Solved by:	**Sahar (well wisher)**	**Class**	**BSCS 6th Semester**
Subject	CS606 (COMPILER CONSTRUCTION)	**Solution Type:**	Final Term Solved Subjective including Papers of Year : 2013,2012,2011,2010,2009...2006
Institute:	**Virtual University of Pakistan**		

Final Term Papers 2013

1)

What is live variable?

ANSWER: **Page no: 128**

A variable is live at a given point in time if it has a next use. Liveness tells us whether we care about the value held by a variable. Here is the algorithm for computing live status of variables in a basic block".

2)

What are the common sub-expressions?

Answer: **Page no: 135**

A node in the DAG with more than one parent is common sub-expression

3)

What is Relocation?

Answer:

Compilers and assemblers generate the object code for each input module with a starting address of zero. Relocation is the process of assigning load addresses to different parts of the program by merging all sections of the same type into one section. The code and data section also are adjusted so they point to the correct runtime addresses.

4)

Third section of YACC represents what kind of information? 3 Marks

Answer: **Page no : 88**

The third section represents C/C++ functions.

5)

Differentiate attribute grammar and syntax-directed grammar.

Answer: Page no: 100

Attribute grammar:

A CFG is augmented with a set of rules. Each symbol in the derivation has a set of values or attributes. Rules specify how to compute a value for each attribute

Syntax-directed grammar:

A syntax directed definition is a generalization of a context free grammar in which each grammar symbol has an associated set of attributes, partitioned into two subsets called the synthesized and inherited attributes of that grammar symbol.

6)

Brief note on Reducible Flow Graphs?

Answer:

7)

What kind of data structure is used to represent Basic Blocks?

Answer:

A basic block is a sequence of consecutive statements with single entry/single exit. Flow of control only enters at the beginning and only leaves at the end. The can be variants of basic blocks with single entry/multiple exit, multiple entry/single exit.

Page no 124

8)

Discuss the issue of Target programs in Code Generation phase?

Answer:

Memory management: mapping names to data objects in the run-time system.

- Instruction selection: the assembly language instructions to choose to encode intermediate code statements

• Instruction scheduling: instruction chosen must utilize the CPU resources effectively. Hardware stalls must be avoided.

• Register allocation: operands are placed in registers before executing machine operation such as ADD, MULTIPLY etc. Most processors have a limited set of registers available. The code generator has to make efficient use of this limited resource

Page no : 121

9)

How register descriptor are used in code generation? 5 Marks

Answer:

The code generator uses two data structures for keeping track of register usage one is Register descriptor has register status (empty, in use) and contents (one or more "values")

Page no : 129

10)

Diff. Between Hop croft's Algorithm and Subset Construction Algorithm?

Answer:

Hop croft's Algorithm:

The Hop croft's algorithm can be used to minimize DFA states. The behind the algorithm is to find groups of equivalent states. All transitions from states in one group G1 go to states in the same group G2. We construct the minimized DFA such that there is one state for each group of states from the initial DFA.

Subset Construction Algorithm:

The algorithm is called subset construction. In the transition table of an NFA, each entry is a set of states. In DFA, each entry is a single state. The general idea behind NFA-to-DFA construction is that each DFA state corresponds to a set of NFA states. The DFA uses its state to keep track of all possible states the NFA can be in after reading each input symbol.

11)

Role of Symbol Table

Answer:

Typically, the compiler augments the IR with a set of tables that record additional information. Foremost among these is the symbol table. These tables are considered part of the IR.

Page no: 99

12)

What is LR(1) parsing and discuss main advantage?

Answer:

The LR(1) parsers can recognize precisely those languages in which one-symbol lookahead suffices to determine whether to shift or reduce. The LR(1) construction algorithm builds a handle-recognizing DFA.

Page no : 71

13)

How inherited attribute are different synthesized with respect to grammar attribute?

Answer:

Synthesized attributes:

Values used to compute synthesized attributes flow bottom-up in the parse tree.

Inherited attributes:

Values flow top-down and laterally in the parse tree.

Page no : 92

14)

From given CFG generate string aa+a*

S->SS+ | SS* | a

Answer:

S→SS+

→ S→SS*

→ S→SSS*+ Replace last S with >> S→SS* it become a form of post order traversal.

