國立交通大學試題紙

九十九學年度第一次
博士班資格考

科目：編譯器設計(A)　　日期： 100 年 1 月 27 日　第 1 頁 共 1 頁

請"✓"明　✓不可看書　可看書

* 請將答案依題號順序寫入答案卷

答題時字跡需工整，否則不予計分。Write your answers legibly; otherwise you will get zero score.

1. (15 points) Find the SLR(1) parse tables for the following grammar:

1. S → V
2. S → W
3. V → a X b
4. W → a X d
5. X → p
6. X → q

You need to show the detailed process of constructing the parse tables.

2. (15 points) Please design an abstract syntax tree for a simple programming language. The programming language should include +, - *, /, ¡, ¿, =, assignment statements, if statements, while loops, and case statements. Finally, write down a translation scheme that translate input programs into corresponding abstract syntax trees.

3. (20 points) Please present a scheme (or pseudo code) for checking exceptions, including exception declaration, exception handling routines, catching an exception, and propagating an exceptions. You may start from Java or C++ and ignore all the unnecessary details.

國立交通大學試題紙

九十九學年度第一次
博士班資格考

科目：編譯器設計(B)　　日期： 100 年 1 月 27 日　第 1 頁 共 1 頁

請 "✓" 明　✓不可看書　可看書

* 請將答案依題號順序寫入答案卷

答題時字跡需工整，否則不予計分。Write your answers legibly; otherwise you will get zero score.

1. Code Generation
   a) What is IR (Intermediate Representation)?
   b) Why does a compiler often generate IR before emitting the final machine code?
   c) Why some compilers use more than one form of IRs, for example, the GCC compiler uses a high-level tree-oriented IR and a low-level RTL IR form.
   d) What IR structures can be efficiently used to represent Common Sub-Expressions (CSE)? For example, in the following expression
      (a-b) * (a-b),
      a-b is a CSE.
   e) How can a compiler capture such CSEs? Your described algorithm should be able to capture arbitrarily complex CSE like such as (a-b)*c*(d-e*f).
   f) CSE may cross multiple statements, such as the sub-expression (a-b) in the following code fragment:
      S1: x = (a-b)*c;
      S2: y = (a-b)*d;
      How is catching such CSEs different from methods used in (e)?

2) Code Generation for Subroutines

It is important for a compiler to generate efficient code for subroutines. Tasks associated with a subroutine calling sequence typically include passing parameters, changing the stack/frame pointer, and saving registers.

a) Why do compilers typically allocate space for arguments in the stack, even when the arguments are actually passed in registers?
b) What are the pros and cons of having the caller to save and restore registers.
c) List the optimizations that can be made to the calling sequence in the case of a ***leaf*** subroutine. (A ***leaf*** routine calls no other routines)
d) Automatic subroutine inlining can be used to reduce calling overhead. Other than calling overhead reduction, what additional benefits can also be introduced by subroutine inlining?
e) Subroutine inlining may significantly increase the generated code size. Describe an algorithm for automatic inlining that minimizes the increase of code size while keeping most of the benefit of optimizations.