Homework 2 (Debugging)

Giacomo Cirò

20875 - Software Engineering, Bocconi University

We consider `/scipoptsuite-9.1.1/` to be the working directory.

Part I

Q1. The following file in `.mps` format causes a `segmentation fault`:

```
NAME
ROWS
 k
```

Background: when using this file format, constraints for an LP can be created in the `ROWS` section by declaring the constraint type and the constraint name. In this corrupted file, only one token `k` is present which is considered to be the constraint type, without any name specified.

Q2. With assertions disabled, a crash happens at line `2858` in `scip/src/scip/misc.c` in the function `SCIPhashKeyValString()` because the pointer variable `str` is `NULL` and the expression `while( *str != '\0' )` de-references a `NULL` pointer.

With assertions enabled, `assert(name != NULL)` fails at line `2954` in `scip/src/scip/scip_prob.c` in the function `SCIPfindCons()`.

Q3. SCIP's `.mps` file parser works by reading and parsing one line at a time from the input file. Inside `main()`, a sequence of nested functions are called up to `readMps()`, which starts the parsing. Inside this function, two other important functions are called:

- `SCIPcreateProb()` takes care of initializing `scip->origprob->consnames`, an hash table used to store the constraints' names (`scip` and `origprob` are other internal structs of SCIP).
- `readRows()` processes the input file one row at a time.

Inside the latter function, each row is parsed by `mpsinputReadLines`() by iteratively updating an internal helper struct called `mpsi`. Among the others, the attributes `mpsi->fi` (`i=0,...,5`) are used to store relevant information parsed so far. In particular, `mpsi->f1` stores the type and `mpsi->f2` the name of the constraint that is currently being processed. After storing its type and name, `readRows()` goes on by checking whether the name has already been used by previously declared constraints, by querying the hash table.

To do so, the function `SCIPhashKeyValString()`, stored in `consnames->hashkeyval`, is called to generate the hash corresponding to the constraint's name. Internally, it casts the name it receives as input to a pointer variable called `str` and loops through the characters in the name to generate the hash.

When the input file is the corrupted one, the constraint name is `NULL`, hence the pointer variable `str` is `NULL` and the loop condition tries to de-reference a `NULL` pointer, causing a crash.

Q4. A possible fix could be to check whether a constraint is declared without specifying the name. For example:

```
diff --git a/scip/src/scip/reader_mps.c b/scip/src/scip/reader_mps.c
index 0c0b111..c06b45e 100644
--- a/scip/src/scip/reader_mps.c
+++ b/scip/src/scip/reader_mps.c
@@ -913,6 +913,9 @@ SCIP_RETCODE readRows(
          SCIP_Bool dynamic;
          SCIP_Bool removable;

+         if( mpsinputField2(mpsi) == NULL )
+            break;
+
          cons = SCIPfindCons(scip, mpsinputField2(mpsi));
          if( cons != NULL )
             break;
```

This would break out the `readRows()` main loop and raise a `SyntaxError` instead of a `segmentation fault` if a constraint is declared without specifying the name.

Part 2

Q5. The following file in `.lp` format causes a `segmentation fault`:

```
Subject to
    x00=0
sos
    S2::x00:0:0
```

Background: Special Ordered Sets (SOS) are used in linear programming to impose additional structure on the solution space. There are two main types of SOS:

- SOS of type 1 (SOS1), out of a particular set of variables, only one can be non-zero.
- SOS of type 2 (SOS2), out of a particular set of variables, at most two adjacent variables (based on their order) can be non-zero.

Correct syntax to declare SOS constraints is:

```
sos
    SOS2: S2:: x00:0 x11:1 ..
```

However, SCIP allows for omitted names and automatically generates artificial ones.

Q6. With assertions disabled, a crash happens at line `1447` in `scip/src/blockmemshell/memory.c` in the function `freeChkmemElement()` because the pointer variable `ptr` points to an invalid memory address and the expression `((FREELIST*)ptr)->next = chkmem->lazyfree;` deferences an invalid memory address.

