Beispiel `simple_rpn_calc`

Dr. Günter Kolousek

14. Juli 2016

1 Allgemeines

- Drucke dieses Dokument **nicht** aus!
- Halte **unbedingt** die Coding Conventions ein! Zu finden am edvossh!
- Lege erstmalig ein Verzeichnis mit deinem Nachnamen und deiner Matrikelnummer in der Form `<nachname>_<matrikelnummer>` an, also z.B. `mustermann_i15999`.
- **Jedes** Beispiel ist in einem Unterverzeichnis `<beispielnummer_beispielname>` von dem vorhergehend angelegten Verzeichnis zu speichern! Für das aktuelle Beispiel heißt dieses Verzeichnis also `03_simple_rpn_calc`.
- Als Betriebssystem wird in diesem Jahr **ausschließlich** Linux verwendet.
- Am Anfang verwenden wir ausschließlich den vorgegebenen Texteditor (keine IDE)!

2 Aufgabenstellung

Schreibe ein `C++` Programm `srpn`, das einen RPN (reverse polish notation) Taschenrechner mit 4 Registern und den Grundrechenarten implementiert. Die Register werden immer mit 0.0 initialisiert bzw. nachgefüllt.
An Kommandos gibt es außerdem `print`, das den aktuellen Inhalt der Register ausgibt und `clear`, das alle Register löscht. Weiters beendet `quit` das Programm.
Nachfolgend siehst du eine Beispielsitzung, die gleichzeitig als Spezifikation gilt (vorausgesetzt wird das Verständnis über die prinzipielle Funktionsweise eines RPN Taschenrechners):

```
$ srpn
>>> +
```

```
0
>>> 1
>>> 2
>>> +
3
>>> 4
>>> +
7
>>> 5
>>> -
2
>>> 6
>>> *
12
>>> 3
>>> /
4
>>> 1
>>> 2
>>> 3
>>> 4
>>> +
7
>>> +
9
>>> +
10
>>> +
10
>>> 0
>>> /
division by zero
>>> print
r4: 0
r3: 0
r2: 0
r1: 10
>>> clear
>>> print
r4: 0
r3: 0
r2: 0
r1: 0
>>> a
```

```
invalid number or operator!
>>> quit
$
```

3 Anleitung

1. Schreibe ein minimales Programm `srpn`. Dieses Programm soll "nichts" tun. Der Dateiname der Quelldatei soll `main.cpp` heißen.

   Das Übersetzen lässt sich mit `g++ -o srpn main.cpp` erledigen.

   Starten kannst du es jetzt mit… `srpn`.

2. Implementiere jetzt den Prompt samt einer Eingabe (vom Typ `string`) und der Ausgabe der Eingabe in einer Endlosschleife. Die Eingabe soll vorerst in der folgenden Zeile ausgegeben werden. Abgebrochen wird das Programm derzeit noch mittels `CTRL-C`. Also ungefähr so etwas wie:

   ```
   $ srpn
   >>> a
   a
   >>> 2
   2
   >>> <CTRL-C>
   $
   ```

3. Erweitere jetzt das Programm, sodass es genau zwei Register gibt, also `double r1` und `double r2` und jede Rechenoperation genau aus zwei Zahlen besteht. D.h. es sind genau zwei Zahlen einzugeben und danach genau ein Operator (`+`, `-`, `*`, `/`). Damit soll der Ablauf vorerst nur folgendermaßen aussehen:

   ```
   >>> 1
   >>> 2
   >>> +
   3
   >>> 2
   >>> 4
   >>> *
   6
   ```

   Dazu ist die Schleife natürlich abzuändern, dass mehrmals hintereinander der Prompt in einem Schleifendurchgang ausgegeben wird. Das wird in weiterer Folge wieder anders werden. Fehler sollen **keine** abgefangen werden. Wir gehen vorerst davon aus, dass der Benutzer sich genau an diesen Ablauf hält.

Also zuerst in `r2` abspeichern, dann in `r1` und danach je nach eingegebenen Operator die beiden Register `r2` und `r1` addieren, subtrahieren, multiplizieren bzw. dividieren und das Ergebnis ausgeben.

Vergiss nicht, dass lokale Variable (fast) immer zu initialisieren sind!

4. Bauen wir jetzt die erste kleine Fehlerbehandlung ein. Ein ungültiger Operator soll zur richtigen Fehlermeldung gemäß der Beispielausgabe der Spezifikation des Programmes führen.

5. Teste einmal was passiert wenn du bei einer erwarteten Zahl zuerst (ein paar Mal) die Enter-Taste und danach erst die Zahl eingibst.