→S→SS+S* Put symbols back using in order traversal

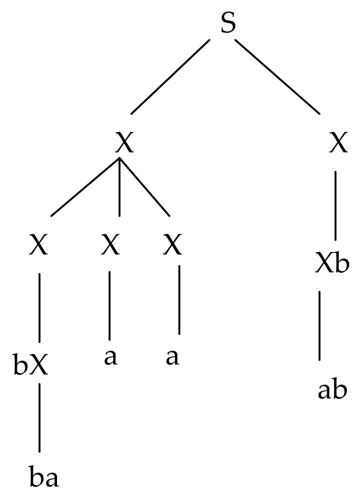
➔S→aa+a*   Replace all S on R.h.S with a and finally we get the string!

15)

S->XX

X->XXX | bX | Xb | a

Draw parse tree for string bbaaaab.



Answer:

S→XX

S→XXb   placing last X with Xb

S→XXXXb   placing First X with XXX

S→bXXXXb   placing first X with bX

S→bbXXXXb   placing second X with bX

S→bbaaaab

16)

Example of Syntax directed and translates it?

while a < b

if c < d then

x = y + z

else

x = y – z

Answer:

The Syntax translation scheme will generate the following code:

```
L1:  if a < b goto L2
 goto Lnext
L2:  if c < d goto L3
 goto L4
L3:  t1 = y + z
  x = t1
 goto L1
L4:  t2 = y – z
  x = t2
 goto L1
Lnext: nop
```

Page no : 110

17)

Why compilers use multiple IR?

Answer:

Compilers are organized as a series of passes. This creates the need for an intermediate representation (IR) for the code being compiled. Compilers use some internal form– an IR –to represent the code being analyzed and translated. Many compilers use more than one IR during the course of compilation. The IR must be expressive enough to record all of the useful facts that might be passed between passes of the compiler. During translation, the compiler derives facts that have no representation in the source code

Page no : 99

18)

S---------->AaBb

E --------> e

E --------> f

Is that LL (1) grammar why or why not explain your answer in one or 2 lines?

Answer:

A context-free grammar whose Predict sets are always disjoint (for the same non-terminal) is said

to be LL(1).

As E-----> e and E---- > f   the predict set of E is disjoint so it is a LL(1) grammar.

19)

What is reducible flow graph?

Answer:

Reducible flow graph

Intuition: Produced by exclusive use of flow of control statements such as if-then-else, while-do, continue, break etc.

Statistics: Even programs written using goto statements are almost always reducible.

Main property: there are no jumps in the middle of loops from outside; the only entry to a loop is through its header.

Definition: A flow graph G is reducible if and only if we can partition the edges into two disjoint groups, often called the forward edges and back edges with properties:- forward edges from an acyclic graph in which every node can be reached from the initial node of G.- the back edges consist only of edges whose heads dominate their tails.

20)

:   Consider grammar SàaS show that grammar is ambiguous by using two parse tree for the same string.

Answer:

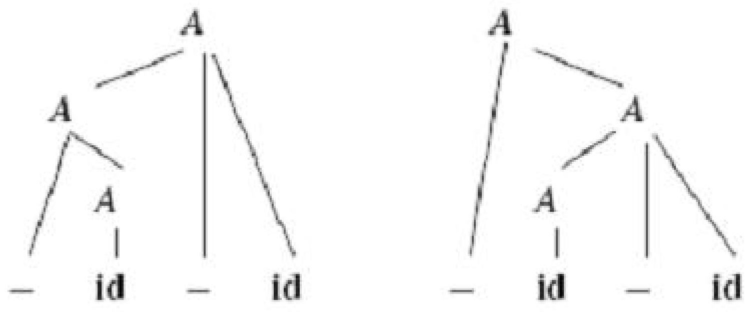
The string given is not correct(incomplete) actually so unable to make parse tree. So to understand the concept of ambgious grammer we can look at following example:

Show that the grammar

$$A \rightarrow -A$$
$$A \rightarrow A - \mathbf{id}$$
$$A \rightarrow \mathbf{id}$$

is ambiguous by finding a string that has two different syntax trees.

Solution:



PAGE NO : 40

21)

In two pass assembler what is the objective of First Pass?

Answer:

The front end recognizes legal and illegal programs presented to it. When it encounters errors, it attempts to report errors in a useful way. For legal programs, front end produces IR and preliminary storage map for the various data structures declared in the program.