With assertions enabled, `assert(lpinput->npushedtokens < LP_MAX_PUSHEDTOKENS)` fails at line `469` in `scip/src/scip/reader_lp.c` in the function `pushToken()`.

Q7. In order to understand the bug, it is important to understand how SCIP parses the input file. It uses the `lpinput` internal struct to store information retrieved from the input file. The main attributes are:

- `lpinput->token`, the current token being processed;
- `lpinput->tokenbuf`, to remember a token while looking ahead for dependencies;
- `lpinput->pushedtokens`, a stack of tokens to be processed;
- `lpinput->npushedtokens`, the number of tokens in the stack.

Inside `main()`, a sequence of nested functions are called up to `SCIPreadLp()`, which takes care of the parsing. Inside this function, three other important functions are called:

- `SCIPallocBlockMemoryArray()` dynamically allocates memory for the `lpinput` struct and its attributes. In particular, it allocates `LP_MAX_PUSHEDTOKENS` memory blocks of size `LP_MAX_LINELEN` for the `lpinput->pushedtokens` attribute. Notice that an expression of the form `lpinput->pushedtokens[i]` where `i>=LP_MAX_PUSHEDTOKENS` causes undefined behavior, as it tries to access out-of-bound memory.
- `readLPFile()` identifies the different sections of the input file and calls the appropriate helper function to parse it. In particular, `readSos()` parses the SOS section.
- `SCIPfreeBlockMemoryArray()` frees the previously allocated memory.

Inside the SOS section, there are 3 main elements the parser needs to identify and process:

- constraint name (if empty, SCIP generates an artificial one), e.g. `SOS1:`.
- constraint type, e.g. `S2:`.
- pairs of variable-weight, e.g. `x00:0`.

The function `readSos()` starts by looking for the constraint name, which is identified by a token followed by a colon. However, if there is a second colon following the first it means the name was actually omitted and the token represents the constraint type. Hence, an artificial name is created and both colons are pushed back into the stack to be processed later on in the constraint type section.

After having retrieved both name and type, a second loop takes care of parsing all the variable-weight pairs in the constraint. As soon as a token is not recognized as a variable, the loop is exited assuming this must be the name/type of a new constraint or the beginning of a new section.

When the input file is the corrupted one, this loop is exited after parsing the first pair `x00:0`, because a second `:` is encountered which is not recognized as a variable declared so far. However, this `:` is not followed by another colon, hence it's not even considered to be a constraint name. Moreover, the `:` makes the program take the omitted name branch with already two tokens in the stack (the original `:` token and the `0` following it). Hence, pushing `:` twice creates a pointer to out-of-bound memory which is stored in the `lpinput` struct, as we are trying to push more tokens than the `LP_MAX_PUSHEDTOKENS` limit.

A crash occurs later on as `SCIPreadLp()` tries to free the memory allocated for the `lpinput` struct, in particular the invalid memory location pointed to by `lpinput->tokenbuf`.

Q8. A possible fix could be to check whether `:` is passed as constraint name. For example:

```
diff --git a/scip/src/scip/reader_lp.c b/scip/src/scip/reader_lp.c
index 0af0862..5b9fb88 100644
--- a/scip/src/scip/reader_lp.c
+++ b/scip/src/scip/reader_lp.c
@@ -2237,6 +2237,13 @@ SCIP_RETCODE readSos(
       /* check if we reached a new section */
       if( isNewSection(scip, lpinput) )
          return SCIP_OKAY;
+
+      /* check the first token is not a colon */
+      if( strcmp(lpinput->token, ":") == 0 )
+      {
+         syntaxError(scip, lpinput, "Invalid constraint name ':'.");
+         return SCIP_OKAY;
+      }

       /* check for an SOS constraint name */
       *name = '\0';
```

This would stop `readSos()` execution and raise a `SyntaxError` if a colon is passed as constraint name.