University of Cape Town

Department of Computer Science

CSC3003S Final Examination

2008

Marks : 100 **Time : 3 hours**

Instructions:

- Answer every section, i.e. sections A, B, C and D.
- All questions in sections A and C are compulsory. Sections B and F offer some choice.
- Show all calculations where applicable.

Section A : COMPILERS [Answer questions 1 and 2 – both compulsory]

Question 1 : You must answer this question.

Consider the following grammar:

S → Session $
Session → Facts Question
Session → (Session) Session
Facts → Fact Facts
Facts →
Fact → ! string
Question → ? string

a) Calculate nullable, FIRST and FOLLOW sets for this grammar. [5]

b) Construct the LL(1) parsetable. [4]

c) Why is this grammar a LL(1) grammar? Give reasons for your answer by referring to your LL(1) parsetable. [1]

Question 2 : You must answer this question.

Multi-core CPUs can influence the design of modern optimising compilers. Firstly, the compiler could automatically convert traditional serial code into a parallel form, where 2 or more pieces of code run at the same time on different CPUs/cores. Secondly, the compiler itself could perform its processing in parallel. Answer the following questions with this in mind.

a) Describe one advantage and one disadvantage of separating the compiler front-end from the compiler back-end. [2]

b) Describe one advantage in having multiple distinct layers (frame generation, code generation, instruction selection, liveness analysis, etc.) within the compiler back-end. [1]

c) Static semantics should be checked before any code is generated. Discuss 2 examples of errors that can be checked for. [2]

d) Describe 3 typical optimisations that a compiler can perform on generated IR code, assuming the code being optimised runs on a single CPU/core. [3]

e) At which point/layer in the compilation process could the code being compiled be converted into segments that may run in parallel (assuming this is done independently of source language)? [1]

f) Briefly describe one technique to exploit multiple cores in the compilation process itself. [1]

Section B COMPILERS **[Answer any 3 questions ONLY]**

Question 3

a) What is the difference between a Concrete Parse Tree and an Abstract Parse Tree? [2]

b) Consider the grammar below describing the abstract syntax of expressions:

$$
\begin{aligned}
E &\rightarrow E + E \\
E &\rightarrow E - E \\
E &\rightarrow E * E \\
E &\rightarrow E / E \\
E &\rightarrow \text{id} \\
E &\rightarrow \text{num}
\end{aligned}
$$

Describe the Visitor Pattern. Supply the java code for the grammar above to illustrate this design pattern**.**

[8]

Question 4

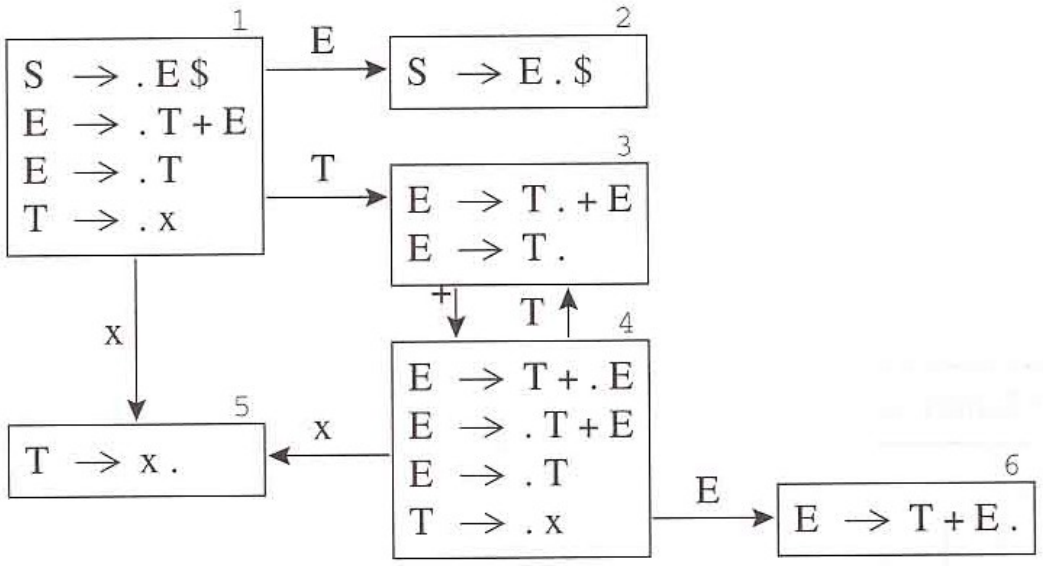
a) Construct the LR(0) and SLR parse tables for the following grammar and its given LR(0)-DFA: [10]

