Question 1 - Language Design and Control Structures

Language Design Criteria - Slides: 3

Explains concept: If you fulfil the criteria it adds on of the three values to your language.

Criterions: **Readability**: has to be in the context of the problem domain, using terms and such and syntax that makes sense , **Writability**: how easily a language can create programs for a chosen problem domain, **Reliability**: Is this if a program performs to its specifications under all conditions. Not included: **Cost**: the total cost of a programming language: cost of training programmers, writing in the language, cost of compiling in the language.

Characteristics:

- Simplicity: is not when: too many features or feature multiplicity: mutiple ways of doing 1 ting. Operator overloading. Too many features can lead to misuse or accidental use. Example: *count* = *count* +*1*, *count*++ and *count* += *1*.
- Orthogonality: relatively small set of primitive constructs and can be combined in a relatively small number of ways. Too many features can lead to misuse or accidental use.
- Data types: enough datatypes to do the desired. Example: using integers to define flags because there is no Booleans in the language.
- Syntax design: Syntax that enhance readability: Special words and form and meaning (make something reflect its purpose/use example: +, *cd* for *change directory*)
- Support for abstraction:
- Expressivity: writing less for expressing something: using for-loops for iteration instead of while. *count*++ instead of *count* = *count* + *1*.
- Type checking: testing for type errors in a given program during compile-time or run-time.
- Exception handling: Being able to intercept exceptions, handle and continue on run-time.
- Restricted aliasing: having two distinct names that can be used to access the same memory cell.

Evaluation of expressions - Slides: 5, 6, 7, 8 + tegn??????

Go though slides.

Explicit sequence control vs. structured sequence control - Slides: 4

Explicit: goto, continue, break

Structured: for, while, if-else

Explicit sequence control vs. structured sequence control - Slides: 4??

Loop constructs - Slides: 12, 13

For: When should what be evaluated or incremented? While: pretest or posttest? Used as for-loop or something else?

Subprogram - Slides: 17, 18

Go though slides. Talk about project implementation.

Parameter mechanisms - Slides: 19, 20

Go though slides.

Question 2 - Structure of the compiler 3-19, 21-28

Describe the phases of the compiler and give an overall description of what the purpose of each phase is and how the phases interface

Slides: 3-19

Single pass vs. multi pass compiler

Slides 21-23

Issues in language design

Slides 24-27

Issues in code generation

Slide 28

Question 3 - Lexical analysis 3, 6-9, 10-17 (bonus: 18)

Describe the role of the lexical analysis phase

Slides: 3

Taking input source program and making into a stream of tokens. Uses token specification described with regular expressions to translate the given program.

Describe how a scanner can be implemented by hand or auto-generated

Slides: 6-9 !! BLIV BEDRE TIL DISSE!

Describe regular expressions and finite automata

Slides: 10-17 (bonus: 18) !! BLIV BEDRE TIL DISSE!

Question 4 - Parsing 3, 4, 5, 6-7, 13-17

LL(1): *In LL1, first L stands for Left to Right and second L stands for Left-most Derivation. 1 stands for number of Look Aheads token used by parser while parsing a sentence.*

Describe the purpose of the parser

Slide 3

Analysis the tokens coming from the scanner and creates a AST from them. Uses grammar to confirm tokens and create tree.

Discuss top down vs. bottom up parsing

4

They both use the grammar to construct the AST but one does it from the leafs and one from the root node.

Top-down: uses left most derivation: Tries to revolve a rule into its left most child. (we expand the start symbol to the whole program.) **Bottom-up**: uses right most derivation: will select rules to try to reduce the input string. (in this Parsing technique we reduce the whole program to start symbol.)

Explain necessary conditions for construction of recursive decent parsers

5

It is a top-down parser but requires the following:

- No left recursion
- No left factoring (multiple rules starting the same symbol: e -> T x | T y | T z)

BEFORE REMOVING LEFT RECURSION	AFTER REMOVING LEFT RECURSION
E –> E + T | T	E –> T E'
T –> T * F | F	E' –> + T E' | e
F –> (E) | id	T –> F T'
	T' –> * F T' | e
	F –> (E) | id

Discuss the construction of an RD parser from a grammar

6-7 explain slides.

Discuss bottom Up/LR parsing

13-17 !! BLIV BEDRE TIL DISSE!

Question 5 - Semantic Analysis 3, 4-17, 19 bonus 20

Describe the purpose of the Semantic analysis phase

Slide: 3

It verifies the parse tree, whether it’s meaningful or not. It furthermore produces a verified parse tree. It also does type checking, Label checking and Flow control checking.

Slide 4-17, 19 bonus 20 !! BLIV BEDRE TIL DISSE!

Discuss Identification and type checking

Discuss scopes/block structure and implication for implementation of identification tables/symbol tables

Discuss Implementation of semantic analysis

Question 6 - Run-time organization 3-5, 6-8,

Data representation (direct vs. indirect)

Slide 3-5

Direct representations are often preferable for efficiency:

- More efficient access (no need to follow pointers)
- More efficient “storage class” (e.g stack rather than heap allocation)

For types with widely varying size of representation it is almost a must to use indirect representation (see previous slide)

Storage allocation strategies: static vs. stack dynamic

Slides 6-8, 10

12-14 bonus (15-16)

Activation records (sometimes called frames)

Routines and Parameter passing

19-20

Question 7 - Heap allocation and Garbage Collection

Slide 3

Why may we need heap allocation?

Slide 5

Because we need something additional to the stack, for dynamic allocation. If we allocate something dynamic on the stack and then later on expend the dynamic thing, that might not be possible since the stack is constantly changing.

Garbage collection strategies (Types of GCs)

Slide 6, 10-16

Question 8 - Code Generation

Describe the purpose of the code generator

Slide 3.

Last section. Generate code in the target language.

Discuss Intermediate representations

Slide 4-6. !!

Used for generating code for multiple platforms, so only the very last step is machine dependant.

https://www.geeksforgeeks.org/intermediate-code-generation-in-compiler-design/

Describe issues in code generation

Slide 7

!!!!!!!!!!!!!

Code templates and implementations

Slide 8-10

Back patching

Slide 11

Used for instance when a jump/goto command is used to a place that is not yet know.

Implementation of functions/procedures/methods

????????????????????

Register Allocation and Code Scheduling

Slide 14-16 !! BLIV BEDRE TIL DISSE!