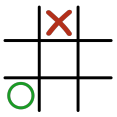
Programming 2 (SS 2023)
Saarland University
Faculty MI
Compiler Design Lab

Sample Solution 5
Advanced 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 5.1:



For the following programs, find out if their execution is uniquely defined by the C11 standard. You may assume that enough memory is available.

1. Task a:

```c
#include <stdio.h>
int main() {
    int a[15];
    for (int i = 0; i < 15; i++) {
        a[i] = 14 - i;
    }
    int x = 3[a];
    printf("Value␣of␣x:␣%i\n", x);
    return 0;
}
```

Open in Browser Open in GitLab [1]

2. Task b:

```c
#include <stdio.h>
int main() {
    int x = 0;
    int y = 42 % x;
    return 0;
}
```

Open in Browser Open in GitLab [2]

3. Task c:

```c
#include <stdio.h>
int main() {
    printf("%f\n", 0);
    return 0;
}
```

Open in Browser Open in GitLab [3]

4. Task d:

[1] https://git.prog2.de/staff/exercise_sheets/-/blob/master/C/undefined_behaviour/code_0.c
[2] https://git.prog2.de/staff/exercise_sheets/-/blob/master/C/undefined_behaviour/code_1.c
[3] https://git.prog2.de/staff/exercise_sheets/-/blob/master/C/undefined_behaviour/code_2.c

```c
#include <stdio.h>
int sum(int x, int y) {
    return x + y;
}

int main() {
    int i = 5;
    int res = sum(i*i, ++i);
    printf("result␣of␣sum:␣%i", res);
    return 0;
}
```

Open in Browser Open in GitLab [4]

5. Task e:

```c
#include <stdio.h>
int main() {
    int a[10];
    for (int i = 0; i < 11; i++) {
        a[i] = a + i;
    }
    return 0;
}
```

Open in Browser Open in GitLab [5]

6. Task f:

```c
#include <stdio.h>
int main() {
    char* p = NULL;
    char c = *p;
    return 0;
}
```

Open in Browser Open in GitLab [6]

7. Task g:

```c
#include <stdio.h>
int foo() {
    int a = 1234;
    double b = 12.34;
}

int main() {
    int x = foo();
    return 0;
}
```

Open in Browser Open in GitLab [7]

8. Task h:

[4] https://git.prog2.de/staff/exercise_sheets/-/blob/master/C/undefined_behaviour/code_3.c
[5] https://git.prog2.de/staff/exercise_sheets/-/blob/master/C/undefined_behaviour/code_4.c
[6] https://git.prog2.de/staff/exercise_sheets/-/blob/master/C/undefined_behaviour/code_5.c
[7] https://git.prog2.de/staff/exercise_sheets/-/blob/master/C/undefined_behaviour/code_6.c

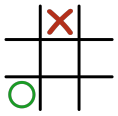
```c
#include <stdio.h>
int main() {
    int x;
    x ^= x;
    return x;
}
```

Open in Browser Open in GitLab [8]

Solution

1. The program contains no undefined behaviour. The output of the program is `Value of x: 11`. The expression `3[a]` is only syntactical sugar for `*(3+a)` and therefore valid.
2. The behaviour of division / modulo by zero is undefined.
3. The behaviour is undefined. `%f` expects a value of type `double`, but 0 is of type `int`.
4. The C standard does not mandate in which order the arguments to a function are evaluated. The resulting situation is called "unsequenced read/write" and the behaviour is undefined in this case, since it is uncertain whether `i` is incremented or read first.
5. The behaviour is undefined. The calculation `a + i` is allowed, also for the case `i = 10`, since pointers to the element behind the array may be used. But the access `a[i]` in the case `i = 10` is invalid, since you may only access the elements of the array. Additionally, the access needs a conversion from a pointer to `int`, which is implementation defined and may therefore produce undefined behaviour.
6. Dereferencing a null pointer is undefined behaviour.
7. Undefined behaviour occurs since `main` calls the function `foo` and accesses the return value, but `foo` lacks a `return` statement.
8. Reading uninitialized memory leads to undefined behaviour. Not only the value itself is undefined, otherwise one could think that the program would always return 0, but the entire program behaviour is undefined. This means the program could also crash with an error message or continue running with $x \neq 0$.

