graph (graph, s_name);
        node_t *t_node = get_node_from_graph (graph, t_name);
        add_edge_to_node (s_node, t_node);
    }

    fclose (file);
    return graph;
}
```

parser.c

Open in GitLab [b]

[a] https://git.prog2.de/staff/exercise_sheets/-/blob/master/C/advanced_structs/parser.c

[b] https://git.prog2.de/staff/exercise_sheets/-/blob/master/C/advanced_structs/parser.c

```h
#ifndef GRAPH_H
#define GRAPH_H

typedef struct node {
    struct node **out_edges;
    unsigned num_edges;
    char* name;
} node_t;

typedef struct graph {
    /* nodes in the graph */
    node_t **nodes;
    unsigned count;
    char* name;
} graph_t;

graph_t *init_graph(const char *);

void free_graph (graph_t *);

node_t *get_node_from_graph (graph_t *, const char *);

void add_edge_to_node (node_t *, node_t *);

#endif
```

graph.h

Open in GitLab [a]

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <ctype.h>
#include <getopt.h>

#include "parser.h"


int main(int argc, char *const *argv) {
  char *filename = NULL;
  filename = argv[argc - 1];
  graph_t *graph = parse_graph (filename);

  free_graph (graph);
  exit(0);
}
```

main.c

Open in GitLab [b]

[a] https://git.prog2.de/staff/exercise_sheets/-/blob/master/C/advanced_structs/graph.js

[b] https://git.prog2.de/staff/exercise_sheets/-/blob/master/C/advanced_structs/main.c

```c
#include "graph.h"
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

graph_t *init_graph (const char *name) {
    graph_t *graph = malloc (sizeof (*graph));
    if (!graph) {
        fprintf (stderr, "Not enough memory to allocate for graph %s\n", name);
        exit(1);
    }
    graph->nodes = NULL;
    graph->count = 0;
    graph->name = malloc (strlen(name) + 1);
    strcpy(graph->name, name);
    return graph;
}

void free_graph (graph_t *graph) {
    while (graph->count--) {
        node_t *node = graph->nodes[graph->count];
        if (node->num_edges)
            free (node->out_edges);
        free (node->name);
        free (node);
    }
    free (graph->name);
    free (graph->nodes);
    free (graph);
}

node_t *get_node_from_graph (graph_t *graph, const char *name) {
    unsigned tmp = graph->count;
    while (tmp--) {
        node_t *node = graph->nodes[tmp];
        if (!strcmp (node->name, name)){
            return node;
        }
    }
    node_t *node = malloc (sizeof(*node));
    node->out_edges = NULL;
    node->num_edges = 0;
    node->name = malloc(strlen(name) + 1);
    strcpy(node->name, name);
    graph->nodes = realloc (graph->nodes, (graph->count + 1) * sizeof (*graph->nodes));
    graph->nodes[graph->count++] = node;
    return node;
}

void add_edge_to_node (node_t *source, node_t *target) {
    source->out_edges = realloc(source->out_edges, (source->num_edges + 1) * sizeof (*
        source->out_edges));
    source->out_edges[source->num_edges++] = target;
}
```

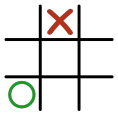
graph.c

Open in GitLab [a]

[a] https://git.prog2.de/staff/exercise_sheets/-/blob/master/C/advanced_structs/graph.c

Exercise 4.9:

Complete the following program so that it prints the array `numbers` in different representations.



```c
#include <stdio.h>

int main(void) {
  int numbers[] = {-13427, -4233, -343, -12, -5, 0,
                   3, 17, 512, 2355, 29367};
  int n = /* Compute the amount of elements in numbers */

  for (int i=0; i < n; ++i) {
    printf("%d␣", numbers[i]); // This line must be changed for each subtask
  }

  return 0;
}
```

Open in Browser Open in GitLab [11]

First, compute the number of elements in the array by use of `sizeof`. Afterwards, print the array in the following representations:

1. In hexadecimal representation with preceding `0x` and padded with zeroes up to size 8, e.g. `0x00000012`. Which type of argument does the format specifier for hexadecimal representation expect?
2. Every number divided by 1000, rounded to 2 decimal places.
3. The numbers as usual decimal numbers, but with the sign of the number in front (i.e. also +).