Page no : 5

22)

Brief note on Parser Generator

Answer:

The earliest and still most common form of compiler-compiler is a parser generator, whose input is a grammar (usually in BNF) of a programming language, and whose generated output is the source code of a parser often used as a component of a compiler. Similarly, code generator-generators such as (Burg) exist, but such tools have not yet reached maturity.

Page no : 88

23)

What is mean by Semantic Actions?

Answer:

A compiler has to do more than just recognize if a sequence of characters forms a valid sentence in the language. It must do something useful with the parsed sentence

The semantic actions of a parser perform useful operations.

- Build an abstract parse tree.
- Type checking.
- Evaluation in the case of an interpreter.
- Begin the translation process in the case of a compiler.

In a recursive-descent parser the semantic actions are mixed in with the control flow of the parsing.

In some compiler constructors (such as JavaCC and Yacc) the semantic actions are attached to the production rules.

24)

What is mean by Intermediate Code? **3 Marks Questions**

Answer:

The intermediate code generation phase transforms parse tree into an intermediate-language representation of the source program.

Intermediate codes are machine independent codes, but they are close to machine instructions.

The given program in a source language is converted to an equivalent program in an intermediate language by the intermediate code generator.

Intermediate language can be many different languages, and the designer of the compiler decides this intermediate language.

- syntax trees can be used as an intermediate language.
- postfix notation can be used as an intermediate language.
- three-address code (Quadraples) can be used as an intermediate language

1. we will use quadraples to discuss intermediate code generation
2. quadraples are close to machine instructions, but they are not actual machine instructions.

- some programming languages have well defined intermediate languages.

1. java - java virtual machine
2. prolog - warren abstract machine
3. In fact, there are byte-code emulators to execute instructions in these intermediate languages.

Read more: http://wiki.answers.com/Q/What_is_intermediate_code_generation#ixzz1vQA2ygJM

2nd answer:

Intermediate code is the register and Processor memory code given by the Programmer about the arrays passed by the PC.

25)

What is Relocation?

Answer:

Compilers and assemblers generate the object code for each input module with a starting address of zero. Relocation is the process of assigning load addresses to different parts of the program by merging all sections of the same type into one section. The code and data section also are adjusted so they point to the correct runtime addresses.

26)

What are the components of CFG?

Answer:

A CFG consists of the following components:

- 1. S is the start symbol
- 2. N is a set of non-terminal
- 3. T is a set of terminals
- 4. P is a set of productions

Page no : 35.

27)

How Lex different from Flex? (2marks)

Answer:

Flex : generates lexical analyzer in C or C++. It is more modern version of the original Lex tool that was part of the AT&T Bell Labs version of Unix.

Jlex: written in Java. Generates lexical analyzer in Java

Page no : 26

28)

What are two properties to reducible flow graph? (2 marks)

Answer:

Reducible flow graph

Intuition: Produced by exclusive use of flow of control statements such as if-then-else, while-do, continue, break etc.

Statistics: Even programs written using goto statements are almost always reducible.

Main property: there are no jumps in the middle of loops from outside; the only entry to a loop is through its header.

Definition: A flow graph G is reducible if and only if we can partition the edges into two disjoint groups, often called the forward edges and back edges with properties:- forward edges from an acyclic graph in which every node can be reached from the initial node of G.- the back edges consist only of edges whose heads dominate their tails.

29)

What type of information stored in Lex/Flex input file? (2 marks)

Answer:

The input specification file consists of three sections:

C or C++ and flex definitions

%%

token definitions and actions

%%

user code

Page no : 26

30)

Write regular expression for the language of all words that starts and ends with different letters? (3marks)

Answer:

a(a+b)*b+b(a+b)*a

31)

When we use garbage collection with respect to memory management? (3marks)

Answer:

In computer science, garbage collection (GC) is a form of automatic memory management. The garbage collector, or just collector, attempts to reclaim garbage, or memory occupied by objects that are no longer in use by the program.

Ref : http://en.wikipedia.org/wiki/Garbage_collection_(computer_science)

32)

How we can implement Syntax directed translator using two pass?

Answer:

The easiest way to implement syntax-directed definitions is to use two passes: construct a syntax tree for the input in the first pass and then walk the tree in depth-first order evaluating attributes and emitting code.

