Exercise Session

Week 09

Adel Gavranović
agavranovic@student.ethz.ch

Overview

polybox for session material mail to TA

Today's Topics

Introduction

Recursion

(Extended-)Backus-Naur-Form

Structs

Follow-up

- Don’t worry about that one ”recursion sequence” exercise. It was hard, but we TAs found it was a universal experience and that basically no group was able to solve it, especially not in that little time. It’s still a good practice example.

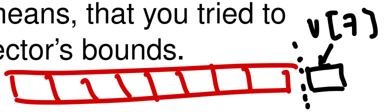
Comments on last [code]expert Exercises

- the ”semantically invalid or invalid during runtime” exercise was rather difficult. I know it’s very hard to tell apart one kind of mistake from another. I recommend just learning the ”types of mistakes” by heart and after a while you will be able to differentiate the two (after you’ve practiced more and encountered more mistakes)
- If you feel like you have trouble solving most exercises every week, I *highly* recommend going to the Study Center. There you can ask questions much more frequently and directly (than via mail)
- I’m sorry if your code suddenly changes its grading. Sometimes I find mistakes later because I forgot to check the first time

Comments on last `[code]expert` Exercises

print (const std::vector&v)

- `const`: use it whenever a function is only supposed to read something. Common functions that do this: `print(const...)`, `count(const...)`, `get(const...)`
- `Segmentation faults`: usually means, that you tried to access something outside of a vector's bounds. (`v.lenght() == 7`, dann `v[7]`)



- Sometimes, a super-long written feedback is not the optimal way of helping you, which is why I sometimes tell you to email me specific questions (no matter how silly the question might be) or visit the Study Center

Comments on last [code]expert Exercises

- the feedback on your first bonus exercise was a bit meager. That's because I need more input from you. What did *you* think was hindering you from achieving 100%? In general, leave comments with questions at the very top of your code and share your thoughts. You can also always write me a mail or go to the Study Center
- *Global Variables*: are variables, that are not in a function and can be accessed by all functions. Don't use them, but rather learn how to implement references properly

Questions or Comments re: Exercises?

Learning Objectives Checklist

Now I...

- ☐ can check whether a character sequence matches given EBNF rules
- ☐ can write EBNF rules that accept only a given set of input character sequences
- ☐ can define and use `C++` structs
- ☐ can define functions that manipulate `C++` structs

Call Graphs

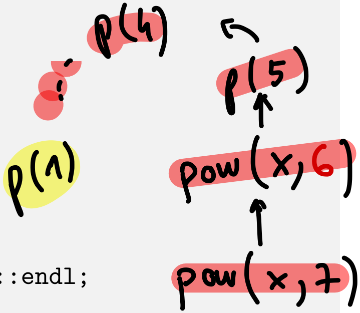
Are a way to visualize function calls `"t()"`

Call Graph for `power(x, 7)`

$x^7 = x^6 \cdot x$

```
// PRE:  base x, power n
// POST: n'th power of x
unsigned int power(const int x, const unsigned int n){
   if(n == 0){
      return 1;
   }else if(n == 1){
      return x;
   }

   return x*power(x, n-1);
}
...
std::cout << power(x, 7) << std::endl;
...
// How will the "Call Graph" look like for this
    function?
// How many times will power() be called in total?
```



You've got the `power(x,n)`

$\longrightarrow$ temp = pow(x, $\frac{n}{2}$)

;

$\longrightarrow$ return temp · temp

return pow(x, $\frac{n}{2}$) · pow(x, $\frac{n}{2}$

Task

- Come up with a better (fewer function calls) implementation of the `power()`-function with pen and paper
  Hint: $x^n = x^{\frac{n}{2}} \cdot x^{\frac{n}{2}}$
- Implement it in `[code]expert` in groups (Breakout Rooms)
- Share your results and analyze the number of function calls your solution does

Master Solution for Power Function

$p(x) \to p(k)$

```
// POST: result == x^n
unsigned int power (const int x, const unsigned int n){
  if (n == 0) {
    return 1;
  } else if (n == 1) {
    return x;
  } else if (n % 2 == 0) {
    int temp = power(x, n/2);
    return temp*temp;
  } else {
    return x*power(x, n-1);
  }
}
```

$$\begin{tikzcd} \text{pow}(x, 32) \arrow[d] \\ \text{pow}(x, 16) \arrow[d] \\ \text{pow}(x, 8) \end{tikzcd}$$

4, 2, 1

$\log_2(32$

6

This function will call itself *at most* $2\log_2(n+1) - 1$ times, i.e. only logarithmically many function calls compared to linearly many with the other implementation.

Questions?

