CS3250

THE UNIVERSITY OF WARWICK

Third Year Examinations: Summer 2016

Compiler Design

Time allowed: 2 hours.

Answer **FOUR** questions. Read carefully the instructions on the answer book and make sure that the particulars required are entered on **each** answer book.
Calculators are neither necessary nor allowed.

---

1. (a) Describe the main phases of traditional compilers, and summarise the kinds of errors in input programs that must be dealt with in each phase. [6]

   (b) Consider the following grammar:

   $$\begin{array}{rcl} \mathit{Rexpr} & \rightarrow & \mathit{Rterm}\,'{+}'\,\mathit{Rterm} \mid \mathit{Rterm}\,'{-}'\,\mathit{Rterm} \\ \mathit{Rterm} & \rightarrow & \mathit{Rterm}\ \mathit{Rfact} \mid \mathit{Rfact} \\ \mathit{Rfact} & \rightarrow & \mathit{Rfact}\,'{*}' \mid \mathit{Rprim} \\ \mathit{Rprim} & \rightarrow & 'a' \mid 'b' \end{array}$$

   i. Explain why this grammar is or is not suitable for top-down parsing. [6]
   ii. Left-factor this grammar. [6]
   iii. Eliminate left-recursion from the resulting grammar. [7]

---

2. (a) Consider the following grammar, assuming S is the start symbol:

   $$\begin{array}{lcl} S & \rightarrow & A\ \# \\ A & \rightarrow & A\ z \mid B\ y \\ B & \rightarrow & A\ x \mid x\ y \mid \epsilon \end{array}$$

   Calculate the *First* and *Follow* sets (where applicable) for S, A, and B. Show the steps taken to reach your solution. [10]

   (b) Draw the *Characteristic LR Automaton* for this grammar and write, for each state, the closure of items matched at that state. [10]

   (c) Explain the need to calculate *Nullable* sets when calculating *First* sets. [5]

---

3. (a) Explain what is meant by an *Abstract Syntax Tree*. Illustrate with an outline of the process to generate an AST for a Boolean expression. [7]

   (b) Discuss the advantages and disadvantages of representing intermediate code in the form of dags instead of trees. [8]

   (c) Show how the decimal value of a binary number can be generated using *attributes*. Explain the difference between *inherited* and *synthesized* attributes. [10]

4. (a) Summarise what is meant by *three-address code* and discuss the difference between triple and quad notations. [5]

   (b) Consider the following code fragment:

```
a = 45 * a + 2 * (a + b);
```

   Translate it into three-address code and explain informally the translation process. [6]

   (c) Explain and illustrate the translation of multidimensional arrays, in particular:

      i. The processing of the declaration of an array, such as in Java or C. [7]

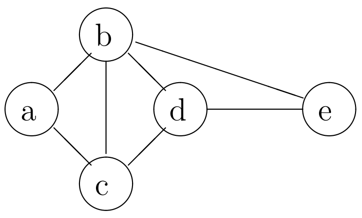
      ii. The translation of an access to an array element into some intermediate representation, assuming a uni-dimensional storage. [7]

Continued

5. (a) What are the advantages and disadvantages of structuring code into basic blocks? Explain how basic blocks can be recognized. [7]

   (b) When managing function calls, values to be stored in activation records are classified as *caller-save* and *callee-save*. Explain what is meant by these terms, and what practical difference there is in making some data caller-save or callee-save. [9]

   (c) Consider the following interference graph:



   Perform the register allocation procedure to assign one of the available 3 registers (R_0, R_1, and R_2) for each variable, and show the resulting allocation. If spilling is needed, explain the choice of variable to be spilled. Explain any algorithms and criteria that are used. [9]

6. (a) Instruction Selection algorithms try to reduce the running 'cost' of generated code. Explain the factors that affect the calculation of the cost of machine instructions. [5]

   (b) Explain the role of the architecture of the target machine (ie whether a RISC or CISC processor) in the design of the Instruction Selection phase of a compiler. [6]

   (c) In the context of data-flow analysis, explain what is meant by a *reaching definition*, and justify whether this information flows backwards or forwards through a control flow graph. [7]

   (d) Explain what is meant by a *live variable* and justify whether this information flows backwards or forwards through a control flow graph. [7]

End of paper, solutions to follow.