   Alles klar?

   Du kannst auch probieren vor oder nach einer Zahl einige Leerzeichen einzugeben. Das Ergebnis ist das Gleiche.

6. So, jetzt probiere einmal anstatt einer Zahl einen Buchstaben einzugeben. Was passiert? Teste!

   Die Schleife wird wirklich endlos mit einer dreifachen Ausgabe des Prompts und der Fehlermeldung durchlaufen. Das kann natürlich so nicht gewünscht sein.

   Aber woran liegt es?

   Ein Buchstabe kann nicht in eine Zahl gewandelt werden und der Eingabestrom `cin` wird in den Fehlermodus wechseln. Wie mit solchen Situationen umgegangen wird, werden wir uns jetzt *nicht* ansehen, da es in solch einem Fall sowieso besser ist anders vorzugehen. Weiter mit dem nächsten Punkt.

7. Baue die Schleife zurück, dass du wieder nur einmal den Prompt ausgibst und danach wieder eine Eingabe mit einer Variable vom Typ `string` hast.

   Um einen String, der eine Zahl enthält in eine Gleitkommazahl vom Typ `double` umzuwandeln, verwende die Funktion `stod`:

   ```
   string inp;  // to be filled with user input
   string num;

   num = stod(inp);  // String TO Double
   ```

   Die `if` Anweisung bleibt vorerst wie sie ist.

   Teste!

   Wenn du eine gültige Zahl eingibst, wird die Umwandlung korrekt funktionieren, aber die Überprüfung des Operators nicht. Gibst du einen gültigen Operator ein, dann die Umwandlung mit einer *Exception* abbrechen. Diese Exception wird – in gewohnter und bekannter Weise – das laufende Programm beenden. Um welche Exception handelt es sich?

   Das werden wir uns gleich im nächsten Punkt ansehen.

8. Da wir eine ungültige Zahl erkennen können wollen, müssen wir die Exception `std::invalid_argument` abfangen.

   Das geht folgendermaßen:

   ```
   try {
       num = stod(inp);
   } catch (invalid_argument) {
       // no valid number
       // here goes the code for checking operators
   }
   ```

   Jetzt sollte das Programm nicht mehr abstürzen.

9. Allerdings funktioniert das Programm so nicht mehr. Was muss geändert werden, damit Zahlen wieder addiert, subtrahiert, multipliziert und dividiert werden?

   Nach der Eingabe einer gültigen Zahl muss diese in das Register `r1` kopiert werden, allerdings darf der vorige Inhalt des Registers `r1` nicht verschwinden, sondern muss zuerst in das Register `r2` kopiert werden.

   Füge entsprechenden Code gleich nach `num = stod(inp)` ein.

   Jetzt sollte das Programm wieder besser funktionieren, nämlich insoferne, dass z.B. bei der Eingabe von 1 und 2 auch das erwartete Ergebnis der entsprechenden Operation ausgegeben wird.

10. Soweit sogut, aber bei der Abarbeitung der Rechenoperationen reicht es nicht aus, dass das Ergebnis berechnet und ausgegeben wird. Es muss auch noch irgendwo abgespeichert werden, damit es bei der nächsten Rechenoperation wieder zur Verfügung steht.

    Der Vorgang ist folgender:

    a) Das Ergebnis ausrechnen und in `r1` abspeichern.

    b) `r2` mit 0 belegen.

    c) Das Ergebnis auf ausgeben.

11. Die Ausgabe und das Setzen von `r2` mit 0 muss nicht in jedem Zweig der `if` Anweisung durchgeführt werden. Das kann ruhig danach passieren. Allerdings gibt es eine Kleinigkeit, die man beachten muss. Gibt man nämlich einen ungültigen Operator ein, dann werden auch diese beiden Anweisungen ausgeführt.

    Das kann verhindert werden, wenn im `else` Zweig nach der Ausgabe der Fehlermeldung eine `continue` Anweisung eingefügt wird.

12. So, jetzt sollten die Basisoperationen mit genau zwei Zahlen so funktionieren, aber das ergibt nur ein ein sehr eingeschränktes Arbeiten.

    Jetzt erweitern wir von den 2 Registern auf die 4 Register, wie in der Spezifikation vorgegeben.

Dazu muss man natürlich wissen wie ein RPN Rechner funktioniert:

- Die Eingabe einer Zahl bewirkt, dass diese in das Register 1 geschrieben wird. Der ursprüngliche Wert von `r1` wird zuerst in `r2` geschrieben. Aber bevor das getan wird, wird der Wert von `r2` in `r3` geschrieben. Genauso wird auch mit `r3` verfahren, womit der Wert von `r4` "verloren" geht.