Exercise 5.2:



Konrad Klug learned a lot about `malloc` and `free` today, but he has not drunk enough coffee before he tried to put his new knowledge into practice. Find the errors involving `malloc` and `free` in his code. Shortly explain each error.

1. Array reverse:

```c
// Reverses an array, returns pointer to reversed array of equal size
// arrayptr: Base address of the array
// arraysize: Number of elements in the array
int *arrayrev(int *arrayptr, int arraysize) {
    int *ret = malloc(arraysize);
    for(int i = 0; i < arraysize; i++){
        ret[i] = arrayptr[arraysize-i-1];
    }

    return ret;
}
```

[8] https://git.prog2.de/staff/exercise_sheets/-/blob/master/C/undefined_behaviour/code_7.c

Open in Browser Open in GitLab [9]

2. Print 5:

```
1 // prints the number 5 twice
2 void broken() {
3     int a = 5;
4     int *b = malloc(sizeof(int));
5     *b = 5;
6     printf("two numbers: %d, %d\n", a, *b);
7
8     free(&a);
9     free(&b);
10 }
```

Open in Browser Open in GitLab [10]

3. Squares:

```
1 // prints squares of the numbers in [1, upto].
2 void quadratics(int upto) {
3     for(int i = 1; i <= upto; i++){
4         int *a = malloc(sizeof(int));
5         *a = i*i;
6         printf("i*i =  %d\n", *a);
7     }
8
9     free(a);
10 }
```

Open in Browser Open in GitLab [11]

4. Why does the code in i) work, but not the code in ii)?

   (a) Code 1:

```
1 int main() {
2     int array[] = {10, 20, 30, 40};
3     int elements = sizeof(array) / sizeof(int);
4         for(int i = 0; i < elements; i++){
5             printf("%d\n", array[i]);
6         }
7     }
8 }
```

Open in Browser Open in GitLab [12]

   (b) Code 2:

```
1 void printArray(int array[]) {
2     int elements = sizeof(array) / sizeof(int);
3     for(int i = 0; i < elements; i++){
4         printf("%d\n", array[i]);
5     }
6 }
7
```

[9] https://git.prog2.de/staff/exercise_sheets/-/blob/master/C/malloc_errors/code_0.c
[10] https://git.prog2.de/staff/exercise_sheets/-/blob/master/C/malloc_errors/code_1.c
[11] https://git.prog2.de/staff/exercise_sheets/-/blob/master/C/malloc_errors/code_2.c
[12] https://git.prog2.de/staff/exercise_sheets/-/blob/master/C/malloc_errors/code_3.c

```
 8 int main() {
 9     int array[] = {10, 20, 30, 40};
10     printArray(array);
11 }
```

Open in Browser Open in GitLab [13]

Solution

1. `malloc` expects the size of the memory to be allocated in bytes, not in the number of elements. The following would be correct:

```
1       int *ret = malloc(arraysize * sizeof(int));
```

2. `free` expects a pointer to a container which was allocated by `malloc`. The container of the local variable `a` is automatically deallocated when the surrounding block (here: the function) is left and must therefore not be freed with `free`. Additionally, `free(&b)` is an error, since `b` is already a pointer to memory allocated by `malloc`. The following would be correct: `free(b)`

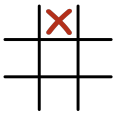
3. It is attempted to refer to the variable `a` when using `free`. This variable is however only defined in the block of the for-loop. This leads to a compilation error. Although the container is automatically freed after every loop iteration, but this is not the case for the container the address in `a` points to. This leads to a memory leak which can be fixed by moving `free(a)` to the end of the `for`-loop behind `printf`.

4. When executing ii) only the first few elements (depending on the processor architecture and operating system more or less many) are printed. The problem is that `int array[]` in the parameter list of the function `printArray` is only syntactical sugar for `int* array`. Thus, `sizeof(array)` evaluates only to the size of an `int*` value. For the other code, `sizeof(array)` evaluates to the amount of bytes in the array.

