CPPLS1 Eindopdracht
Speuren met Krul

Bob Polis, Avans Hogeschool, 's-Hertogenbosch

30 september 2020

Inhoudsopgave

1 Wat ga je bouwen?

Ergens, op een geheime plek op het web, staat een bericht dat je moet zien te vinden. Je krijgt een start-URL, waar je via een HTTP GET-request een stukje platte tekst van ophaalt.

Deze tekst is *source code*, geschreven in een door mij verzonnen mini-taal, *Krul* genaamd, voor een door jou te bouwen *interpreter*.

Als je de code runt geeft dat een *string* als resultaat. Die string is een bestandsnaam waarmee je opnieuw een URL samenstelt, waarvandaan je een volgend stukje Krul-code kunt downloaden.

Zo ga je door, totdat je een speciale instructie in de code tegenkomt die aangeeft dat je de oplossing gevonden hebt. De laatst samengestelde string is dan het geheime bericht.

Je bouwt hiervoor een command-line tool (*console application*) in C++, die de uiteindelijke oplossing in de terminal rapporteert.

Ter controle laat je in je output steeds elk resultaat van een stukje code ook zien in de terminal, zodat je dus uiteindelijk een opsomming ziet van bestandsnamen, gevolgd door het gevonden geheime bericht.

2 HTTP GET-requests met `libcurl`

Je gebruikt voor het doen van HTTP GET-requests een immens populaire library, `libcurl`[1] genaamd, die voor elk platform te krijgen is. Die library stelt je in staat om allerlei soorten requests te doen, zodat je bijvoorbeeld gemakkelijk webpagina's en data van API's kunt ophalen.

`libcurl` is een C-library, dus je zult in elk geval een C++-class moeten schrijven die de verbinding met jouw C++-code in goede banen leidt.

3 De interpreter voor Krul

De interpreter die je gaat schrijven is niet ingewikkeld, maar er zitten wel een paar leuke uitdagingen in om hem netjes geïmplementeerd te krijgen.

Krul maakt gebruik van *reversed polish notation (RPN)*[2].

Bijvoorbeeld, de volgende code:

```
2
3
add
4
mul
```

levert "`20`" als resultaat.

[1] `https://curl.haxx.se/libcurl/`

[2] `https://en.wikipedia.org/wiki/Reverse_Polish_notation`

In je implementatie gebruik je een stack om waarden op te zetten en vanaf te halen. Het is handig om daar een `std::vector` voor te gebruiken.

Bovenstaand voorbeeld is dan gemakkelijk te verwerken: je zet de waarde `2` op de stack, dan de waarde `3`, en vervolgens roep je een functie aan om de "`add`"-instructie te interpreteren. Die haalt de twee waarden van de stack, telt ze op, en zet het resultaat "`5`" op de stack.

Daarna wordt `4` op de stack gezet, en wordt de instructie "`mul`" uitgevoerd. Die haalt twee waarden van de stack (`5` en `4` in dit geval), vermenigvuldigt ze, en zet het resultaat "`20`" op de stack.

3.1 Gedetailleerde beschrijving van Krul

Alle waarden op de stack zijn strings.

Dat betekent dat numerieke berekeningen altijd eerst de benodigde waarden naar integer converteren voordat ermee gerekend kan worden. Ook moet een numeriek resultaat naar string geconverteerd worden alvorens het op de stack wordt gezet.

Na de laatste regel code, als er geen verdere instructies meer zijn, stopt het programma vanzelf. Als de stack niet leeg is, is de laatste waarde op de stack het eindresultaat van dit Krul-programma.

3.1.1 Values & Types

`42` *Digits: integer literal.* Zet het gegeven integer getal, geconverteerd naar string, op de stack. Er zijn alleen positieve integers of de waarde `0` toegestaan. (Voor negatieve getallen wordt een positief getal gegeven, gevolgd door een "`neg`"-instructie.) Het getal `42` is hier slechts een voorbeeld.

`\text` *Text to end of line.* Beschouw alle tekst na de backslash tot het einde van de regel als een letterlijke string, en zet die op de stack.

`:label` *Label definition.* Beschouw de tekst na de dubbele punt tot het einde van de regel als een label-naam, en onthoud die, samen met de volgende regelpositie, zodat je later naar die regel kunt springen.

`>label` *Label reference.* Zet de *waarde* van het label op de stack. Dit is een door jou gekozen representatie van de programmaregel die volgde op de labeldefinitie.