The Concept of (E)BNF

- is a way to go through an input (e.g. `std::cin`, or a file) and analyze if it is valid according to our given (E)BNF
- works *recursively*
- the **E** in EBNF stands for extended. **E**BNFs offer a few additional (shorter/better) ways of describing what a *valid* input is
- a lot of exercises will be of the form *"is xoo-xooxoo valid according to the given EBNF?"*
- think of them as *rules for allowed words* (but these "words" look weird af)
- good videos on that topic: ▸ EBNF I ▸ EBNF II
  (small mistake in EBNF I @6:00: in `digit` all the numbers should be seperated by |s

BNF Example ”Aa_”

We want to define a BNF that encompasses the following rules:

Rules

Alphabet = {’A’, ’a’, ’_’}
’A’ can only appear directly after an underscore or as the very first symbol. And underscores cannot occur in pairs and cannot be placed as the very first or the very last symbol.

For example, the following sequence is valid: ”Aaaaa_aa”, but not ”AaaAa”. Our task is now to come up with a BNF expressing such (valid/allowed) sequences

BNF Example "Aa_"

OR

exactly _

BNF

```
rules → seq          = term | term '_' seq
        term         = 'A' | 'A' lowerterm | lowerterm
        lowerterm    = 'a' | 'a' lowerterm        a or aaa...
```

When checking if a *word* is valid, try to deconstruct it bit by bit with the given BNF. This BNF has 3 rules: the first and last one have two alternatives, the second one has three. This BNF has 3 *non-terminal* symbols (`seq`, `term`, `lowerterm`) and 3 *terminal* symbols ('A', 'a', '_')

Questions?

EBNF

→ code

tricky (cuz recursion)

Task

Rewrite the BNF from the previous slides into an EBNF with the follwing additional syntax:

- `{...}`: at the location of this syntax, the content between the brackets can be repeatet $n \in \{\mathbb{N}_0\}$ times 0,1,2 ..
- `[...]`: at the location of this syntax, the content between the brackets can be repeatet $m \in \{0, 1\}$ times

EBNF

```
seq   = term ['_' seq]
term  = 'A' {'a'} | 'a' {'a'}
```

(not unique)

alternative: ['A']{'a'}

not really need

actually, no
-Jan
because empty is not vali

Exercise ”Valid Words”

Aa_Aa valid . ✓

Task

```
seq    = term ['_' seq]
term   = 'A' {'a'} | 'a' {'a'}
```

Which of the following concatenations are *valid* seqs in the sense of the EBNF above?

1. A ✓
2. a ✓
3. _ ✗ inv. cut. "_" ≠ term
4. Aaa ✓
5. ~~aaA~~ ✗ ✓
6. A_A ✓
7. Aa_Aa

(valid: 1, 2, 4, 6, 7)

Exercise ”Valid Words” Helper Functions

```
// PRE: valid input stream input
// POST: returns true if further input is available
//       otherwise false
bool input_available(std::istream& input);

// PRE: valid input stream input
// POST: the next character at the stream is returned
    (but not consumed)
//       if no input is available, 0 is returned
char peek(std::istream& input);
```

"is"

Adelint

Exercise ”Valid Words” Helper Functions

```
// POST: leading whitespace characters are extracted
//       from input, and the first non-whitespace
    character is returned (but not consumed)
//       if an error or end of stream occurs, 0 is returned
char lookahead(std::istream& input);

// PRE: Valid input stream input, expected > 0
// POST: If ch matches the next lookahead then it is
    consumed and true is returned
//       otherwise no character is consumed and false is
    returned
bool consume(std::istream& input, char expected);
```

is_z

consume(is, E)

~~e~~xercise_session

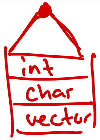
Let’s Code Together!

- Help me code a solution to this
- Ask, whenever something is unclear or weird to you
- Feel free to answer the questions of your fellow students, if you think you know the answer

Questions?

Structs

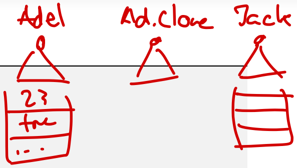
struct :



Structs are bundles of stuff mental image

- this stuff can be types, functions and more (”members”)
- the types don’t have to be the same
- structs are our way of creating new ”things”, like our own type of number (complex numbers), mathematical structures (lines, squares, circles) or things (person data)

Structure of strucs



```
struct Person {
    unsigned int age;
    bool alive;
    std::vector<int> LuckyNumbers;
};

int main () {
    Person Adel = {23, true, {6,7,9}}; ("new instance of "Person"
    Person Adel_clone = Adel;  // all elements are copied
    Person Jack = {21, true, {1,2,6}};
    std::cout << "Adel's " << Adel.age << " years old\n";

    [illegible] 1 2 return 0; (mistake in handout)
}
```

Exercise "Geometry Exercise"

try :)

Task

- open the exercise "Geometry Exercise" in [code]expert
- how would you write the function `sum`?
- how would you write the struct `line`?
- how would you write the function `shift_line`?
- try to implement them

Questions?