Page no : 111

33)

For each instruction, show which variable are live variable are live imidiately after instruction execution (5marks)

1. A=7

2. B=A+2

3. C=A+B

4.D=C+B

5.B=C+B

Answer:

Live {A}

A=7

Live {A}

B=A+2

Live{A,B}

C={A+B}

Live{C,B}

D=C+B

Live={C,B}

B={C+B}

Live={ A,B,C}

Page no : 129

34).

Add semantic rules to the following grammar to compute the attribute rm, whose value is the rightmost terminal in the string we parsed. For example, if the string parsed were zxyxy, S.rm would be y. (5marks)

```
S  → A             {S.rm =

A  → A1 x y
      | B A1 y
      | C

B  → B1 z
      | x

C  → w C1
      | y C1
      | z
```

Answer:

S.nptr = A.nptr

S.nptr = mknode (‘rm’, y.val)

A.nptr = mknode(‘∗’, A1.nptr, x.val, y.val)

A.nptr = mknode(‘∗’, B.nptr, A1.val, y.val)

B.nptr = mknode(‘∗’, B1.nptr, z.val)

B.nptr = mkleaf(‘*’, x.val)

I am not sure about this solution.

35).

How many place holder positions we can use if production has ‘’K’’ symbol at right hand side at the shift reduce parsing?

Answer:

Suppose

S--- > ▶ K

S--- > K ▶

In my view two Place holder positions can be used.

Page no : 65

Final Term Papers 2012

36)

What are the terminals, non-terminals, and the start symbol for the grammar? [02 Points]

S → a B C d | d C B e

B → b B | ε

C → c a | a c | ε

Answer:

Terminals: a, c, e, d

Non Terminal: S, B, C

Start symbol: S

37)

S → a B C d | d C B e

B → b B | ε

C → c a | a c | ε

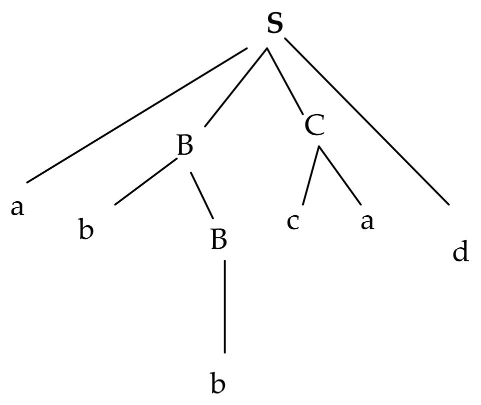
Give the parse tree for the input string abbcad. **[02 Points]**

Answer:

S → a B C d

S---> abBCd

S--- > abbcad



38) Compute the First sets and Follow sets for each of the non terminals in the grammar.

S → a B C d | d C B e

B → b B | ε

C → c a | a c | ε

Answer:

First {S}= {d, e, a, c, ε }

First{B}={ ε }

First{C}={a, c, ε }

Rules for making first set:

1) If S→ A….Z then first of S will be First of A….Z
2) If First of A….Z all contains ε then first of S will also contain ε
3) If any (single) First set from A….Z $_{\text{doesn't}}$ contain ε in their First set then First of S will also doesn't contain ε.
4) The first of A will be first non terminal on r.h.s
5) The first of B will also be the non terminal on r.h.s
6) Note: ε is also a terminal.

Follow set :

$S \rightarrow a\ B\ C\ d \mid d\ C\ B\ e$

$B \rightarrow b\ B \mid \varepsilon$

$C \rightarrow c\ a \mid a\ c \mid \varepsilon$

Follow{S}--- > {a,d,$}
Follow{B}---- > {a, d, $ }
Follow{C}---- >{a, c, $ }

Rules for making follow set:

1. First put $ (the end of input marker) in Follow(S) (S is the start symbol)
2. If there is a production A → aBb, (where a can be a whole string) **then** everything in FIRST(B) except for ε is placed in FOLLOW(B).S
3. If there is a production A → aB, **then** everything in FOLLOW(A) is in FOLLOW(B)
4. If there is a production A → aBb, where FIRST(b) contains ε, **then** everything in FOLLOW(A) is in FOLLOW(B)

39) Construct an LL (1) parsing table for the grammar

Rules for making LL(1) table:

For each terminal a in FIRST(a), add A---- > a to M[A,a].