Every subtask must be solved using the standard library function `printf` and the corresponding formatting string. `printf` is a function with a variable number of arguments, i.e. calls of the form `printf(format, x_1, ..., x_n)` are viable for all n. As the *format*, a string is passed which may contain format specifiers. In the printed output, the i-th format specifier is replaced by the corresponding formatted representation of x_i. A format specifier has the form

```
%[flags][min field width][precision][length]conversion specifier
```

Open in Browser Open in GitLab [12]

where all parameters enclosed in `[]` are optional. A description of the parameters can be found on the manpage for `printf`. You can view it with the command `man 3 printf`.
Remark: You should learn to find the relevant parts and extract the required information from the manpage.

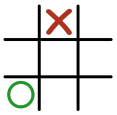
Solution

```c
#include <stdio.h>

int main(void) {
  int numbers[] = {-13427, -4233, -343, -12, -5, 0,
                   3, 17, 512, 2355, 29367};

  int n = sizeof(numbers) / sizeof(numbers[0]);

  for (int i = 0; i < n; ++i) {
    // part (a)
    printf("0x%08x\t", (unsigned int)numbers[i]);
    // part (b)
    printf("%.2f\t", numbers[i] / 1000.0);
    // part (c)
    printf("%+d\n", numbers[i]);
  }
  return 0;
}
```

[11] https://git.prog2.de/staff/exercise_sheets/-/blob/master/C/printf/code_0.c
[12] https://git.prog2.de/staff/exercise_sheets/-/blob/master/C/printf/code_1.c

Open in Browser Open in GitLab [13]

Exercise 4.10:



We want to write a program that computes how much a recipe costs. It gets a file containing a list of items and their prices, and a list of ingredients and the amount needed of each for the recipe. Using this data, it should calculate how much every ingredient costs and also what the recipe costs in total.
The input file has the following format:

```
Prices:
<item count>
<name of item1> : <price of item1 followed by the euro sign>
...

Recipe:
<ingredient count>
<amount of ingredient1> <name of ingredient1>
...
```

Open in GitLab [14]

You may assume that the names don't contain spaces. You can find an example file at the end of this exercise.
You can use the following code as a starting point for your program:

main.c

```c
#include "recipe_structs.h"
#include <string.h>
#include <stdio.h>

int main(int argc, char *argv[]) {

    if (argc != 2) {
        printf("You have to provide an input file.\n");
        return 1;
    }

    FILE *file = fopen(argv[1], "r");

    // TODO: your code

    return 0;
}
```

Open in Browser Open in GitLab [15]

recipe_structs.h

```c
#ifndef RECIPE_STRUCTS_H
#define RECIPE_STRUCTS_H

struct item {
    char name[256];
    float price;
};
```

[13] https://git.prog2.de/staff/exercise_sheets/-/blob/master/C/printf/code_2.c
[14] https://git.prog2.de/staff/exercise_sheets/-/blob/master/C/basic-fscanf/code_0.py
[15] https://git.prog2.de/staff/exercise_sheets/-/blob/master/C/basic-fscanf/main.c

```c
typedef struct item item_t;

struct ingredient {
    char name[256];
    float amount;
};
typedef struct ingredient ingredient_t;

#endif
```

Open in Browser Open in GitLab [16]

data.txt

```
Prices:
9
flour : 0.79€
baking-powder : 0.20€
apples : 0.24€
cheese : 4.24€
butter : 7.48€
sugar : 1.49€
tomatoes : 0.66€
eggs : 0.49€
brown-sugar : 3.38€

Recipe:
7
4 apples
0.250 butter
0.350 flour
0.125 sugar
0.125 brown-sugar
5 eggs
1 baking-powder
```

Open in GitLab [17]

Solution

main.c

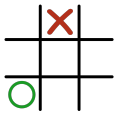
```c
#include "recipe_structs.h"
#include <string.h>
#include <stdio.h>

int main(int argc, char *argv[]) {

    if (argc != 2) {
        printf("You␣have␣to␣provide␣an␣input␣file.\n");
        return 1;
    }

    FILE *file = fopen(argv[1], "r");

    // reading prices
    fscanf(file, "Prices:");
```

