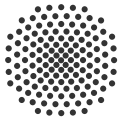

University of Stuttgart

Programming Paradigms Prof. Dr. Michael Pradel

University of Stuttgart, Summer 2023

Exercise 5: Composite Types

(Deadline for uploading solutions: July 6, 2023, 11:59pm Stuttgart time)

The materials provided for this homework are:

- a pdf file with the text of the homework (this);
- a zip file with **the folder structure and the templates that must be used for the submission**.

The folder structure is shown in Figure 1.

```
exercise5.zip
├─ task_1
│  └─ task1.csv (Insert your solution in this file)
├─ task_2
│  ├─ structBag.csv (Insert your solution in this file)
│  └─ structSuitcase.csv (Insert your solution in this file)
├─ task_3
│  ├─ puzzle1.c (Insert your solution in this file)
│  ├─ puzzle2.c (Insert your solution in this file)
│  └─ puzzle3.c (Insert your solution in this file)
└─ task_4
   ├─ pl_c
   │  ├─ LibraryCatalog.c (Insert your solution in this file)
   │  └─ LibraryCatalogTest.c
   ├─ pl_java
   │  ├─ lib/*.jar
   │  └─ src
   │     ├─ LibraryCatalog.java (Insert your solution in this file)
   │     ├─ LibraryCatalogTest.java
   │     └─ TestRunner.java
   └─ pl_python
      ├─ LibraryCatalog.py (Insert your solution in this file)
      └─ LibraryCatalogTest.py
```

Figure 1: Directory structure in the provided .zip file and in your uploaded solution file.

The submission must be compressed in a zip file using the given folder structure. The names of folders and files must not be changed or moved, otherwise the homework will not be evaluated.

General tips for making sure your submission is graded as you expect:

- Use only the zip format for the archive (**NOT** rar, 7z, gz, or anything else).
- Do not rename files or folders, simply open the files provided and put your solutions.

1 Task I (10% of total points of the exercise)

This task is to check your understanding of some basic concepts around composite types. For that, we give you code snippets written in C/C++. You have to **answer one question per snippet** by selecting **exactly one** correct answer out of multiple choices. To submit your answer, please fill in the file *exercise5/task_1/task1.csv* by adding either "A", "B", or "C" to the corresponding line.

Code Snippet 1

```c
struct foo {
  int n;
  char cs[8];
  _Bool b;
};
```

Which statement is true for Code Snippet 1?

A `struct foo` is a primitive type.

B `struct foo` is a composite type.

C `struct foo` is a function.

Code Snippet 2

```c
struct s1 {
    int n;
};
struct s2 {
    struct s1 s1;
};

struct s2 x;
x.s1.n = 3;
struct s2 y;
y.s1 = x.s1;
y.s1.n = 5;
printf("%d\n", x.s1.n);
```

Which statement is true for Code Snippet 2?

A The code prints 3 because `y.s1` is a copy of `x.s1`, and hence, the write to `y.s1.n` does not affect `x.s1.n`.

B The code prints 3 because `x.s1` and `y.s1` are references to the same memory object.

C The code prints 3 because the write to `x.s1.n` is reordered by the compiler so that it happens after the write to `y.s1.n`.

Code Snippet 3

```c
int arr[3][4] = {
    {1,2,3,4},
    {5,6,7,8},
    {9,10,11,12}
};

int sum = 0;
for (int j=0; j<3; j++) {
    for (int i=0; i<4; i++) {
        sum += arr[j][i];
    }
}
```

Which statement is **false** for Code Snippet 3?

A To improve efficiency, the code should change `arr[j][i]` to `arr[i][j]`.

B The code access the elements of the array in the same order as they are stored in memory.

C The code accesses a multi-dimensional array that is stored in row-major layout.

Code Snippet 4

```
1  struct n {
2    int val;
3    struct n* left;
4    struct n* right;
5  };
6
7  struct n n1;
8  struct n n2;
9  struct n n3;
10 n1.left = &n2;
11 n1.right = &n3;
12 n1.right->val = 42;
13 printf("%d\n", n3.val); // prints 42
```

Which statement is **false** for Code Snippet 4?

A Lines 7, 8, and 9 should call `malloc` to ensure that there is enough memory to store the structs.

B Line 10 uses `&` to store a reference to the `n2` into the `left` field..

C Line 12 uses `->` to dereference the pointer and then access its `val` field.

Code Snippet 5

```
1  double *ds = malloc(5 * sizeof(double));
2  for (int i=0; i<5; i++) {
3    ds[i] = 1.234;
4  }
5  free();
```

Which statement is true for Code Snippet 5?

A The code correctly frees all allocated memory.

B The code should remove the call to `free` because memory objects allocated with `malloc` are implicitly freed anyway.

C The code should pass the pointer `ds` to free to actually free its memory.

Evaluation Criteria: Your answers will be compared against the correct ones.