   When compiling the erroneous with gcc, the compiler prints a helpful warning:

```
1       warning: 'sizeof' on array function parameter
2       'array' will return size of 'int␣*'.
```

Exercise 5.3: median



Write a program that reads a list of numbers from a file and calculates the median of them.
In order to calculate the median, you should create a linked list of the numbers. For this, use the following struct:

```
typedef struct el_t el_t;
struct el_t {
  et_t *next;
  int   data;
};
```

Open in GitLab [14]

After inserting all numbers sorted into the list, you can easily find the median by taking the middle element of the list.
First, write a function `insert` which inserts a number into an already sorted linked list such that it stays sorted. The function takes a pointer to the first element of the list as argument and should return a pointer to the (maybe new) first element of the list. Use the following command to create a container for the new list element:

```
el_t *new_el = malloc(/* Size of the list element */);
```

[13] https://git.prog2.de/staff/exercise_sheets/-/blob/master/C/malloc_errors/code_4.c
[14] https://git.prog2.de/staff/exercise_sheets/-/blob/master/C/c-median/code_0.txt

Open in GitLab [15]

You just have to set the size of the list element, which you can compute with `sizeof`. Try to find out which header file you have to include next to `stdio.h` by using the `man` command.
Next, write a function `get` which gets a position and a linked list and returns a pointer to the element at that position. At the end, you should print the median.

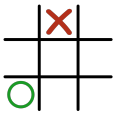
Solution

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

typedef struct el_t el_t;
struct el_t {
  el_t *next;
  int   data;
};

el_t *insert(el_t *head, int val)
{
  el_t *prev = NULL, *cur = head;
  while (cur != NULL && cur->data < val) {
    prev = cur;
    cur = cur->next;
  }

  el_t *new = malloc(sizeof(*new));
  new->data = val;
  new->next = cur;
  if (prev != NULL) {
    prev->next = new;
  }
  if (cur == head)
    return new;
  return head;
}

el_t *get(el_t* head, int n)
{
  while (n-- > 0 && head != NULL) {
    head = head->next;
  }
  return head;
}

int main(void)
{
  FILE *in = fopen("a.txt", "r");
  el_t *head = NULL;
  int size = 0;
  int val;
  while (1 == fscanf(in, "%d", &val)) {
    head = insert(head, val);
    ++size;
  }
  fclose(in);
  float median = get(head, (size -1 ) / 2)->data;
  if (size % 2 == 0) { // if even number of elt's, median is mean of middle numbers
```

[15] https://git.prog2.de/staff/exercise_sheets/-/blob/master/C/c-median/code_1.txt

```
    median += get(head, size / 2)->data;
    median /= 2;
  }
  printf("%.2f\n", median);
  return 0;
}
```

Exercise 5.4:



1. Write a function which takes an integer parameter and returns a new C string "False" if the argument is 0. For every other value, the function should return "True", also as a new string. The function shall store the return value in a local variable and return it afterwards. Allocate the memory for the string using `malloc()`!

2. Write a main function which receives a number as a command line parameter and uses the function you implemented in a) to print "False" if a 0 was passed and "True" otherwise. Make sure to manage the memory correctly.

   Hint: The arguments of the main function are always passed as `char` arrays. You can use the standard library function `atoi` to transform the string representation of a decimal number to its numeric value. For example: `int x = atoi("1234"); // x = 1234`

3. On the next day, Rainer Wahnsinn and Konrad Klug meet each other. Konrad Klug is amazed by the abilities of Rainer. However, Konrad would like to return "Richtig" and "Falsch" instead of "True" and "False". Unfortunately, he caught a virus so that he cannot change the functions he already wrote, except for the main function.

   Write a function `char* convert_to_german(char*)` which transforms a given string into the German equivalent ("True" to "Richtig" and "False" to "Falsch"). For the case where neither "True" nor "False" are passed, the function shall return the C string "Undefiniert".

   Attention: Also make sure to correctly allocate and deallocate the used memory here! It could make sense to use the standard library function `strcmp`.

4. Rainer Wahnsinn has caught a rare encrypting virus which encrypted his `string.h` file! Help him by providing your own implementation of the `strcpy` function.

   Tip: You can display the specification of `strcpy` with `man strcpy`.

   Test your implementation by using it in your implementation of a).

Solution

1. Solution for subtask a):

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

char* bool_eval(int x) {
    char* ret = NULL;
    if (x == 0) {
        ret = malloc(6 * sizeof(char));
        strcpy(ret, "False");
    }
    else {
        ret = malloc(5 * sizeof(char));
        strcpy(ret, "True");
    }
    return ret;
}
```