[16] https://git.prog2.de/staff/exercise_sheets/-/blob/master/C/basic-fscanf/recipe_structs.c
[17] https://git.prog2.de/staff/exercise_sheets/-/blob/master/C/basic-fscanf/data.py

```c
    int items_count;
    fscanf(file, "%u", &items_count);

    item_t items[items_count];

    for (int i = 0; i < items_count; ++i) {
        fscanf(file, "%s : %f€", items[i].name, &items[i].price);
    }

    // reading recipe
    fscanf(file, "\nRecipe:");
    int ingredients_count;
    fscanf(file, "%u", &ingredients_count);

    ingredient_t ingredients[ingredients_count];

    for (int i = 0; i < ingredients_count; ++i) {
        fscanf(file, "%f %s", &ingredients[i].amount, ingredients[i].name);
    }

    // calculating results
    printf("RESULTS\n");
    printf("-------------\n");
    float total_costs = 0;
    for (int i = 0; i < ingredients_count; ++i) {
        float calc_price;
        for (int j = 0; j < items_count; ++j) {
            if (strcmp(items[j].name, ingredients[i].name) == 0) {
                calc_price = items[j].price * ingredients[i].amount;
            }
        }
        total_costs += calc_price;
        printf("%s: %.2f€\n", ingredients[i].name, calc_price);
    }
    printf("-------------\n");
    printf("TOTAL: %.2f\n", total_costs);

    return 0;
}
```

Open in Browser Open in GitLab [18]

Exercise 4.11: Read and write



Write a C program that reads a list of natural numbers from a file `input.csv`. The file may contain multiple lines. Your program should then produce an output file `output.csv` which has the same format as the input file with one column of numbers squared. If the transformation is successfull, your main method should return 0. In case it fails (e.g. unexpectedly large numbers, wrong console input...), the return value should be 1.
The program should get the index indicating which column to square as command line input. Therefore, the main function needs two arguments `int argc` and `char *argv[]`. You may use the function `atoi` to convert a string to an integer value (will need `stdlib.h`).
After compilation, you should be able to run your program using `./<name> <columnToSquare>`.
Example for input `input.csv` and column 2:

```
5,6,2,7\n
7,3,8,7,3,2\n
6,23\n
```

[18] https://git.prog2.de/staff/exercise_sheets/-/blob/master/C/basic-fscanf/main.c

```
4   63,4,6,8,54\n
```

input.csv

Open in GitLab [19]

```
1   5,6,4,7\n
2   7,3,64,7,3,2\n
3   6,23\n
4   63,4,36,8,54\n
```

output.csv

Open in GitLab [20]

You may assume that your input file ends with a new line.
One can open a file using `fopen`, which will return a value of type `FILE*`. This value represents the file and has to be passed on to input and output functions. You can read more about `fopen` by opening the manual using the command `man 3 fopen`. Use the library function `fscanf` to read from a file, and `fprintf` to write to a file. Remember to close all open files with `fclose`.

Solution

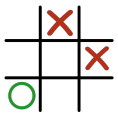
```c
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {

    if (argc < 2) {
        printf("Please␣specify␣an␣integer.\n");
        return 1;
    }

    int rowToSquare = atoi(argv[1]);

    FILE *in = fopen("input.csv", "r");      // opens input file
    FILE *out = fopen("output.csv", "w");    // opens output file

    char currChar;
    int counter = 0;
    int accu = 0;

    while (fscanf(in, "%c", &currChar) == 1) {
        if (currChar >= 48 && currChar <= 57) {

            accu = accu * 10 + (currChar - 48);

            if (accu < 0) {
                printf("Overflow␣occured,␣number␣too␣large");
                return 1;
            }

        } else if (currChar == ',' || currChar == '\n') {

            long result =
                (counter == rowToSquare ? (long)accu * (long)accu : accu);
```