Marking Information - do not include in Exam Paper

1. (a) Bookwork. We went through the conceptual structure of compilers: lexing, parsing, semantic analysis, etc. Student should explain difference between lexical and other syntactic errors, as well as type checking. A good answer may also discuss possiblee bugs such as division by zero and memory leaks, but this is not necessary. [6]

   (b) Calculation.

      i. Not suitable as the grammar has left recursion and besides it is not possible to decide on the first two productions for *Rexpr*. [6]

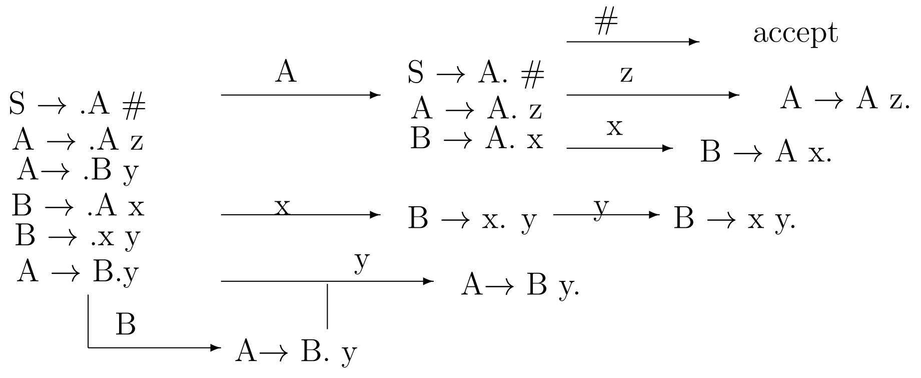
      ii. Only productions for *Rexpr* need to be changed:
      $Rexpr \rightarrow Rterm\ Lexpr$
      $Lexpr \rightarrow +\ Rterm \mid -\ Rterm$ [6]

      iii. Both *Rterm* and *Rfact* are left-recursive so need changing:

$$\begin{array}{lcl} Rterm & \rightarrow & Rfact\ Lterm \\ Lterm & \rightarrow & Rfact\ Lterm \mid \epsilon \\ Rfact & \rightarrow & Rprim\ Lfact \\ Lfact & \rightarrow & *\ Lfact \mid \epsilon \end{array} \quad [7]$$

2. (a) Calculation.
   First(S) = First(A) = First(B) = {x,y}.
   Follow(A)={x,z, #}, Follow(B)= {y}

   (b) (The tricky bit might be knowing how to deal with the epsilon production. Partial credit as appropriate.) [10]



   (c) Bookwork. Should explain about epsilon productions, such that if $A \rightarrow \alpha B$ and α is Nullable then First(A) must contain First(B).

3. (a) Bookwork. We went over many examples, where the concrete syntax tree has many artifies due to form of grammar productions, including non-terminals to make precedence rules work. On the other hand, ASTs may have semantic information, such as types, added to them in the form of attributes. [7]

   (b) Analysis. Throughout the module we discussed different approaches to obtaining optimized code, and one of them was in our approach to graphical representations of programs. Dags allow code that is repeated to be stored only once, and when the code occurs many times, such as in the case of vector/array programs, then dags can reduce the size of intermediate code as well as generate optimal low-level code directly, with reduced register usage. On the other hand, the Sethi Ullman algorithm applies to tree representation, and generates the best solution in general, but at higher implementation cost. [8]