2 Task II (20% of total points of the exercise)

This task is about the **memory layout** of structs and **alignment** in C (or other "systems" languages). Given several definitions of structs in C and several sets of rules, you should compute the memory layout of each struct for each set of rules. That is, you should determine at **which offset each field starts** and the **overall size** of the struct in memory. Table 1 shows the sizes and natural alignment of primitive types.

Types	Size (byte)	Natural alignment (byte)
`char`	1	no alignment
`int`	4	4
`float`	4	8
`pointer`	8	8

Table 1: Size and natural alignment of primitive types. Natural alignment, where memory access of the underlying architecture would be fastest. An alignment requirement of 4 bytes means that the byte offset of that field must divide without remainder by 4. The first offset in each struct is 0.

You have to compute the **most compact** memory layout for each struct under the following rules:

- **Packed**: The natural alignment requirements (see above) do not have to be respected, i.e., each field can start at an arbitrary offset. The order of fields must not be changed.
- **Default**: The above alignment requirements must be respected for each field. The order of fields must not be changed.
- **Reordered**: The above alignment requirements must be respected for each field, but the order of fields can be changed compared with the declaration. (But not across structs, that would change semantics.)

The two struct definitions are:

Struct Bag

```
struct Bag {
  char brand[20];
  float *price;
  int sales;
};
```

Struct Suitcase

```
  struct Date {
      int month;
      int year;
  };

  struct Suitcase {
      char brand[15];
      int sales;
      struct Date productionDate;
      struct Size {
          float length;
          float width;
          float height;
      } size;
  };
```

For solving the task, it may be useful to draw layout figures with pen and paper, similar to the lecture. However, your final answers must be submitted in the *structBag.csv* and *structSuitcase.csv* files in the *exercise5/task_2/* directory. There is one row for each field of the struct and a final row for the overall size in bytes. For the fields, fill in the offset (i.e., the byte index at which the field begins), viewed from the start of the outermost

Struct Bag: Field / Rule: Packed	Byte offset
`brand`	0
`price`	20
`sales`	28
Overall size of the struct	32

Table 2: The layout of `struct Bag` under the packed rule.

struct, in the columns corresponding to each of the three rules. Table 2 shows the layout of struct Bag under the packed rule. The solution is already filled into the solution template in *exercise5/task_2/structBag.csv*.

Evaluation Criteria: Your solution will be compared against the correct byte offsets of each struct field and the correct overall struct size under each set of alignment requirements.

3 Task III (30% of total points of the exercise)

This task is about **pointer arithmetic and arrays** in C/C++. You are provided with three slightly incorrect programs and have to fix them, so they actually offer the behavior specified by the comments in the code. See *exercise5/task_3/puzzle*.c* for the three programs to fix. You are allowed to modify only those parts of the code between the "start modify" and "end modify" comments. Each provided program has one or more bugs, i.e., you will need to modify one or more lines of the given code.

The solution should be implemented in C17, i.e., the most recent standard for the C programming language. We recommend using the gcc compiler, which is compatible with C17. We will use gcc version 11.3.0 for grading.

Evaluation Criteria: Your code will be evaluated by running tests that check whether your implementation matches the behavior specified in the comments.

4 Task IV (40% of total points of the exercise)

This task is about **composite types** and the differences between **reference model** and **value model**. You will write a small program that models a library catalog, using three different languages: Java, C, and Python. For each language, we provide an incomplete program for you to complete. In particular, you should implement the *addNewVersion* function in all three languages. You can run the code by using the provided *main* function, and test it with the provided test cases. Figure 2 shows the overall architecture of the expected library catalog.

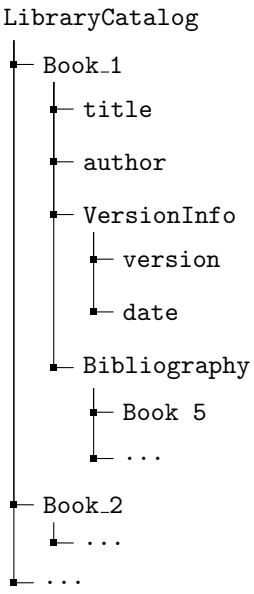

Figure 2: Overall architecture of the library catalog.

Here are several **requirements** for implementing the *addNewVersion* function:

- It should allow adding a new version of an existing book to the catalog by using existing information (title, author, and bibliography). After adding a new version, the book should have two entries in the catalog, one for the existing, old version, and another for the newly added version.
- There should be a message if the book you want to add a new version for is not in the catalog. See the code for the exact message.

Notes: Our test cases for this task will call the newly implemented *addNewVersion* function and also the given functions. That is, do not change the code in the provided functions. We provide several test cases in the corresponding *Test* file. As usual, adding further test cases is recommended.

The assignment should be implemented in Java (version >= 20), C17 (will be tested with gcc 11.3.0), and Python (version >= 3.8).

Evaluation Criteria: Your code will be evaluated by running on a set of test cases. For any of the languages, do not use any third-party dependencies, except for the libaries that are already included in the given C code, and for Java, the jar given in */task_4/pl_java/lib/* for running the tests.