`=var` *Variable assignment.* Beschouw alle tekst na het "`=`"-teken tot het einde van de regel als de naam van een variabele, waaraan je de huidige waarde op de stack toekent. Verwijder vervolgens de waarde van de stack.

`$var` *Variable reference.* Zet de waarde van de variabele op de stack.

Elk andere letterlijke tekst wordt opgevat als een instructie voor de machine. Alle mogelijke machine-instructies staan hieronder.

3.1.2 Integer operaties

`add` *Optellen.* Haal 2 waarden van de stack, converteer naar integer, tel ze op, converteer naar string, en zet het resultaat weer op de stack.

`sub` *Aftrekken.* Haal 2 waarden van de stack, converteer naar integer, trek de tweede van de eerste af, converteer het resultaat naar string, zet het op de stack.

`mul` *Vermenigvuldigen.* Haal 2 waarden van de stack, converteer naar integer, vermenigvuldig ze, converteer naar string, en zet het resultaat op de stack.

`div` *Delen met afkappen.* Haal 2 waarden van de stack, converteer naar integer, deel de tweede door de eerste, converteer het resultaat naar string en zet dit op de stack.

`mod` *Modulo.* Haal 2 waarden van de stack, converteer naar integer, bereken `tweede % eerste`, converteer het resultaat naar string en zet dat op de stack.

`neg` *Negate.* Haal één waarde van de stack, converteer naar integer, keer het teken om (van plus naar min of omgekeerd), converteer het resultaat naar string en zet op de stack.

`abs` *Absolute waarde.* Haal één waarde van de stack, converteer naar integer, bereken de absolute waarde, converteer het resultaat naar string en zet op de stack.

`inc` *Increment.* Haal één waarde van de stack, converteer naar integer, tel er één (`1`) bij op, converteer naar string en zet op de stack.

`dec` *Decrement.* Haal één waarde van de stack, converteer naar integer, trek er één (`1`) van af, converteer naar string en zet op de stack.

3.1.3 String operaties

`dup` *Duplicate.* Lees één waarde van de stack (haal hem er *niet* af), en zet deze nogmaals op de stack.

`rev` *Reverse.* Haal één waarde van de stack, draai de string om, en zet het resultaat op de stack.

`slc` *Substring.* Haal 3 waarden van de stack, achtereenvolgens *to*, *from*, en de *waarde* waarvan we een substring willen nemen. De *from*- en *to*-waarden worden naar integer geconverteerd, zodat je ze op de juiste wijze kunt toepassen in de `substr()`-method van `std::string`. De *to*-waarde is de string-index die nèt niet meer meedoet. Het is dus van-tot, niet van-tot-en-met.

Let op: dit is niet helemaal hetzelfde als wat `substr()` verwacht.

`idx` *Indexeren.* Haal 2 waarden van de stack: eerst een index die je naar integer converteert, dan de string-waarde waaruit een character wordt genomen. Zet het correcte character als string op de stack.

`cat` *Concatenation.* Haal 2 waarden van de stack, bouw een string door de tweede achter de eerste te plaatsen en zet het resultaat op de stack.

`len` *Lengte.* Haal één waarde van de stack, bepaal de length van deze string, converteer die naar string en zet op de stack.

`rot` *Roteren.* Haal één waarde van de stack, voer er een rot-13[3] transformatie op uit en zet het resultaat op de stack.

`enl` *Add a newline.* Haal één waarde van de stack, voeg daar een newline (`'\n'`) aan toe, en zet het resultaat op de stack.

3.1.4 Tests & Jumps

`gto` *Goto.* Haal één waarde van de stack, interpreteer die als nodig om de volgende code van de regel te lezen die door deze label-waarde wordt gerepresenteerd.

`geq` *Goto if equal.* Haal 3 waarden van de stack, achtereenvolgens: *label-waarde*, *val2* en *val1*. Als *val1* en *val2* gelijk zijn, vervolg het interpreteren van instructies dan vanaf de locatie die gerepresenteerd wordt door de *label-waarde*. Zo niet, ga dan gewoon door met de volgende instructie.

`gne` *Goto if not equal.* Haal 3 waarden van de stack, achtereenvolgens: *label-waarde*, *val2* en *val1*. Als *val1* en *val2* ongelijk zijn, vervolg het interpreteren van instructies dan vanaf de locatie die gerepresenteerd wordt door de *label-waarde*. Zo niet, ga dan gewoon door met de volgende instructie.