[19] https://git.prog2.de/staff/exercise_sheets/-/blob/master/C/c-einkauf/code_0.js
[20] https://git.prog2.de/staff/exercise_sheets/-/blob/master/C/c-einkauf/code_1.js

```
34              fprintf(out, "%ld%c", result, currChar);
35
36              accu = 0;
37              counter = (currChar == '\n' ? -1 : counter);
38              counter++;
39
40          } else {
41
42              printf("Encountered␣unexpected␣character");
43              return 1;
44
45          }
46      }
47
48      fclose(in);
49      fclose(out);
50      return 0;
51  }
```

Open in Browser Open in GitLab [21]

Exercise 4.12: gamereader



Write a C program that reads a game configuration from an input file and stores the information in `PlayerData` and `GameConfig` structs.

```
 1 typedef struct {
 2     char name[7]; // Always exactly 6 characters long
 3     int age;
 4     double experience;
 5     double health;
 6 } PlayerData;
 7
 8 typedef struct {
 9     int level;
10     int enemies;
11     PlayerData *player1;
12     PlayerData *player2;
13 } GameConfig;
```

Open in Browser Open in GitLab [22]

The input file has the following format: (The <...> are placeholders for the actual values):

```
level: <level>
enemies: <enemies>
- <name player 1>: <age player 1> <experience player 1> <health player 1>
- <name player 2>: <age player 1> <experience player 1> <health player 1>
```

Open in GitLab [23]

Your task now is to implement the `readGameConfig` function

```
1 GameConfig* readGameConfig(char* filename) {
2     // TODO
3 }
```

Open in Browser Open in GitLab [24]

[21] https://git.prog2.de/staff/exercise_sheets/-/blob/master/C/c-einkauf/code_2.c
[22] https://git.prog2.de/staff/exercise_sheets/-/blob/master/C/c-fscanf/code_0.c
[23] https://git.prog2.de/staff/exercise_sheets/-/blob/master/C/c-fscanf/code_1.java
[24] https://git.prog2.de/staff/exercise_sheets/-/blob/master/C/c-fscanf/code_2.c

Solution

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

typedef struct {
    char name[7]; // Always exactly 6 characters long
    int age;
    double experience;
    double health;
} PlayerData;

typedef struct {
    int level;
    int enemies;
    PlayerData *player1;
    PlayerData *player2;
} GameConfig;

GameConfig *readGameConfig(char *filename) {
    // Try to open the file
    FILE *file = fopen(filename, "r");
    if (file == NULL) {
        return NULL;
    }

    // Allocate memory for the config and the players
    GameConfig *config = malloc(sizeof(GameConfig));
    config->player1 = malloc(sizeof(PlayerData));
    config->player2 = malloc(sizeof(PlayerData));

    // Read game data
    fscanf(file, "level: %d\n", &config->level);
    fscanf(file, "enemies: %d\n", &config->enemies);

    // Read player data
    fscanf(file, "- %6s : %d %lf %lf\n", config->player1->name,
           &config->player1->age, &config->player1->experience,
           &config->player1->health);
    fscanf(file, "- %6s : %d %lf %lf\n", config->player2->name,
           &config->player2->age, &config->player2->experience,
           &config->player2->health);

    // Close the file handle and return the config
    fclose(file);
    return config;
}

int main() {
    GameConfig *config = readGameConfig("c-fscanf-test");
    if (config == NULL) {
        printf("Error reading file\n");
        return 1;
    }

    printf("Level: %d\n", config->level);
    printf("Enemies: %d\n", config->enemies);

    printf("Player 1: %s\n", config->player1->name);
    printf("Age: %d\n", config->player1->age);
    printf("Experience: %lf\n", config->player1->experience);
```

```
60      printf("Health: %lf\n", config->player1->health);
61
62      printf("Player 2: %s\n", config->player2->name);
63      printf("Age: %d\n", config->player2->age);
64      printf("Experience: %lf\n", config->player2->experience);
65      printf("Health: %lf\n", config->player2->health);
66
67      free(config->player1);
68      free(config->player2);
69      free(config);
70      return 0;
71  }
```

Open in Browser Open in GitLab [25]

[25] https://git.prog2.de/staff/exercise_sheets/-/blob/master/C/c-fscanf/code_3.c