0 $S \rightarrow E\,\$$

1 $E \rightarrow T + E$

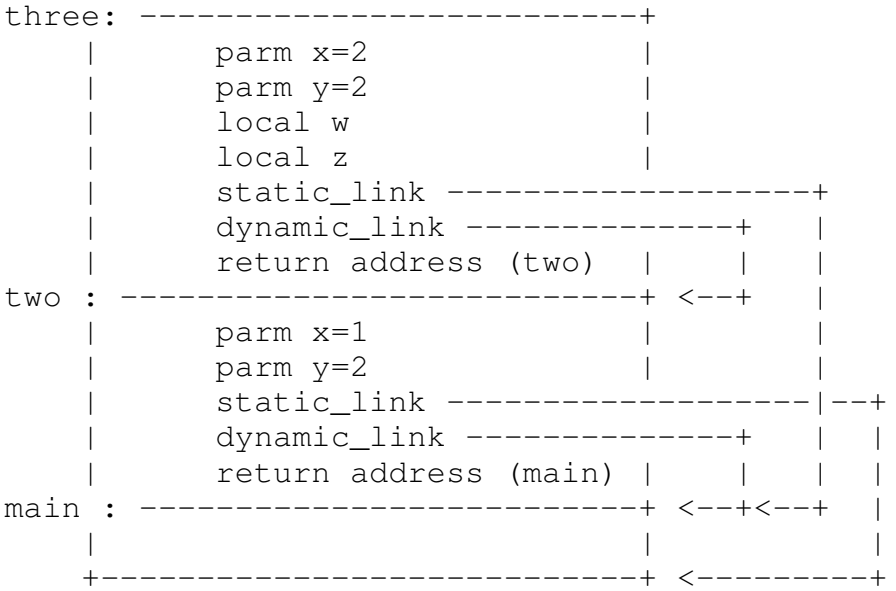
2 $E \rightarrow T$

3 $T \rightarrow x$



Question 5 [Subprogram Analysis]

a) Write the skeleton of a C-like program (with nested subprograms) that can result in the following activation record stack. All information that can be determined from the activation record stack must be indicated in the program. [5]



b) For the following program, first separate the code into basic blocks, then rearrange the blocks into traces and finally optimise the code by removing redundant jumps. Show each step separately. [5]

```
Start:                  Statement1
                        Jump X
Z:                      Statement2
                        Jump A
X:                      Statement3
                        Jump Y
A:                      Statement4
Y:                      Statement5
                        Jump Z
```

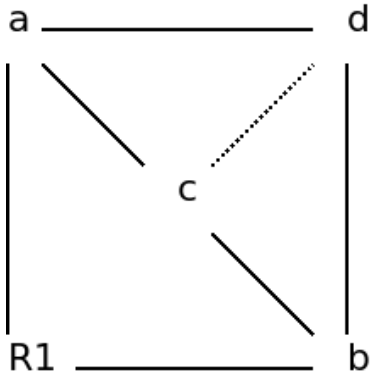
Question 6 [Register Allocation]

a) Use the iterative liveness analysis algorithm to calculate the live-in and live-out sets for each of the following statements in a program, with the initial and final live sets indicated - assume live-in (succ (d=a+b+c)) = {d}. Use the statement numbers provided. Show the succ, use, def and pairs of in/out sets. The last calculated in/out pair must be identical to the previous pair to indicate convergence. [6]

```
[live-in: a, b]
1: if (a<b)
2:   then c = a;
3:   else c = b;
4: d = a + b + c;
[live-out: d]
```

Hint: The relevant formula can be stated as follows:
out[n] = union of in[s] for all successors s of n
in[n] = union of use[n] and (out[n] - def[n])

b) Consider the following graph with nodes indicating temporaries and arcs indicating interference. Apply a register colouring algorithm to 2-colour the graph. Assume that R1 is a precoloured node and use George's criterion for conservative coalescing. At each step, draw the updated graph and briefly explain what rule has been applied. Indicate the final register allocation (R1, R2) to temporaries. [4]



Section C Theory of Algorithms **[Answer questions 7 and 8 – both compulsory]**

Question 7 : You must answer this question.

a) Explain the difference between **intractable** and **undecidable** problems. Give an example of each type. [4]

b) After weeks of secretive work, your friend, Bob, reveals to you a watertight algorithm for the decision version of the Traveling Saleman's problem. His algorithm has a time efficiency of $O(n^4)$. Should you be impressed with Bob's work? Explain the significance of his result, in the context of problem complexity. Define and use the complexity classes **P, NP** and **NP-complete** in your answer**.** [6]

Question 8 : You must answer this question.