2. Solution for subtask b):

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

int main(int argc, char* argv[]) {
    if (argc != 2) {
        fprintf(stderr, "Invalid number of arguments!\n");
    }
    else {
        for (char* c = argv[1]; *c != '\0'; c++) {
            if (*c < '0' || *c > '9') {
                fprintf(stderr, "Invalid argument!\n");
                return -1;
            }
        }
        char* test = bool_eval(atoi(argv[1]));
        printf("%s\n", test);
        free(test);
    }
}
```

3. Solution for subtask c):

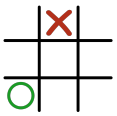
```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

char* bool_eval(int x) {
    char* ret = NULL;
    if (x == 0) {
        ret = malloc(6 * sizeof(char));
        strcpy(ret, "False");
    }
    else {
        ret = malloc(5 * sizeof(char));
        strcpy(ret, "True");
    }
    return ret;
}

char* convert_to_german(char* string) {
    if (strcmp(string, "True") == 0) {
        string = realloc(string, 8 * sizeof(char));
        strcpy(string, "Richtig");
        return string;
    }
    if (strcmp(string, "False") == 0) {
        string = realloc(string, 7 * sizeof(char));
        strcpy(string, "Falsch");
        return string;
    }
    string = realloc(string, 12 * sizeof(char));
    strcpy(string, "Undefiniert");
    return string;
}

int main(int argc, char* argv[]) {
    if (argc != 2) {
        fprintf(stderr, "Invalid number of arguments!\n");
    }
    else {
        for (char* c = argv[1]; *c != '\0'; c++) {
```

```
40              if (*c < '0' || *c > '9') {
41                  fprintf(stderr, "Invalid argument!\n");
42                  return -1;
43              }
44          }
45          char* test = bool_eval(atoi(argv[1]););
46          test = convert_to_german(test);
47          printf("%s\n", test);
48          free(test);
49      }
50 }
```

4. Solution for subtask d):

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3
 4 char* strcpy(char* s1, const char* s2) {
 5     char* s = s1;
 6     while ((*s++ = *s2++) != 0);
 7     return s1;
 8 }
 9
10 char* bool_eval(int x) {
11     char* ret = NULL;
12     if (x == 0) {
13         ret = malloc(6 * sizeof(char));
14         strcpy(ret, "False");
15     }
16     else {
17         ret = malloc(5 * sizeof(char));
18         strcpy(ret, "True");
19     }
20     return ret;
21 }
22
23 int main(int argc, char* argv[]) {
24     if (argc != 2) {
25         fprintf(stderr, "Invalid number of arguments!\n");
26     }
27     else {
28         for (char* c = argv[1]; *c != '\0'; c++) {
29             if (*c < '0' || *c > '9') {
30                 fprintf(stderr, "Invalid argument!\n");
31                 return -1;
32             }
33         }
34         char* test = bool_eval(atoi(argv[1]););
35         printf("%s\n", test);
36         free(test);
37     }
38 }
```

Exercise 5.5:



We consider the API for polynomials defined in the `poly.h` file:

```
1 #ifndef POLY_H
```

```c
#define POLY_H

struct poly_t;
typedef struct poly_t poly_t;

poly_t *poly_alloc(unsigned degree);
void poly_free(poly_t *poly);

void poly_set_coeff(poly_t *poly, unsigned i, int val);
int poly_eval(poly_t const *p, int x);

int poly_degree(poly_t *poly);

#endif
```

Open in Browser Open in GitLab [16]

Write the file `poly.c`, where you implement the functions declared in the header file:

1. Implement a function `poly_alloc` that allocates the space for a polynomial.
2. Implement the function `poly_free` that frees a polynomial that was constructed.
3. Implement the function `poly_set_coeff` that sets one coefficient of a polynomial.
4. Implement the function `poly_eval` that evaluates a polynomial using the Horner schema.
5. Implement the function `poly_degree` that returns the degree of a polynomial.

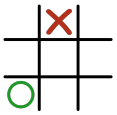
Solution

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

#include "poly.h"

struct poly_t {
    unsigned degree;
    int *coeffs;
};

poly_t *poly_alloc(unsigned degree) {
    poly_t *poly = malloc(sizeof(*poly));
    poly->degree = degree;
    poly->coeffs = malloc((degree + 1) * sizeof(poly->coeffs[0]));
    return poly;
}

void poly_free(poly_t *poly) {
    free(poly->coeffs);
    free(poly);
}

void poly_set_coeff(poly_t *poly, unsigned i, int val) {
    assert(i <= poly->degree);
    poly->coeffs[i] = val;
}

int poly_eval(poly_t const *poly, int x) {
```

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