`glt` *Goto if less.* Haal 3 waarden van de stack, achtereenvolgens: *label-waarde*, *val2* en *val1*. Converteer *val1* en *val2* naar integer. Als *val1* kleiner is dan *val2*, vervolg het interpreteren van instructies dan vanaf de locatie die gerepresenteerd wordt door de *label-waarde*. Zo niet, ga dan gewoon door met de volgende instructie.

[3] https://en.wikipedia.org/wiki/ROT13

`gle` *Goto if less or equal.* Haal 3 waarden van de stack, achtereenvolgens: *label-waarde*, *val2* en *val1*. Converteer *val1* en *val2* naar integer. Als *val1* kleiner of gelijk is aan *val2*, vervolg het interpreteren van instructies dan vanaf de locatie die gerepresenteerd wordt door de *label-waarde*. Zo niet, ga dan gewoon door met de volgende instructie.

`ggt` *Goto if greater.* Haal 3 waarden van de stack, achtereenvolgens: *label-waarde*, *val2* en *val1*. Converteer *val1* en *val2* naar integer. Als *val1* groter is dan *val2*, vervolg het interpreteren van instructies dan vanaf de locatie die gerepresenteerd wordt door de *label-waarde*. Zo niet, ga dan gewoon door met de volgende instructie.

`gge` *Goto if greater or equal.* Haal 3 waarden van de stack, achtereenvolgens: *label-waarde*, *val2* en *val1*. Converteer *val1* en *val2* naar integer. Als *val1* groter of gelijk is aan *val2*, vervolg het interpreteren van instructies dan vanaf de locatie die gerepresenteerd wordt door de *label-waarde*. Zo niet, ga dan gewoon door met de volgende instructie.

3.1.5 Functies

`fun` *Functie.* Bewaar de locatie van de volgende instructie op een aparte stack (de *call stack*), en voer een `gto` uit.

`ret` *Return.* Haal de laatste locatie van de *call stack*, en zorg dat de volgende instructie vanaf die locatie wordt uitgevoerd.

3.1.6 Eindoplossing

`end` *End of search.* Waarde op de stack is het uiteindelijke geheime bericht. Deze instructie komt alleen in de laatste file die je ophaalt voor.

3.2 Voorbeeld

`3`	*zet 3 op de stack*
`=cnt`	*haal 3 van de stack en ken toe aan* `cnt`
`\Hello, world`	*zet "Hello, world" op de stack*
`enl`	*voeg daar een newline aan toe*
`=hello`	*ken het resultaat toe aan* `hello`
`\`	*zet een lege string op de stack*
`=result`	*ken die toe aan* `result`
`:loop`	*definieer de volgende stap als locatie* `loop`
`$result`	*zet* `result` *op de stack*

`$hello`	*zet `hello` op de stack*
`cat`	*maak er één string van*
`=result`	*ken die toe aan `result`*
`$cnt`	*zet `cnt` op de stack*
`dec`	*trek daar 1 van af*
`=cnt`	*ken dat weer toe aan `cnt`*
`$cnt`	*zet `cnt op de stack`*
`0`	*zet `0` op de stack*
`>loop`	*zet een verwijzing naar locatie `loop` op de stack*
`ggt`	*als `cnt > 0`, ga naar `loop`, anders verder*
`$result`	*zet `result` op de stack — dit is het eindresultaat*

3.3 Tips

- Laat je niet afschrikken door de lijst *instructies* die je moet implementeren, dat is minder werk dan je denkt. Elke instructie is steeds een korte functie met een paar regels recht-toe-recht-aan code.

- Een stack kun je makkelijk met een `std::vector`[4] implementeren. Gebruik de methods `push_back()` en `pop_back()` om objecten op de stack te zetten of er van af te halen. Gebruik `back()` om de laatste waarde op de stack te lezen zonder die er van af te halen.

- Een `std::map`[5] kun je gebruiken voor de opslag van *key-value-pairs*. Stel, je wilt een *map* gebruiken die strings als sleutel gebruikt en integers als waarde, declareer en gebruik je die als volgt:

```
std::map<std::string, int> dingen;
dingen["een"] = 1;
dingen["twee"] = 2;
std::cout << dingen["een"] << '\n';
```

- Gebruik bij voorkeur `std::string` om tekst te representeren. Blijf zoveel mogelijk weg van C-strings (character arrays).