   (c) Bookwork/problem-solving. The classic example by Knuth. Students may choose to explain only the first solution, where the value is generated with synthesized attributes. [5]

$$\begin{array}{rcl@{\qquad}rcl}
BN & \longrightarrow & BD_1 \,.\, BD_2 & BN.v & = & BD_1.v + BD_2.v/2^{BD_2.l} \\
BD & \longrightarrow & BD_1\ B & BD.v & = & 2 * BD_1.v + B.v \\
 & & & BD.l & = & BD_1.l + 1 \\
BD & \longrightarrow & B & BD.v & = & B.v \\
 & & & BD.l & = & 1 \\
B & \longrightarrow & 0 & B.v & = & 0 \\
B & \longrightarrow & 1 & B.v & = & 1
\end{array}$$

   Contrast with inherited attributes, can be made using a simpler example. [5]

4. (a) Bookwork/Analysis. In 3-AC each line consists of an operator, one or two operands and a location for the result. In triple notation the location of result is implicit in the line number, while in quad notation a variable name is used - allowing for further optimization.[5]

   (b) Problem-solving. Student should refer to the use of an AST which solves the precedence issue. The student should then be able to order the instructions. [6]

   Result:

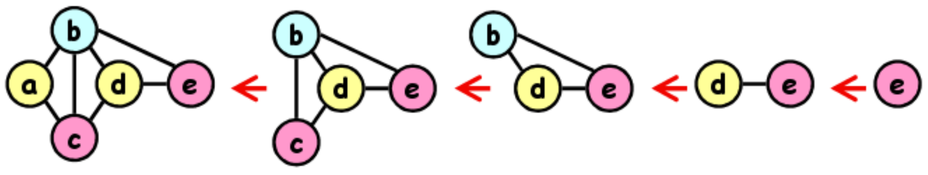
```
* a  45   t1
+ a  b   t2
*  2 t2  t3
+ t1 t3   a
```

   (c) Bookwork/analysis. In lectures we discussed the calculation necessary to address for an array element:
   $mem(A[i_1 \ldots i_n]) = mem(A) + ((i_1 - min_1) \times s_2 \times s_3 \ldots \times s_n + (i_2 - min_2) \times s_3 \times s_4 \ldots \times s_n + \ldots + (i_n - min_n)) \times w$ [7]
   The declaration yields the values of $min_1 \ldots min_n$ which need to be stored in the symbol. The widths are derived from the type of elements of the array. The value of $mem(A)$ is usually only known at runtime. [7]

5. (a) Bookwork. Separation into Basic Blocks makes global optimizations feasible - a flat flowgraph for a program might have hundreds of nodes and calculaating dataflow through them would be very time-consuming. Another advantage is that specialized algorithms can be used for instruction selection and register allocation for blocks. Basic blocks can be recognized as starting with a statement which is either the target of a jump or a statement after a jump, orthe first statement in program. Blocks end with a jump or termination of program. [7]

   (b) Bookwork/Analysis. Caller-save are registers released by the caller, which re-occupies them after return. Callee-save are those that are only spilled in case the called procedure needs the registers, if not they are left alone. Each has advantages/disadvantages in terms of spilling, so typically compilers allocate some in each category. [9]

   (c) Problem-solving. We can allocate a and d to R1, b to R2, and c and e to R3.



[9]

6. (a) Bookwork/Analysis. Cost used is typically the number of machine cycles which are needed, but other factors can also come into play, such as number of memory accesses and in fact theregisters that are used. Students will have done computer architecture and are expected to extrapolate. [5]

   (b) Bookwork/Analysis. For RISC machines we often can make the simplification that all instructions have equal cost and they cover only one node of IR, so register allocation becomes important and the Sethi-Ullman algorithm can be used. In CISC machines we have the notion of variable coverage of multiple nodes by single instructions and thus one may wish to use Maximal Munch or Dynamic-Programming based methods.[6]

   (c) Bookwork. Reaching Definitions shows, for any point in the program, which are the 'assignments' that can reach it, this is information that flows forward. [7]

   (d) Bookwork. Live Variable analysis shows which values *may* be used later and therefore the information flows backwards. [7]