Consider the 4 possible algorithms given below for computing $f(a,n) = a^n$:

A. $f(a,n) = a * a * a * - - - * a$ (multiply a by itself n times)

B. $f(a,n) = a^{|n/2|} * a^{|n/2|}$ (e.g. $f(a,25) = a^{12} * a^{13}$)

C. $f(a,n) = a^{n-1} * a$

D. $f(a,n) = \begin{cases} (a^{n/2})^2) & \text{if n is even and positive} \\ (a^{(n-1)/2)2} * a & \text{if n is odd and } n > 1 \\ a & \text{if } n = 1 \end{cases}$

a) What is the time efficiency of each of the following (expressed in terms of n):

i) algorithm C

ii) algorithm D [2]

b) Each algorithm above uses a different problem-solving technique. For each technique below, give the letter (A, B, C or D) of the corresponding algorithm:

i) brute force

ii) decrease by a constant

iii) decrease by a constant factor

iv) divide and conquer

[2]

c) Name 4 other problem-solving techniques studied in this course, i.e. besides those 4 given above.

[2]

d) Which of the 4 **other** techniques in (c) above would be the most efficient way of finding a^n? Why?

[2]

e) For the problem below:

i) **Name the problem-solving strategy** you would use to solve this problem.

ii) **Name a problem** studied in lectures which can most easily be adapted to solve this problem.

Jo is collecting for charity from homes in Park Avenue. Jo knows how much money each home has pledged to give, but she only has time to collect from N neighbours, and she must maximize the amount collected.
Input: an integer K (no. homes in this very long Avenue), followed by K integers (the money pledged at each home from no. 1 Park Avenue up to no. K Park Avenue);

followed by an integer M (no. queries), followed by M integers N_1, N_2, - - -, N_M. For

each N_j **find the maximum amount she can collect from N_j neighbouring homes** starting anywhere along the Avenue. Example:
Input:
6
100 500 50 600 450 20
2
2
4
Output:
1050
1600
(there are 6 homes that pledged R100, R500, R50, R600, R450 and R20 respectively; there are 2 queries: asking about sequences of length 2 and 4 respectively.
The max amount for 2 neighbours is 1050 *(as totals are 100+500 for homes 1&2, 500+50 for homes 2&3, 50+600 for homes 3&4, 600+450 for homes 4&5, 450+20 for homes 5&6 so largest is 600+450 = 1050);*
the max amount for 4 neighbours is 1600 *(totals are 1250 for homes 1-4, 1600 for homes 2-5,1120 for homes 3-6).*

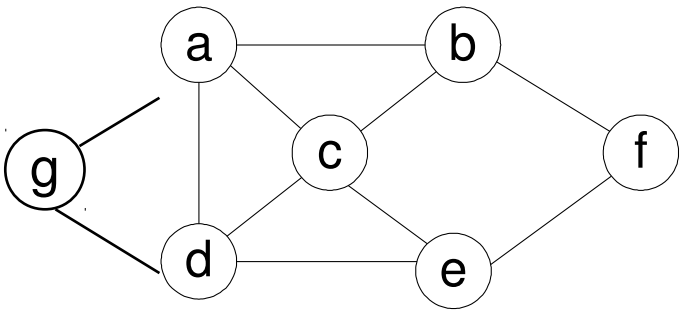
[2]

Section D THEORY OF ALGORITHMS **[Answer any 3 questions ONLY]**

Question 9

a) Prove that the ***halting*** problem is unsolvable. [4]
b) What good are approximation algorithms when we use them for NP-complete or NP-hard problems? Why do we use them? What alternatives are there to this approach? [3]

c) Apply backtracking to the problem of finding a Hamiltonian circuit in the following graph. Show all your working. [3]



Question 10

Pseudocode for the classic bubble sort algorithm is:

Bubblesort(A[0..*n-1*])
//Input: An array A[0..*n*-1] of *n* orderable elements
//Output: Array A[0...*n-1*] sorted in ascending order.
for *i*←0 **to** *n-2* **do**
 for *j*←0 **to** *n-2-i* **do**
 if *A[j+1]* < *A[j]* swap *A[j]* and *A[j+1]*

a) Do a **mathematical analysis** of this algorithm to find a closed form formula for the worst case time efficiency of bubble sort. [3]
b) Now use a decision tree analysis to find the number of key comparisons in the worst and average cases for the three element enhanced bubblesort (that stops if no swaps have been made on its last pass). [4]

c) Give an adversary argument proof that the time efficiency of any algorithm that checks the connectivity of a graph with n vertices is in $\Omega(n^2)$, provided the only operation allowed for an algorithm is to inquire about the presence of an edge between two vertices of the graph. [3]

Question 11

a) Consider the problem of maximizing the value of the following 4 items that can fit into a knapsack if that knapsack can accommodate at most 5 kgs.

