ANATOMY OF THE COMPILER - SCHEDULED FOR 11.12.07 4TH

How does a compiler work?

Compiler works in two stages. Two main stages are

1. Analysis stage – Front end of the compiler
2. Synthesis stage – Back end of the compiler

Analysis stage – Front end

Breaks the source program into pieces and creates (language independent) intermediate representation of the program

Synthesis stage – Back end

Desired target program from intermediate representation

Analysis stage – Front end

It has 4 phases

Lexical Analysis phase:

Stream of characters making up a source program is read from left to right and grouped into tokens which are sequence of characters that have collective meaning.

Ex: identifiers, reserved words, special characters, kinds of operators.

Example:
int a;
a=a+2;
Scanning the above code the scanner will return

int - reserved word
a - identifier
; - special character
a - identifier
= - assignment operator
a - identifier
+ - arithmetic operator
2 - integer constant
; - special character

Syntax analysis stage

Tokens found during scanning are grouped together using context free grammar. A grammar is ser of rules that define valid structures in programming language. Each token is associated with a specific rule and grouped together accordingly. This process is parsing. Output of this phase is parse tree or derivation.

Example:

E-> E + E |E - E |E * E |E / E |V |C
V-> id
C-> integer const |float const

To parse a+2 apply the following rule

E-> E + E
-> V + E
-> id + E
-> id + C
-> id + integer const

When we reach a point in the parse, where we have only tokens, we have finished .

Semantic analysis phase

Statement can be syntactically correct but disobeys the semantic rules of source language.

Ex:
Undeclared variable
Function called with improper arguments
Access violation
Incompatible operands
Type mismatches

Example:

```
int arr[2], c;
c=arr*10;
```

In semantic analysis phase, compiler checks the types and reports that you cannot use an array variable in a multiplication expression.

Intermediate code generation

Intermediate code is generated because it is easy to generate and translate and it is generated using three address code.

Example:
a=b*c+b*d
t1=b*c
t2=b*d
t3=t1+t2
a=t3

Synthesis stage – Back end

It has 3 phases

Intermediate code optimization

Various techniques are applied to produce the smallest, fastest and most efficient code. Techniques used are given below.

- Inhibiting code generation of unreachable code
- Getting rid of unused variable
- Elimination of multiplication by 1 and addition by 0.
- Loop optimization
- Common sub expression elimination
- Strength reduction

Example:
t1=b*c
t2=t1+0
t3=b*c
t4=t2+t3
a=t4

The above code will be optimized to

t1=b*c
t2=t1+t1
a=t2

Object code generation

Three address code is translated into sequence of assembly or machine language instructions that perform the ame task.

Object code optimization

Object code is transformed into tighter and more efficient object code. Compiler can take the advantage of machine specific idioms (special instructions such as pipelining, branch prediction etc.)