```
29     int res = poly->coeffs[0];
30     for (unsigned i = 1; i <= poly->degree; i++)
31         res = res * x + poly->coeffs[i];
32     return res;
33 }
34
35 int poly_degree(poly_t *poly) {
36     return poly->degree;
37 }
```

Exercise 5.6:



Farmer John wants to implement a new system for his milking machine by milking his cows in order of registrations: If a cow wants to be milked, it registers itself on the milking machine using its ID number. If the milking machine is not occupied, and no other cow has registered before the cow, the cow is the next to be milked by the milking machine.
Konrad Klug thinks that this can be implemented by a FIFO (first-in, first-out) queue with the operations `pushBack()` and `popFront()`. A linked list is viable as the underlying data structure. In a linked list, an element also contains a pointer to the next element in addition to the element value.
To this end, Konrad built a small code skeleton and wants you to implement the functions `createQueueElem()`, `pushBack()`, and `popFront()`. If you implemented everything correctly, the numbers 1, 2, and 3 should be printed in this order.

```
 1 #include <assert.h>
 2 #include <stdio.h>
 3 #include <stdlib.h>
 4
 5 typedef struct Queue Queue;
 6 typedef struct QueueElem QueueElem;
 7
 8 struct QueueElem {
 9     int value;
10     QueueElem* successor;
11 };
12 struct Queue {
13     QueueElem* front;
14     QueueElem* back;
15 };
16
17 QueueElem* createQueueElem(int value) {
18     // TODO
19 }
20
21 void pushBack(Queue* queue, int value) {
22     // TODO
23 }
24
25 int popFront(Queue* queue) {
26     // TODO
27 }
28
29 int main() {
30     Queue myQueue = {NULL, NULL};
31     pushBack(&myQueue, 1);
32     pushBack(&myQueue, 2);
33     pushBack(&myQueue, 3);
34
35     printf("%d\n", popFront(&myQueue));
```

```
36      printf("%d\n", popFront(&myQueue));
37      printf("%d\n", popFront(&myQueue));
38
39      return 0;
40  }
```

Open in Browser Open in GitLab [17]

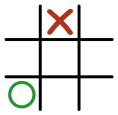
Solution

```
 1  #include <assert.h>
 2  #include <stdio.h>
 3  #include <stdlib.h>
 4
 5  typedef struct Queue Queue;
 6  typedef struct QueueElem QueueElem;
 7
 8  struct QueueElem {
 9      int value;
10      QueueElem *successor;
11  };
12
13  struct Queue {
14      QueueElem *front;
15      QueueElem *back;
16  };
17
18  QueueElem *createQueueElem(int value) {
19      QueueElem *element = (QueueElem *) malloc(sizeof(QueueElem));
20      element->value = value;
21      return element;
22  }
23
24  void pushBack(Queue *queue, int value) {
25      QueueElem *element = createQueueElem(value);
26      if (queue->front == NULL) {
27          queue->front = element;
28      } else {
29          queue->back->successor = element;
30      }
31      element->successor = NULL;
32      queue->back = element;
33  }
34
35  int popFront(Queue *queue) {
36      assert(queue->front != NULL);
37      int returnValue = queue->front->value;
38      QueueElem *firstElem = queue->front;
39      if (firstElem->successor == NULL) {
40          queue->back = NULL;
41      }
42      queue->front = firstElem->successor;
43      free(firstElem);
44      return returnValue;
45  }
46
47  int main() {
48      Queue myQueue = {NULL, NULL};
```

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

```
49      pushBack(&myQueue, 1);
50      pushBack(&myQueue, 2);
51      pushBack(&myQueue, 3);
52
53      printf("%d\n", popFront(&myQueue));
54      printf("%d\n", popFront(&myQueue));
55      printf("%d\n", popFront(&myQueue));
56
57      return 0;
58  }
```

Exercise 5.7:



Farmer John would like to automate the storage process of his hay bundles. Due to the maximum height of his storehouse, the bundles are stored as stacks of a fixed size. New bundles can be added to the top if space is left, or removed from the top if a stack is not empty, according to the LIFO (last-in, first-out) principle. For testing purposes, Konrad Klug has started working on an implementation where the hay bundles are represented as simple integers.

1. Complete the skeleton given to you by Konrad Klug. If you implemented everything correctly, the numbers 1, 2, and 3 should be printed in reverse order.
2. Thanks to a revolutionary compression method, individual stacks no longer need to have a fixed size. Instead, the capacity of a stack now doubles whenever it is completely full. Change your code to reflect this.
3. *Bonus:* After a successful testing phase, Farmer John would like to extend the system to other inventory items like seeds. He also wants to represent each item in an individual `struct` and have a stack per item category. Konrad notices that to support these requirements, the existing implementation would need to be duplicated for each type, but only the name of the structure and the stored type would change, which he thinks is a lot of unnecessary work. Help him by researching ways to make data structures type-generic in C.

