Assignment 5 - Code Generation

Due Mar 6 at 11:59pm **Points** 13 **Questions** 5
Available Feb 20 at 8pm - Mar 6 at 11:59pm **Time Limit** None
Allowed Attempts Unlimited

Take the Quiz Again

Attempt History

	Attempt	Time	Score
LATEST	Attempt 1	76 minutes	3 out of 13 *

* Some questions not yet graded

Score for this attempt: **3** out of 13 *
Submitted Feb 27 at 8pm
This attempt took 76 minutes.

Question 1 — 1 / 1 pts

In a compiler, what is the data structure that is passed from the parser to the code generator?

Correct!

parse tree

Correct Answers

parse tree

Question 2 — 1 / 1 pts

A single-pass code generator is able to build a symbol table if:

Correct!

- (●) The language requires all variables to be declared before use
- ( ) The target instruction set supports offset addressing
- ( ) The language does not support complex data types
- ( ) The language does not allow "local" variables

Question 3

1 / 1 pts

Storing local data on function calls - and supporting recursive calls - is typically done with a _____ data structure.

Correct!

stack

orrect Answers

stack

Question 4

Not yet graded / 4 pts

Give TWO reasons why the code generator may have to generate symbols that do not appear in the source program.

Your Answer:

The code generator may also have to create new symbols that don't appear in the source program because of the following two reasons.

1. Symbols pointing into complex data structures and allowing them to be referenced

2. Symbols as jump targets

Brief Explanation:

1. *Symbols pointing into complex data structures and allowing them to be referenced.*

The code generator may need to generate symbols that are not present in the source program to allow for the referencing of complex data structures such as arrays, structures, and classes. These data structures often require additional symbols to represent their internal layout and facilitate efficient referencing in the target code. For instance, an array in the source program may be represented by a single variable name, but in the target code, it may require additional symbols to represent the memory layout of the array elements and indexing operations.

The additional symbols generated by the code generator allow the target code to access and manipulate the complex data structures more efficiently without needing to perform costly runtime calculations or reference the original source program. The generated symbols are essential to represent the underlying data structure of the program in a way that is compatible with the target machine and the programming language requirements.

Example:

struct product { int weight; double price; } apple;

- apple_weight

- apple_price

2. *Symbols as jump targets*

The code generator may have to generate symbols that are not present in the source program to use them as jump targets in the target code. Jump targets are locations in the code where control flow can be transferred to via jump or branch instructions. In the source program, these jump targets may be represented by labels associated with specific statements or blocks of code. However, in the target code, these labels may not be sufficient to represent the jump targets due to differences in the code layout or the instruction set of the target machine.

To enable efficient control flow in the target code, the code generator may need to generate symbols for jump targets that are represented by relative or absolute addresses, offsets, or other machine instructions. These generated symbols ensure correct control flow and efficient execution of the target code. Moreover, generating symbols as jump

targets is also essential for effective debugging and profiling of the target code.

The generated symbols allow debugging tools to associate the target code with the original source program and provide useful information to the user, such as source code line numbers, variable names, and call stacks. They can also be used to collect performance data during program execution, such as profiling information and code coverage statistics. Overall, generating symbols as jump targets is a crucial aspect of the code generator's task to create efficient and correct target code and facilitate effective debugging and profiling of the program.

Example:

```
if ( var1 == var2)
{
  some code;
}

  LDA    var1
  COMP   var2
  JLT   generated_label_1
  JGT   generated_label_1
  some code
generated_label_1  next instruction
```

Question 5

Not yet graded / 6 pts

Once the parser has generated its internal data structure, representing the program, give three things that can be done based on that structure.

Your Answer:

Once the parser has generated its internal data structure, representing the program, the three things that can be done based on that structure are:

1. Code Generator -> Assembly Language Source File

2. Code Generator -> Abstract (P-code) Language Source File

3. Interpreter

1. *Code Generator -> Assembly Language Source File*

The original – and still the primary – purpose of a high-level language compiler is to convert the high-level code to assembly language for a particular machine.

The parser generates an internal data structure representing the program, which can be utilized by the code generator to create an assembly language source file. Assembly language is a low-level programming language that interacts directly with a computer system's hardware, such as device drivers, operating systems, and system utilities. The internal data structure provides complete information about the program, including syntax, semantics, and structure, which enables the code generator to convert the program into assembly language.

The code generator analyzes the program's structure, generates assembly code, and creates labels and symbols to represent program entities such as functions, variables, and constants. The resulting assembly language source file is a text file that can be assembled using an assembler program to produce machine code that the computer's processor can execute directly.

2. *Code Generator -> Abstract (P-code) Language Source File*

After generating its internal data structure, the parser can be used by the code generator to create an abstract (P-code) language source file, which is an intermediate code designed to be executed by a virtual machine. The parser's internal data structure provides a complete representation of the program, including its syntax, semantics, and structure. This information can be used by the code generator to translate the program into P-code, which can then be executed by a P-code interpreter.

The code generator analyzes the program's structure, generates P-code for each statement and expression, and creates labels and symbols to represent program entities such as functions, variables, and constants. The resulting P-code program is a text file that can be executed by a P-code interpreter, which emulates the functionality of a computer system.

3. *Interpreter*

The internal data structure generated by the parser can be used by an interpreter to execute the program. An interpreter reads and executes the source program instructions one at a time, taking the internal data

structure generated by the parser and executing it directly without further translation or code generation.

The internal data structure contains all the necessary information about the program, including its syntax, semantics, and structure. The interpreter uses this information to execute the program in a step-by-step manner, following the control flow and handling each instruction as it is encountered.

The interpreter reads the program instructions from the source file, converts them to an internal representation, and uses the internal data structure to perform the necessary computations and operations required by the program.

Quiz Score: **3** out of 13