# University of Cape Town

# Department of Computer Science

# CSC3003S Class Test Rewrite

# 2007

---

**Marks : 20**

**Time : 45 minutes**

**Instructions:**

- Answer all questions from Section A and 3 questions from Section B.
- Show all calculations where applicable.

---

## Section A [ Answer Question ONE – this is compulsory ]

### Question 1

a) Before any code can be generated, context-sensitive analysis is applied to check for errors. Discuss 3 types of errors that can be detected. [3]

*fwd declarations, unused decls, type inconsistency, null ptr dereference, array out of bounds, etc.*

b) Although intermediate representations are widely accepted, there are still disadvantages. Discuss 2 disadvantages in using intermediate representations in compilers. [2]

*disad: additional processing, mismatch between IR and real MC, etc.*

## Section B [ Answer 3 questions ONLY ]

### Question 1: Activation Records

a) For what types of programs do we NOT need to store activation records on a stack? [1]

*no recursion*

b) Assuming stack-based activation records, draw the full activation record stack corresponding to the function **not_main** at the position marked "%%%", as called by the function **main** in the following program: [4]

```
function main
start
   call output not_main (1, 2)
stop
function integer not_main (x, y)
start %%%
   return x + y
```

```
    stop

not_main: ----------------------+
    |      parm x=1               |
    |      parm y=2               |
    |      return value           |
    |      static_link ------------------+
    |      dynamic_link -------------+   |
    |      return address (main) |    |   |
main : --------------------------+ <--+   |
    |                            |        |
    +----------------------------+ <------+
```

*Minus one mark for each missing field.*

## Question 2: Basic Blocks and Traces

a) Why must traces not overlap? [1]

*to ensure no statement is executed twice*

b) Why must traces cover all nodes of the IR tree? [1]

*to ensure every statement is executed*

c) Separate the following program into basic blocks, generate a set of traces and then optimise the resulting code. Show each step of the process. [3]

```
label a:
 statement 1
 jump c
label b:
 statement 2
label c:
  jump b
```

*[1] basic blocks*

```
label a:
  statement 1
  jump c
label b:
  statement 2
  jump c
label c:
```

*jump b*

*[1] traces*

*label a:*

*statement 1*

*jump c*

*label c:*

*jump b*

*label b*

*statement 2*

*jump c*

*[1] optimisation*

*label a:*

*statement 1*

*label c:*

*label b:*

*statement 2*

*jump c*

## Question 3: Optimisations

a) What is the difference between peephole optimisation and global optimisation? [1]

*peephole optimisation applies only to small portion of code as opposed to entire program/module*

b) Briefly discuss 2 types of peephole optimisations and provide code examples to indicate the effect of each. [4]

*constant folding: 1+2 => 3*

*constant propagation: x=1; y=x => x=1; y=1*

*loop unrolling: for ( int i=0; i<2; i++ ) x[i]=0 => x[0]=0; x[1]=0*

*etc.*

## Question 4: Instruction Selection

a) Describe the steps of the maximal munch algorithm. [3]

*Start at top of tree*

*Find largest matching tile and cover nodes*

*Repeat for remaining subtrees*

*Generate instructions in reverse order*

b) Maximal munch is an optimal algorithm. What is the different between an optimal algorithm and an optimum algorithm? [2]

*Optimal = no 2 tiles can be replaced by one with lower cost*

*Optimum = no lower cost tiling*