Exam #2
R. Eigenmann
Fall 2001

Name_____

EE573 OR EE468 _____

******* FORM B ***** FORM B ***** FORM B ***** FORM B ***** FORM B**

Mark all answers that apply. If you think a question is ambiguous, state your assumptions.

Advanced questions are bonus questions for EE468 students.

1. Grammar:

a- The production L → R1 | R2 is in standard BNF form
b- (a) is false, but it can be turned into exactly one production in standard BNF form.
c- In standard BNF a Nonterminal can appear on the left hand side of several productions
d- The following grammar can be parsed with an LL1 parser:
   L → L1 L2 L3
   L1 → t1 t2 t3
   L2 → t1 t4 t5
   L3 → t1 t2 t6
e- (d) is false but would be true if the last production were L4 → t1 t6 t7

2. Semantic records:

a- Semantic records help communicate information between scanners and parsers
b- Semantic records help communicate information between different semantic action procedures
c- The generated code is part of the semantic record information
d- The semantic record generated by the semantic action at the end of a production is always empty
e- A semantic action may generate an empty semantic record

3. Code generation for arrays and structs:

a- Address generation for arrays and structs involves a base address and an offset
b- For a two-dimensional array x[dim1,dim2], the offset of a reference x[i,j] has the form i*dim1 + j*dim2
c- In all major programming languages, two array references x[i,j] and x[i,j+1] access adjacent memory cells.
d- An efficient code generated for reading the array element x[3] would look something like this: load &x,R1; add 2,R1; load (R1),R2
e- Two references to the same array but with different indices always access different memory cells.

4. The following are possible tasks of a compiler:

a- Creating a syntax tree from Java source code
b- Register allocation and code scheduling
c- Loading a program into memory
d- Translate source code into modified source code
e- Detecting syntax errors in a program

5. Register save:

a- Registers that hold a live value of the calling subroutine must be saved during the subroutine call
b- The calling subroutine (as opposed to the callee) always saves the necessary registers
c- The register save mechanism may use a bit vector in which the callee describes the registers holding live values
d- If (c) is used, the called subroutine will save all registers described by the bitvector
e- A simple register save optimization is not to save any registers if the caller and callee use different registers.

6. Aliasing

a- Two variable names that refer to the same memory location are said to be aliased
b- Two pointers that can access the same memory location are said to be aliased
c- Aliasing is an optimization that improves the generated code
d- Common subexpression elimination has to be more conservative in the presence of aliases
e- Aliasing prevents all advanced compiler optimizations

7. Basic code generation and optimization:

a- In simple code generation, each tuple code is translated into exactly one assembly instruction
b- Simple code generation schemes are inefficient in terms of the size of the generated code and the number of memory references.
c- Register allocation is an important technique for reducing the number of memory accesses
d- Tuple codes may expand into several assembly instructions if there are enough registers
e- Classical compilers include optimization techniques that produce code with typically an order of magnitude higher performance than simple code generation schemes.

8. Symbol table:

a- Symbol tables contain all terminal and non-terminal symbols
b- Symbol tables may contain the value of an identifier
c- Binary trees are suitable implementation methods for symbol tables if all identifiers have equal length
d- Identifiers, types, and subroutine names can all be part of the symbol table information
e- At runtime, the symbol table contains the length of dynamically-sized variables

9. Parameter passing:

a- One can pass either variables or expressions to reference parameters
b- If a value parameter is modified in a subroutine, the change will not be visible after the end of the subroutine
c- For reference parameters, the calling subroutine pushes the address of the passed variable onto the stack.
d- Accessing a reference parameter inside a subroutine involves some form of indirect addressing.
e- "Dope vectors" are not visible to the programmer

10. The following compiler tasks can be part of a code generation pass:
a- Instruction selection
b- Address mode selection
c- Register allocation
d- Code scheduling
e- Code optimization

11. Syllabus. Grading policy:
a- Class participation can make the difference between an A and a B grade
b- Grading is 50% Tests, 10% Class participation and Exercises, 40% projects
c- If I disagree with a grade, it is sufficient to tell the TA or instructor after class.
d- The final project has equal weight to all other projects
e- Turning a project in late does not affect the grade if the machine was down