If epslon is in FIRST(a), add A----> a to M[A,b] for each terminal b in FOLLOW(A).

If epslon is in FIRST(a), and $ is in FOLLOW(A), add A ---- > a to M[A,$].

$\mathbf{S \rightarrow a\ B\ C\ d \mid d\ C\ B\ e}$

$\mathbf{B \rightarrow b\ B \mid \varepsilon}$

$\mathbf{C \rightarrow c\ a \mid a\ c \mid \varepsilon}$

	a	b	c	d	e		*	(	)	$
S	S--- > a B C d				S→dCBe		S→dCBe S--- > a B C d			
B		B→bB							B→bB	B→bB
C	C → c a C → ac		C → ac				C → c a C → ac			C → c a C → ac

Table is incomplete.

40)

What is the role of Automatic Code Generator with respect to compiler?

Answer:

It generates efficient tokens automatically. It is known as Lexer generator.

41)

How Linker play an important role with respects to compilers constructions?

Answer:

A **linker** or **link editor** is a computer program that takes one or more source files generated by a compiler and combines them into a single executable program.

42)

Consider the grammar for arithmetic expressions?

E→ id Q

Q→ ER| ε

R→ +Q | -Q| *Q| /Q

Show the follow set for all the non-terminal in the grammar

Follow{E}= ?

Solution:

Rules for making follow set:

5. First put $ (the end of input marker) in Follow(S) (S is the start symbol)
6. If there is a production A → aBb, (where a can be a whole string) **then** everything in FIRST(b) except for ε is placed in FOLLOW(B).
7. If there is a production A → aB, **then** everything in FOLLOW(A) is in FOLLOW(B)
8. If there is a production A → aBb, where FIRST(b) contains ε, **then** everything in FOLLOW(A) is in FOLLOW(B)

Here a and b is terminal while A,B are non terminals.

In our question

E→ id Q

Q→ ER| ε

Follow{Q}={+,-,*,/}

R→ +Q | -Q| *Q| /Q

Id, ε, + , -, * , / are **terminals**

E, Q, R are non terminals.

Follow{E}={+,-,*,/,$}

43)

Consider the following grammar. Calculate the first set of non- terminal S, A and B

S→ AB

A→ a|ε

B→ b|ε

Solution:

S→ AB

Rules for making first set:

1) If S→ A….Z then first of S will be First of A….Z
2) If First of A….Z all contains ε then first of S will also contain ε
3) If any (single) First set from A….Z doesn't contain ε in their First set then First of S will also doesn't contain ε.
4) The first of A will be first non terminal on r.h.s
5) The first of B will also be the non terminal on r.h.s
6) Note: ε is also a terminal.

Now make first sets for S,A and B

First{A}={a, ε }

First{B}={b ,ε}

44)

Consider the grammar

A → B C D

B → h B | ε

C → C g | g | C h | i

D → A B |ε

Answer:

Rules for making first set:

7) If S→ A….Z then first of S will be First of A….Z

8) If First of A….Z all contains ε then first of S will also contain ε
9) If any (single) First set from A….Z doesn't contain ε in their First set then First of S will also doesn't contain ε.
10) The first of A will be first non terminal on r.h.s
11) The first of B will also be the non terminal on r.h.s
12) Note: ε is also a terminal.

$A \rightarrow B\ C\ D$

$B \rightarrow h\ B \mid \varepsilon$

$C \rightarrow C\ g \mid g \mid C\ h \mid i$

$D \rightarrow A\ B \mid \varepsilon$

First {A}={h,g,i}
Note: As the production C didn't contain ε so we didn't add ε in First {A}.
First {B}={h, ε}
First{C}={g,i}
First{D}={ h,g,i, ε }
Note: **We added** ε there because D originally has ε as a terminal in it.

45)

How recursive descent parser is implemented using any Object Oriented language?

(3Marks)

Answer:

We can use C++ to code the recursive descent parser in an object oriented manner. We associate a C++ class with each non-terminal symbol. An instantiated object of a non-terminal class contains pointer to the parse tree.

46)

Write a regular expression for the language of all words that starts and ends with different letters.

(2 Marks)

Answer:

a(a+b)*b+b(a+b)*a