  Das bedeutet, dass der neue Wert die anderen Werte "verschiebt".
- Bei der Berechnung einer Operation werden die Werte von `r2` und `r1` entsprechend der Operation verknüpft und das Ergebnis in `r1` gespeichert. Da die beiden Werte von `r2` und `r1` jetzt "verbraucht" sind und das Ergebnis in `r1` abgelegt worden ist, kann man die anderen Register Richtung `r2` verschieben. Also `r3` kommt in `r2` und `r4` kommt in `r3` und `r4` wird mit 0 belegt.

Programmiere!

Jetzt sollte das Programm schon mit den Grunkoperationen gemäß der Spezifikation funktionieren.

13. Jetzt ist ein guter Zeitpunkt sich wieder anzusehen, wie das Programm derzeit auf "Sonderfälle" reagiert.

    Probiere einmal aus $1 \div 0$ auszurechnen und probiere auch $-1 \div 0$.

    - Was siehst du bei deinem Programm? $\rightarrow$ `inf`
    - Was bedeutet `inf`? $\rightarrow$ infinity, zu dt. unendlich
    - Warum bekommen wir keinen Fehler? Kopiere dein Programm kurz um (z.B. in die Datei `main.cpp.bak`) und ändere das Programm so um, dass `int` anstatt `double` als Typ für die Zahlen verwendet wird.

      Was passiert jetzt bei der Division von 1 durch 0?

    Gehe daher her und teste ob das Ergebnis ein positives oder negatives `inf` ist indem du die Funktion `isinf(x)` aufruft und in diesem Fall eine entsprechende Meldung ausgibst und wieder die `continue` Anweisung bemühst.

14. Teste gleich einmal, ob du nach einer Division von 1 durch 0 anschließend die ursprünglich eingegebenen Zahlen 1 und 0 addieren kannst:

```
>>> 1
>>> 0
>>> /
division by zero
>>> +
1
```

    Wenn das bei dir nicht so funktioniert, dann finde den Fehler und bessere diesen Fehler umgehend aus!

15. Nächster Sonderfall gefällig? Probiere $0 \div 0$! Und adaptiere wiederum dein Programm! Ach ja, `nan` bedeutet "not a number".

16. Noch eine Fehlersituation gefällig? Gib z.B. einmal als Zahl "3a" ein. Hmm, das kleine "a" wird nicht erkannt. Es wird einfach überlesen. Das liegt an der Funktion `stod`, die nach einer gültig erkannten Zahl abbricht und den Rest des Strings ignoriert.

    Gibt man der Funktion aber einen weiteren Parameter mit, dann kann man die Anzahl der als gültig erkannten Zeichen ermitteln und diese mit der Länge des Strings vergleichen:

    ```
    string inp;  // to be filled with user input
    size_t pos;  // size_t used for size details

    num = stod(inp, &pos);
    if (pos != inp.size()) {
        // inp contains invalid characters
    ```

17. Ein Punkt ist noch offen: Was passiert wenn du `CTRL-D` drückst? Probiere es aus!

    `CTRL-D` bewirkt unter Unix/Linux das Schließen des Eingabestroms `cin` und damit kann man im Programm auch nicht mehr weiter von `cin` lesen. In diesem Fall muss/soll sich das laufende Programm natürlich auch beenden.

    Die Überprüfung, ob ein Ein- bzw. Ausgabestrom sich in einem "guten" Zustand befindet kann ganz einfach folgendermaßen durchgeführt werden:

    ```
    if (cin)
    ```

    Das funktioniert wie wir es schon kennen (Alles ungleich "Null" ist wahr). Nur ist hier eher das Gegenteil interessant, da wir die Schleife abbrechen wollen, wenn sich der Eingabestrom nicht mehr in einem "guten" Zustand befindet.

18. Jetzt ist es einfach: Implementiere noch die Befehle `print`, `clear` und `quit`!

4 Übungszweck dieses Beispiels

- Auffrischen der Programmierkenntnisse
- Einfaches `C++` Programm
- Übersetzen von der Kommandozeile
- Einfache Ein- und Ausgabe: `cin`, `cout`, `<<`
- Umwandeln von String zu Zahl: `stod`
- `double`, `string`, `string::size()`, `size_t`

- Initialisieren von lokalen Variablen
- Einfache Rechenoperationen
- `if`, `else`, `else if`
- `while`, `break`, `continue`
- `try`, `catch`
- Verwendung eines Texteditors