```
 1  #include <assert.h>
 2  #include <stdio.h>
 3  #include <stdlib.h>
 4
 5  typedef int HayBundle;
 6
 7  typedef struct {
 8      size_t size;
 9      size_t capacity;
10      HayBundle* data;
11  } HayBundleStack;
12
13  HayBundleStack HayBundleStack_construct(size_t initial_capacity) {
14      // TODO: create new stack instance and initialize data members
15  }
16
17  void HayBundleStack_destroy(HayBundleStack* stack) {
18      // TODO: free memory allocated by the stack
19  }
20
21  void HayBundleStack_push(HayBundleStack* stack, HayBundle value) {
22      // TODO: push new value onto the stack if space left
23      // use an assertion to verify the stack is not full
24  }
25
```

```c
HayBundle HayBundleStack_pop(HayBundleStack* stack) {
    // TODO: if possible pop the top-most value and return it
    // use an assertion to verify the stack is not empty
}

int main() {
    HayBundleStack stack = HayBundleStack_construct(4);

    HayBundleStack_push(&stack, 1);
    HayBundleStack_push(&stack, 2);
    HayBundleStack_push(&stack, 3);

    printf("%d\n", HayBundleStack_pop(&stack));
    printf("%d\n", HayBundleStack_pop(&stack));
    printf("%d\n", HayBundleStack_pop(&stack));

    HayBundleStack_destroy(&stack);

    return 0;
}
```

Open in Browser Open in GitLab [18]

Solution

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

typedef int HayBundle;

typedef struct {
    size_t size;
    size_t capacity;
    HayBundle* data;
} HayBundleStack;

HayBundleStack HayBundleStack_construct(size_t initial_capacity) {
    HayBundleStack stack = {.size = 0, .capacity = initial_capacity,
            .data = malloc(initial_capacity * sizeof(HayBundle))};
    return stack;
}

void HayBundleStack_destroy(HayBundleStack* stack) {
    free(stack->data);
}

void HayBundleStack_push(HayBundleStack* stack, HayBundle value) {
    assert(stack->size < stack->capacity);

    stack->data[stack->size++] = value;
}

HayBundle HayBundleStack_pop(HayBundleStack* stack) {
    assert(stack->size > 0);

    return stack->data[--stack->size];
}
```

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

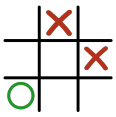
```
34
35 int main() {
36     HayBundleStack stack = HayBundleStack_construct(4);
37
38     HayBundleStack_push(&stack, 1);
39     HayBundleStack_push(&stack, 2);
40     HayBundleStack_push(&stack, 3);
41
42     printf("%d\n", HayBundleStack_pop(&stack));
43     printf("%d\n", HayBundleStack_pop(&stack));
44     printf("%d\n", HayBundleStack_pop(&stack));
45
46     HayBundleStack_destroy(&stack);
47
48     return 0;
49 }
```

2. Instead of returning once the size equals the capacity, we now reallocate the array:

```
1 void HayBundleStack_push(HayBundleStack* stack, HayBundle value) {
2     if (stack->size >= stack->capacity) {
3         stack->data = realloc(stack->data,
4                 (stack->capacity <<= 1) * sizeof(HayBundle));
5     }
6
7     stack->data[stack->size++] = value;
8 }
```

3. C offers two main ways to achieve generics. The first option is to use void pointers to abstract from the underlying data type, and store/retrieve individual elements byte-wise. However, this requires to keep track of the size of the stored elements, since void pointers have no information about the size of the objects they point to, and is inherently not type-safe because of the nature of type casts in C. The other option is to use macros to "instantiate" a new stack implementation for each required type, which improves type-safety because it leverages the existing type system.