Item Number	Weight of that item in kgs	Value of that item in Rands
1	2	12
2	1	10
3	3	20
4	2	15

i) **Apply Dynamic Programming without a memory function** to the given problem, but show only the **first 7** non-zero values that would be computed. [4]
ii) Now suppose a **memory function** was used. One element of the solution array that would be calculated is A[4,5]. Give **any 2** other elements of that array that would be calculated using this approach (just give the array elements e.g. A[1,1], **not** their values). [2]

b) **Name**, or briefly describe, an algorithm for **finding a value in a sorted array** using **any two** of the following techniques:

i) Brute force
ii) Divide-and-conquer
iii) Decrease-and-conquer [1]

c) **If the values to search among are not sorted**, they could be sorted first and then the above algorithms could be used for searching afterwards.

i) Would this be a good strategy if we want to find a value in a huge array?

ii) When, if ever, would it be a good strategy (to sort first and then search)? [1]

d) For the problem below:
   i) **Name the problem-solving strategy** you would use to solve this problem.
   ii) **Name a problem** studied in lectures which can most easily be adapted to solve this problem.

A population census has just taken place, and a program is needed that takes 2 input files (one is a list of integers - the age of each person in the country; the other is a list of queries) and creates an output file giving the answer to each query in turn. Each query comprises two integers L and U, L <= U, and asks how many people in the country fall within that age range. A small example:

Age file:
1 2 1 3 4 2 2 1 3
Query file:
0 45
3 5
4 60
Output file:
9
3
1

(i.e. the total no. of people aged between 0 and 45 is 9 (all the ages fall in this range);
the total no. of people aged between 3 and 5 is 3 (picks up … **3 4** … **3**);
the total no. of people aged between 4 and 60 is 1 (there's one 4, other ages too low).

[2]

Question 12

a) Consider the problem of searching for the pattern/string "BANANA" in the text
"I_SAW_HER_GAIN_BANANAS."
   i) Apply **Horspool**'s algorithm to the given problem, showing all your working. [4]
   ii) If the **Boyer-Moore** algorithm were used instead, another array, the **good-suffix table**, would be used as well. Show the contents of this good-suffix table for the search pattern "BANANA". [1]
   iii) **Why** is the Boyer-Moore algorithm more efficient than Horspool's algorithm? [1]

b) Four sorting algorithms and 4 problem-solving techniques are given below. **Pair off** the algorithms with the corresponding technique they are using in such a way that each algorithm is paired with a different technique. The techniques are:

Decrease-and-conquer; Divide-and-Conquer; Space-Time-Tradeoff; Transform-and-Conquer.

   i) **Distribution counting** (where the set of keys to be sorted is taken from a fixed range L..U and we count the number of occurrences of each key)
   ii) **Heapsort** (where we build a special data structure called a heap and then repeatedly apply the root-deletion algorithm to this structure)

iii) **Insertion sort** (we sort A[1..1], then A[1..2], then A[1..3], etc. until all keys are sorted – each time the new element is simply inserted in the right place among its already-sorted predecessors)

iv) **Quicksort** (we take the first key K and partition the array into two: keys <= K and keys > K; we then repeat the process on each of the 2 partitions)

[2]

c) For the problem below:

i) **Name the problem-solving strategy** you would use to solve this problem.

ii) **Name a problem** studied in lectures which can most easily be adapted to solve the given problem.

A large data set of friendships has been created from email logs, blogs like Facebook, etc. A program is needed that takes 2 input files (one is a list of friends; the other is a list of queries) and create an output file giving a Yes or No answer to each of these queries in turn. Each friendship is given as a pair of integers (the user-numbers of the 2 people who are friends) and each query comprises two integers X and Y, asking whether or not there is a friendship chain linking X and Y. A small example:

Friends file:

1 2
4 2
4 5
6 3

Query file:

1 5
2 3

Output file:

Yes
No

(i.e. Yes, there is a chain linking 1 and 5 (as 1 knows 2, 2 knows 4 and 4 knows 5); No, there is no chain linking 2 and 3 (as 2 only knows 1 and 4, and 4 only knows 5)).

[2]

- END OF EXAMINATION PAPER -