12. Activation record:
a- Activation records are always placed onto the stack, hence the name stackframe
b- The activation record contains subroutine parameters, return address, and the type of local variables.
c- If the stack grows upwards, the location of subroutine parameters on the stackframe is at lower addresses than the return address.
d- One use of the frame pointer is to cleanup the stack at the end of the subroutine
e- the framepointers of caller and callee subroutines form a pointer chain in memory

13. Common subexpression elimination, value numbering. Consider the following code.
s1: X = Y * Z + U
s2: K = Y * Z + U
s3: L = Y + Z + U
s4: U = T + Y * Z
s5: Y = Y * Z + U
s6: K = Y * Z + T
a- CSE eliminates redundant computation
b- Expressions Y*Z in statement s1 has the same value number as Y*Z in s2
c- Expression Y*Z+U in s1 has the same value number as Y*Z+U in s2
d- Expressions Y*Z in statement s5 has the same value number as Y*Z in s1
e- Expressions Y*Z in statement s6 has the same value number as Y*Z in s2

14. Semantic stack:
a- In LL parsing, the semantic stack grows in synch with the parse stack
b- At a reduction operation of an LR parser, the semantic stack shrinks by n+1 elements if the corresponding production has n right-hand side symbols
c- Semantic stack handling is easier to implement in LR parsing than in LL parsing
d- Semantic processing can be done without the use of a semantic stack
e- Usually, semantic routines perform push and pop operations to/from the semantic stack

15. Processing control constructs:

a- A compiler could choose fixed names for jump labels of control constructs, as long as it uses different names for different types of constructs (e.g., ENDIFLABEL for if statements, ENDWHILELABEL for while statements)

b- The code for the statements enclosed by a while construct are generated as part of the #finishwhile action routine.

c- CASE constructs make use of a jump table in order to branch to all possible statements following the end of the CASE contruct.

d- "Back patching" may be necessary to update information in the generated code that was not available at the time the code was issued.

e- The only information that needs to be passed between action routines for WHILE loops is the labels at the beginning and end of the loop.

16. Peephole optimization:

a- Peephole optimizations look at a small number of instructions at a time and replace them with code that perform better

b- The transformation store R1 A; load R2 B → move R1 R2 is a peephole optimization

c- Eliminating the instruction store R1 B, where B is no longer used is a peephole optimization

d- Transforming mult 4 R1 → shiftright 2 R1 is a peephole optimization

e- Peephole optimizations may replace one or several instructions with an architecture-specific instruction.

17. Sequence of compiler passes:

a- Several passes can be combined into one

b- It is possible to create one-pass compilers for all major programming languages

c- Compiler optimization always happens after semantic processing

d- If a compiler contains several optimization passes, their relative order may sometimes be reversed

e- Assembly code generation isn't always the last pass of a compiler

18. Runtime storage organization:

a- Compilers assign different programs to different address ranges so that the programs don't overlap

b- The program stack grows continuously as the program executes

c- The size of a program's code (excluding libraries) is determined at compile time

d- Program code and constants can be placed in memory sections that are read-only

e- Virtual memory helps compilers manage the placement of programs and data in memory

ADVANCED QUESTIONS

19. (advanced question) Consider question 17 again. Assume that the variables Z and K are aliased.

a- Expressions Y*Z in statement s1 has the same value number as Y*Z in s2
b- Expressions Y*Z in statement s5 has the same value number as Y*Z in s1
c- CSE can replace exactly 2 expressions by a temporary holding a previously computed expressions
d- CSE can replace exactly 3 expressions
e- CSE can replace exactly 4 expressions

20. (advanced question) Consider the following code, which was generated for a subroutine call.

```
push
push x
load A R1
add  2 R1
push R1
push &y
jsr midterm
pop
pop
pop
pop R1
store R1 Z
```

a- The subroutine call statement could be this: Z = midterm(x,A+2,y)
b- The subroutine call statement could be this: Z = midterm(x,A+2,*y)
c- The first parameter is a reference parameter, the second is a value parameter
d- All parameters are value parameters
d- It is illegal to pass the expression A+2 as the second parameter
e- The code is incorrect because there is no register save