AKTU COMPILER DESIGN 2017-18 SOLUTIONS

1. a) what is translator?

Answer : A translator or programming language processor is a compiler for converting a program written in the source language into a program in a different programming language (the target language) that is functionally equivalent. This is without losing the functional or logical structure of the original program. These include translations between high-level and human-readable computer languages such as C++ and Java, intermediate-level languages such as Java bytecode, low-level languages such as the assembly language and machine code, and between similar levels of language on different computing platforms, as well as from any of these to any other of these. The term is also used for translators between software implementations and hardware/ASIC microchip implementations of the same program, and from software descriptions of a microchip to the logic gates needed to build it.

b) Differentiate between compiler and assembler.

Answer : COMPILER

The compiler works on a basic principle. It simply reads the whole program written on it. Then it converts the whole program to the machine/computer language. For this process, the compiler takes a lot of time to read the whole program or to analyze the whole program. The compilers are generally the memory eaters. They need a lot of memory to complete their process. Because they create the object code by analyzing the program written on them. They work fast as compared to the interpreters. Because they have the very less execution time. In this process, the whole program doesn't require to be created/compiled every time. Languages like to C and the C++ commonly used the compilers.

ASSEMBLER

Assembler is also a compiler. But in the assembler, the source code is written in the Assembly language. This assembly language is a simple language. And this language is understandable by the humans as well. The work of an assembler is to convert the assembly language to the machine language. The assembler works on the principle of the one to one mapping translation. There are improvements coming in the assemblers day by day. They are becoming very easy and user-friendly programs to use as well. The latest editions of the assemblers work very efficiently and they help us in the debugging of the written code as well.

DIFFERENCE BETWEEN COMPILER AND ASSEMBLER

The compiler is a simple program which converts the source code written by the humans to a machine language. While the assembler has a little bit different work, it converts the assembly language to the machine language.

Compilers work more directly than the assemblers. The compilers can convert the human written code in the machine language directly. But the assembler can't do this at once. It converts a source code to an object code first then it converts the object code to the machine language with the help of the linker programs.

c) Discuss conversion of NFA into a DFA also give the algorithm used in this conversion.

Answer Problem Statement

Let X = (Qx, ∑, δx, q0, Fx) be an NDFA which accepts the language L(X). We have to design an equivalent DFA Y = (Qy, ∑, δy, q0, Fy) such that L(Y) = L(X). The following procedure converts the NDFA to its equivalent DFA −

Algorithm

Input − An NDFA

Output − An equivalent DFA

Step 1 − Create state table from the given NDFA.

Step 2 − Create a blank state table under possible input alphabets for the equivalent DFA.

Step 3 − Mark the start state of the DFA by q0 (Same as the NDFA).

Step 4 − Find out the combination of States {Q0, Q1,... , Qn} for each possible input alphabet.

Step 5 − Each time we generate a new DFA state under the input alphabet columns, we have to apply step 4 again, otherwise go to step 6.

Step 6 − The states which contain any of the final states of the NDFA are the final states of the equivalent DFA.

d) Write down the short note on symbol table.

Answer:

Symbol Table is an important data structure created and maintained by the compiler in order to keep track of semantics of variable i.e. it stores information about scope and binding information about names, information about instances of various entities such as variable and function names, classes, objects, etc.

It is built in lexical and syntax analysis phases.

The information is collected by the analysis phases of compiler and is used by synthesis phases of compiler to generate code.

It is used by compiler to achieve compile time efficiency.

It is used by various phases of compiler.

e) Describe Data structure for symbol table.

Following are commonly used data structure for implementing symbol table :-

List –In this method, an array is used to store names and associated information.

A pointer "available" is maintained at end of all stored records and new names are added in the order as they arrive To search for a name we start from beginning of list till available pointer and if not found we get an error "use of undeclared name" While inserting a new name we must ensure that it is not already present otherwise error occurs i.e. "Multiple defined name"

Insertion is fast O(1), but lookup is slow for large tables – O(n) on average

Advantage is that it takes minimum amount of space.

Linked List – This implementation is using linked list. A link field is added to each record.

Searching of names is done in order pointed by link of link field.

A pointer "First" is maintained to point to first record of symbol table.

Insertion is fast O(1), but lookup is slow for large tables – O(n) on average

Hash Table –

In hashing scheme two tables are maintained – a hash table and symbol table and is the most commonly used method to implement symbol tables..

A hash table is an array with index range: 0 to tablesize – 1.These entries are pointer pointing to names of symbol table.

To search for a name we use hash function that will result in any integer between 0 to tablesize – 1.

Insertion and lookup can be made very fast – O(1).

Advantage is quick search is possible and disadvantage is that hashing is complicated to implement.

Binary Search Tree –

Another approach to implement symbol table is to use binary search tree i.e. we add two link fields i.e. left and right child.

All names are created as child of root node that always follow the property of binary search tree.

Insertion and lookup are O(log2 n) on average.

f) What is mean by Activation record.

Answer: Control stack or runtime stack is used to keep track of the live procedure activations i.e the procedures whose execution have not been completed. A procedure name is pushed on to the stack when it is called (activation begins) and it is popped when it returns (activation ends). Information needed by a single execution of a procedure is managed using an activation record or frame. When a procedure is called, an activation record is pushed into the stack and as soon as the control returns to the caller function the activation record is popped.

A general activation record consist of the following things:

Local variables: hold the data that is local to the execution of the procedure.

Temporary values: stores the values that arise in the evaluation of an expression.

Machine status: holds the information about status of machine just before the function call.

Access link (optional): refers to non-local data held in other activation records.

Control link (optional): points to activation record of caller.

Return value: used by the called procedure to return a value to calling procedure

Actual parameters

g) What is postfix notations?

Answer:

Postfix Notation

Postfix notation is the useful form of intermediate code if the given language is expressions.

Postfix notation is also called as 'suffix notation' and 'reverse polish'.

Postfix notation is a linear representation of a syntax tree.

In the postfix notation, any expression can be written unambiguously without parentheses.

The ordinary (infix) way of writing the sum of x and y is with operator in the middle: x * y. But in the postfix notation, we place the operator at the right end as xy *.

In postfix notation, the operator follows the operand.

h) Define Three address Code

Answer:

In computer science, three-address code[1] (often abbreviated to TAC or 3AC) is an intermediate code used by optimizing compilers to aid in the implementation of code-improving transformations. Each TAC instruction has at most three operands and is typically a combination of assignment and a binary operator. For example, t1 := t2 + t3. The name derives from the use of three operands in these statements even though instructions with fewer operands may occur.

Since three-address code is used as an intermediate language within compilers, the operands will most likely not be concrete memory addresses or processor registers, but rather symbolic addresses that will be translated into actual addresses during register allocation. It is also not uncommon that operand names are numbered sequentially since three-address code is typically generated by the compiler.

A refinement of three-address code is A-normal form (ANF).

i) What are Quadruples.

Answer

Quadruples

Each instruction in quadruples presentation is divided into four fields: operator, arg1, arg2, and result. The equation r1 = c * d; is represented below in quadruples format:

Op	arg1	arg2	result
*	c	d	r1

j) what do you mean by regular expression?

Answer

A regular expression (sometimes called a rational expression) is a sequence of characters that define a search pattern, mainly for use in pattern matching with strings, or string matching, i.e. “find and replace”-like operations.

Regular expressions are a generalized way to match patterns with sequences of characters. It is used in every programming language like C++, Java and Python.