Exercise 5.8: ChatGPT



Konrad Klug has written some C code and showed it to his friend Dieter Schlau. Dieter is a total fan of MIPS, so he would like the code that Konrad wrote to be in MIPS assembly rather than in C. Dieter doesn't know how to do this, so you need to help him. Use ChatGPT to help understand the code and to translate it to MIPS.

```
1 #include <stdio.h>
2
3 static int count_DigitOne(int n) {
4         int m = 0, k = 0, x = 0, base = 1;
5         while (n > 0) {
6             k = n % 10;
7             n = n / 10;
8
9             if (k > 1) { x += (n+1)*base; }
10            else if (k < 1) { x += n*base; }
11            else { x += n*base+m+1; }
12
13            m += k*base;
14            base *= 10;
15        }
16        return x;
17    }
```

```c

int main(void)
{
    int n = 42;
    int result = count_DigitOne(n);
    printf("Result: %d\n", result);
    return 0;
}
```

Open in Browser Open in GitLab [19]

Solution

The C program counts the total number of 1 digits appearing in all positive integers up to a given positive integer n and then outputs the result.

```
.data

str:   .asciiz "Result: "

.text
.globl main

# Function to calculate the sum of '1' digits present
# in all numbers up to and including the number in $a0
# Arguments:
# $a0 - Argument n
# Return Values:
# $v0 - The sum of '1' digits

countDigitOne:
        # a0 - n
        li $t0 0   # m
        li $t1 0   # k (remainder)
        li $t2 1   # base (current base)
        li $v0 0   # x (result)

        li $t3 10 # Constant for division
        # $t4 - temporary

countDigitOne_loop:
        blez $a0 countDigitOne_end   # If $a0(n) is not greater than 0,
                                          # jump to end of function
        divu $a0 $t3       # Divide $a0 by 10($t3)
        mflo $a0       # Move quotient to $a0
        mfhi $t1       # Move remainder to $t1

        bgtu $t1 1 countDigitOne_klarger1 # Go to countDigitOne_klarger1,
                                               # if k($t1) is greater than 1
        bltu $t1 1 countDigitOne_ksmaller1   # Go to countDigitOne_ksmaller1,
                                               # if k($t1) is less than 1

        mulou $t4 $a0 $t2 # Store result of n($a0) * base($t2) in $t4
        addu $v0 $v0 $t4   # Add result stored in $t4 to x($v0)
        addu $v0 $v0 $t0   # Add m to x($v0)
        addiu $v0 $v0 1    # Add 1 to x($v0)

        b countDigitOne_loopend # Go to the end of the loop

```

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

```
44 countDigitOne_klarger1:
45         addiu $t4 $a0 1    # Store the result of n($a0) + 1 in $t4
46         mulou $t4 $t4 $t2 # Store the result of (n($a0) + 1) * base($t2) in $t4
47         addu $v0 $v0 $t4   # Add the final result to x($v0)
48
49         b countDigitOne_loopend # Go to the end of the loop
50
51 countDigitOne_ksmaller1:
52         mulou $t4 $a0 $t2 # Store the result of n($a0) * base($t2) in $t4
53         addu $v0 $v0 $t4   # Add the previous result to x($v0)
54
55 countDigitOne_loopend:
56         mulou $t4 $t1 $t2 # Store the result of k($t1) * base($t2) in $t4
57         addu $t0 $t0 $t4   # Add the previous result to m($t0)
58         mulou $t2 $t2 10   # Multiply base($t2) by 10
59
60         b countDigitOne_loop   # Goto the beginning of the loop
61
62 countDigitOne_end:
63         jr $ra     # Return to caller
64
65
66 main:
67         # Load 42 into argument register 1
68         li $a0 42
69
70         jal countDigitOne # Call counter function
71         move $t0 $v0     # Save result of function call in $t0 as we need $v0
72
73         # Print string "Result: "
74         la $a0 str   # Load address of string to print into $a0
75         li $v0 4   # Load 4 into $v0 -> Syscall for printing
76                         # a null terminated string
77         syscall    # Execute syscall
78
79         # Print result number
80         move $a0 $t0   # Move function result from $t0 into $a0 to use in syscall
81         li $v0 1   # Load 1 into $v0 -> Syscall for printing integer
82         syscall    # Execute syscall
83
84         # Terminate program
85         li $v0 10 # Load 10 int $v0 -> Syscall for terminating execution
86         syscall    # Execute syscall
```