Chapter 6

Exercise: Copy Constructor and Assignment Operators

6.1 Leaning Objectives

References to literature are provided in brackets:

- avoid global variables,
- Using classes,
- Get-ter and Set-ter
- deep copy versus shallow copy,
- Copy-Constructor
- Assignment Operator

see directory ExeCopyConstructor for solution code

6.2 Exercises at a glance

Exercise 6.1 Replace the global variables.

Exercise 6.2 Use classes instead of the structures.

Exercise 6.3 Enable that a file may contain an arbitrary number of commands for one utterance (and not only 6).

Exercise 6.4 Implement also a minimal standard interface for all of your classes, which allocate memory on the heap (e.g. for class `AtcoCmds`).

6.3 Detailed Exercise Descriptions

Exercise 6.1: If you are still using global variables, replace them by e.g. using a class. Your previous code might contain

```
// dynamic array to store the commands and the worconstd sequences
AtcoCommand* gp_atcoUtterances = nullptr;
// number of elements currently used in mp_atcoUtterances
int g_actoUtterancesCnt = 0;
// maximal number of elements currently usable in gp_atcoUtterances
int g_max_utterance_cnt;
```

Replace this code by classes, e.g.

```
class AtcoCmds
{
public:
  ...
  int GetUtteranceCnt() const {return m_actoUtterancesCnt;}
  string GetFilename(int i) const
  {
    CheckAgainstUttCnt(i);
    return mp_atcoUtterances[i].GetFilename();
  }
  ...

private:
  void CheckAgainstUttCnt(int i) const {
    if (i >= m_actoUtterancesCnt)
    {
      throw std::exception("bad array index");
    }
  }

  // dynamic array to store the commands and the worconstd sequences
  AtcoCommand* mp_atcoUtterances = nullptr;
  // number of elements currently used in mp_atcoUtterances
  int m_actoUtterancesCnt = 0;
  // maximal number of elements currently usable in gp_atcoUtterances
  int m_max_utterance_cnt;
};
```

Of course all class members should be private and also be usable via methods. Show that your class implementation works via suitable tests, i.e. it should be possible to input two files at the same time without resetting (the not existing) global variables. Avoid memory leaks via using suitable destructors.

Exercise 6.2: Replace all structs via classes, i.e. avoid attributes in the public area. All attributes should be in the private area and all be usable via Getter and Setter method and other public methods.

```
class AtcoCommand
{
public:
  AtcoCommand();
 /*...*/
private:
```

```
  FileName m_filename;
  string m_wordSeq;
  enum { MAX_CMDS_PER_UTTERANCE = 6 };
  // do not try to also make this number dynamic
  string m_cmds[MAX_CMDS_PER_UTTERANCE];
  int    m_cmdCnt;
};

class FileName
{
public:
/* ....*/
private:
  string mstr_filename;
  int m_year;
  int m_month;
  int m_day;
  int m_hour;
  int m_min;
  int m_sec;
  int m_millisec;
};
```

Exercise 6.3: Currently the `AtcoCommand` can only store six strings (at maximum).[1]

```
class AtcoCommand
{
...
private:
  enum { MAX_CMDS_PER_UTTERANCE = 6 };
  // do not try to also make this number dynamic
  string m_cmds[MAX_CMDS_PER_UTTERANCE];
  int    m_cmdCnt;
};
```

Improve this so that an arbitrary number is possible. Demonstrate this also by a test.

The following solution is only half of the necessary improvement.

```
class AtcoCommand
{
...
private:

  // pointer to an array of strings
  string* mp_cmds;
  // currently stored strings/commands
  int    m_cmdCnt;
  // maximal number of commands (before extension is necessary)
  int    m_maxCnt;
};
```

You also need to implement a copy constructor and an assignment operator for this class.

[1] Maybe you have chosen a different class name. Then improve your class so that also can handle more than six commands.

```
class AtcoCommand
{
public:
  AtcoCommand(const AtcoCommand& copy);
  AtcoCommand& operator=(const AtcoCommand& copy);
```

Exercise 6.4: If not already done, also implement copy constructor and assignment operator for the class `AtcoCmds` and provide suitable tests.[2]

If you have more classes implemented which allocate memory on the heap, also implement a minimal standard interface for them (at least you should do the implementation of two of them).

6.4 Evaluation criteria

For exercise 6.1 and exercise 6.2 no special evaluation criteria are defined.

Your code should be in the folder `Aufgabe06/Code`, i.e. your screen dumps are expected in SVN in folder `Aufgabe06/Images`.

For **Exercise 6.3**:
Write a test, which reads from a file with more than six commands for one single utterance (sequence of words). You can define arbitrary commands for your input file. It is not necessary that they fit to the words. The test should just show that the limit of six commands does not exist any more. Do not put too many lines into your test input file. You just should show for one example that the limit of six does not exist any more.

Make a screen dump and store in SVN. The screen dumps should show that you are able to read more than six commands for one utterance. `exercise6-3-MoreThanSix.jpg` etc. See excel file with evaluation criteria what is exactly expected in exercise6-3-MoreThanSix01.jpg, exercise6-3-MoreThanSix02.jpg, exercise6-3-MoreThanSix03.jpg and exercise6-3-MoreThanSix04.jpg.

For **Exercise 6.4**:
Implement at least one test for testing the assignment operator and the copy constructor for at least two of your classes, which allocates heap memory. The tests must show that the assignment operator and the copy constructor are correctly implemented.[3] Make screen dumps of your tests. Store your screen dumps in SVN in the files with names `exercise6-4-TestAssOp01.jpg`, `exercise6-4-TestAssOp02.jpg` etc. and in `exercise6-4-TestCopyConstr01.jpg`, `exercise6-4-TestCopyConstr02.jpg` etc.

[2] Maybe you have chosen a different class name. Then use that class for adding your assignment operator and copy constructor.

[3] If only the default assignment operator and default copy constructor are implemented the test should crash. That is sufficient, e.g. implement something similar as the `DoNothing` function in the lecture. It is not necessary that your tests crash or fail. They should just crash or fail, in the case I would delete your copy constructors and assignment operators.

Make also screen dumps of your assignment operators and copy constructors. Store your screen dumps in SVN in the files with names `exercise6-4-AssOp01.jpg`, `exercise6-4-AssOp02.jpg` etc. and in `exercise6-4-CopyConstr01.jpg`, `exercise6-4-CopyConstr02.jpg` etc.