- Voeg eventueel een "**`out`**" instructie aan de interpreter toe, die de laatste waarde op de stack naar `std::cout` schrijft. Dat is handig voor testen en debuggen.

[4] https://en.cppreference.com/w/cpp/container/vector
[5] https://en.cppreference.com/w/cpp/container/map

- Voor vragen over de organisatie van deze eindopdracht richt je je tot Bob Polis. Kijk eerst bij de opdracht in Blackboard of de antwoorden daar te vinden zijn.
- Voor vragen over de inhoud van deze eindopdracht richt je je, na het doorzoeken van je boek en/of Google, tot Bob Polis.

4 Requirements

1. Je maakt een command-line tool (*console application*) in C++.
2. Je haalt steeds platte tekst binnen vanaf een URL die als basis
   `https://student.aii.avans.nl/doc/rpbpolis1/cpp1/`
   heeft.
3. De eerste complete URL waar je een mini-programma van ophaalt wordt gevormd door de bestandsnaam "`start.txt`" achter de basis-URL te zetten. Dat wordt dus:
   `https://student.aii.avans.nl/doc/rpbpolis1/cpp1/start.txt`
4. Je bouwt een interpreter voor de in dit document beschreven taal.
5. Je interpreteert het ontvangen mini-programma, wat leidt tot een string als resultaat.
6. De gevonden string is de volgende bestandsnaam die je achter de basis-URL plaatst om een nieuwe URL te krijgen. Krijg je als resultaat uit het programma de tekst "`hoera.txt`", dan wordt de volgende URL waar je code vandaan haalt dus:
   `https://student.aii.avans.nl/doc/rpbpolis1/cpp1/hoera.txt`
7. Rapporteer de gevonden URL ter controle in de terminal.
8. Zo ga je door met programma-tekst ophalen, todat je in een programma de "`end`" instructie tegenkomt. In dat geval is de resulterende string de oplossing van deze speurtocht.
9. Je gebruikt een *memory leak detection tool* om aan te tonen dat je programma geen memory leaks heeft.
10. Je maakt op correcte wijze zoveel mogelijk gebruik van `const`.
11. Voor resource management gebruik je een daarvoor geschikte RAII-class.

12. Je programma maakt gebruik van exception handling.
13. De enige externe library die is toegestaan is `libcurl`.[6]

5 Beoordeling

- **Dealbreakers**: niet voldaan aan technische requirements 1, 9, 11, 12, 13. Je scoort dan een 1,0.
- Je hebt minimaal een 7 wanneer het juiste resultaat wordt gevonden met het ingeleverde programma.
- Extra punten voor correct C++ idioom:
  - mate van *const correctness*: +1 punt wanneer alles wat `const` zou moeten zijn dat ook is; +0,5 punt wanneer je gelegenheden hebt laten liggen, of wanneer je het juist overdreven toepast in gevallen waar het geen betekenis heeft (zoals bij primitieve datatypen voor functieparameters).
  - gebruik van *initialisation lists* en/of *delegating constructors*: +1 punt indien consequent gebruikt voor elke class; +0,5 punt wanneer wel toegepast, maar niet overal waar het kan.
- Extra punten (max. +2) voor aanwezigheid van (unit) tests voor het gedrag van de interpreter, dus voor:
  - correct gedrag van elke instructie (daar zitten vanzelf tests voor correct label-gebruik bij, anders werken de goto's niet)
  - herkennen van integer literals
  - herkennen van string literals
  - waarde toekennen aan variabelen
  - variabelen gebruiken

  Dit hoeven niet persé echte unit tests te zijn in de zin van dat je een officieel unit test framework erbij haalt, zoals GTest[7]. Je kunt ook je programma geschikt maken voor het runnen van (eigengemaakte) lokale Krul scripts, en dan met bijvoorbeeld shell scripts testen of je de verwachte output krijgt.

  Wanneer de tests niet alle gevallen afdekken krijg je proportioneel minder van het extra punt. Dus: helft van de gevallen gedekt: +1,0 punt; kwart van de gevallen gedekt: +0,5 punt; etc.

[6] De C++ Standard Library mag natuurlijk wel: die beschouwen we niet als *externe* library.
[7] https://github.com/google/googletest