End Semester Examination

Name: Ro Mohnish Devaraj

Reg No: 39110636

Roll No: 195115398

Date: 14 May 2022

Subject Name: Compiler Design

Subject Code: scsA1604

No. of pages: 6

360415775792028

(31) Discuss the phases of a compiler indicating the inputs and outputs of each phase in translating the statement "amount = principle rate * 36.0".

Ans:

Phase1: Lexical Analysis

Lexical Analysis is the first phase when ~~computer~~ compiler scans the source code. This process can be left to right, character by character, and group these characters into tokens.

The primary functions are:

-> Identify the lexical unit in a source code

-> Identify token which is not a part of the language

Eg: amount = principle rate * 36.0 --> Token

amount -> identifier

principle rate -> identifier

36.0 -> Number

* -> Multiplication operator

Phase 2: Syntax Analysis

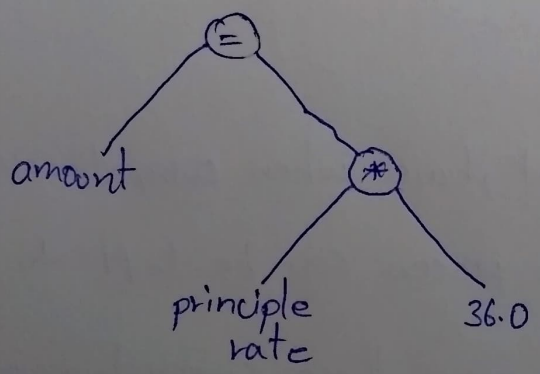
Syntax Analysis is all about discovering structure in code. It determines whether or not a text follows the expected format. The main aim of this phase is to make sure that the source code was written by the programmers is correct or not.

tasks performed:

→ Obtain tokens from the lexical analyzer

→ Checks if the expression is syntactically corred or not.

Eg: amount = principle rate * 36.0



Phase 3: Semantic Analysis

Semantic analysis checks the semantic consistency of the code. It uses the syntax tree of the prerous phase along with the symbol table to verify that the given source code is semantically consistent.

tasks performed:

→ Allows you to perform type checking

→ Checks if the source language permits the operands or not.

Eg: float x = 36.0;
float amount = principle rate * x;

Phase 4: Intermediate Code Generation

Once the semantic analysis phase is over the compiler, generates the intermediate code for the target machine. It represents a program for some abstract machine

tasks performed:

-> Holds the values computed during the process of translation

-> Allows you to maintain precedence ordering of the source language.

Eg: amount = principle rate * 36.0

t1 := int_to_float(36.0)

t2 := principle rate * t1

total := t2

Phase 5: Code Optimization

The next phase of is Code optimization or intermediate code. This phase removes unnecessary code lines and arranges the sequence of statements to speed up the execution of the program without wasting resources. the main goal of this phase is to improve on the intermediate code to generate a code that runs faster and occupies less space.

tasks performed:

-> Improves the running time of the target program

-> Removing statements which are not altered from the loop

Eg: a = ~~intof~~ intofloat(36.0)
b = c * a
d = b

→ b = c * 36.0
d = c * 36.0

Phase 6: Code Generation

Code Generation is the last and final phase of a compiler. It gets inputs from Code optimization phases and produces the page code or object code as a result. the objective of this phase is to allocate storage and generate relocatable machine code.

Eg: amount = principle rate * 36.0

```
MOVF  amount, R1
MULF  #36.0, R2
MULF  R1, R2
```

All the memory locations and registers are also selected and alloted during this phase. The code generated by this phase is executed to take inputs and generate expected outputs.

(34) Criticize on the issue that arise during the code generation phase.

Ans.

Code generator converts the intermediate representation of source code into a form that can be readily executed by the machine. A code generator is expected to generate the correct code.. Designing of code generator should be done in such a way so that it can be easily implemented, tested and maintained.

The following issues arises during the code generation phase:

1. Input to code generator –

The input to code generator is the intermediate code generated by the front-end, along with information in the symbol table that determine the run-time addresses of the data-objects denoted by the names in the intermediate representation.

2. Target program –

The target program is the output of code generator. The output may be absolute machine language, relocatable machine language, assembly language.

3. Memory Management

Mapping the names in the source program to the addresses of data objects is done by the front end and the code generator. A name in the three address statements refers to the symbol table entry for name

4. Instruction selection

Selecting the best instructions will improve the efficency of the program. It includes that should be complete and uniform. Instruction speeds and machine idioms also plays a major role when efficiency is considered.

5. Register allocation issues

Use of register make the computations faster in comparison to that of memory, so efficient utilization of registers is important.

6. Evaluation order

The code generator decides the order in which the instruction will be executed. The order of computational effects the efficency of the target code. Among many computational orders, some will require only fewer registers to hold the intermediate results, However, picking the best order in the general case is difficult NP-complete problem.