IMPERIAL COLLEGE LONDON

DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING
EXAMINATIONS 2016

EIE PART II: MEng, BEng and ACGI

Corrected Copy

LANGUAGE PROCESSORS

Thursday, 26 May 2:00 pm

Time allowed: 2:00 hours

There are THREE questions on this paper.

Answer ALL questions.
Q1 carries 40% of the marks. Questions 2 and 3 carry equal marks (30% each).

Any special instructions for invigilators and information for candidates are on page 1.

Examiners responsible First Marker(s) : D.B. Thomas
Second Marker(s) : J.V. Pitt

LANGUAGE PROCESSORS

Ensure throughout that your written characters are unambiguous, especially in terms of '*' versus '+' and white-space. If necessary, use a square under-bracket to indicate space characters.

Bison and C++ will be interpreted by a human, so some syntax errors can be tolerated as long as the intended solution is clear.

1. a) How is a right-linear grammar classified using Chomsky's hierarchy? [2]

b) Is Bison a top-down or bottom-up parser? Use a feature or capability of Bison to support your answer. [3]

c) Where can branches appear within a basic block? [2]

d) Given an n character input string, what is the worst-case size needed for the stack of an LR(1) parser? [2]

e) Describe the following sets: symbols, terminals, and non-terminals. [3]

f) Left-factor the following grammar:
`X ::= 'c' 'a' 't' | 'c' 'a' 'r'` [2]

g) Give two advantages of interpreters over compilers. [3]

h) Given the following grammar:
`E ::= E '+' E | E '*' E | Num`
use the input string `6+7*10` to show that the grammar is ambiguous. [5]

i) Give the First set for the production "α b", where α is a non-empty sequence of symbols, and b is a terminal. [4]

j) Give pseudo-code for a general-purpose DFA. [6]

k) Consider the following chain of reasoning:

- Fact: Context-free grammars are defined over a finite set of tokens.
- Fact: The set of identifiers in C is infinite.
- Inference: C does not have a context-free grammar.

This appears to lead to a contradiction with:

- Fact: context-free grammars *do* exist for C.

i) Identify the faulty reasoning that leads to the contradiction. [4]

ii) Describe the technique used to resolve this problem in compilers. [4]

2. In the following, assume we are working with regular expressions with the following constructs: sequence; alternation; one-or-more; zero-or-more; groups; character ranges; and anchors (start and end of string).

Many regular expression engines also support *capture groups*, which allows the user to indicate parts of the match that should be remembered (captured), and made available under a label. The labels can then be referred to from a substitution string. For our purposes, we will state that all bracketed groups define a capture group, and we can refer to them using the symbol $n, where n is a decimal integer. $1 then defines the first capture group, $2 the second, and so on.

Some examples of using capture groups are:

	Regex	Substitution	Input	Output
1	`[a-z]([0-9])`	`$1`	c4	4
2	`([a-z]+)=([0-9]+)`	`$1:$2`	debug=3	debug:3
3	`[a-z]+@([a-z]+([.][a-z]+)+)`	`X@$1`	bib@bob.co.uk	X@bob.co.uk
4			gpg.tar.gz	gpg

a) Write a regular expression and substitution pattern for taking a file name and extracting just the base filename, excluding any filename extensions. An example input and output is shown in line 4 of the table. [3]

b) What is the order of precedence for the regular expression constructs, from highest to lowest? [3]

c) Give a Bison-like definition of a symbol "CharRange", which recognises a regular expression character range (e.g. `[a]`, `[01]`, `[0-9a-z]`). Terminals can be defined as literals or using regular expressions. You can define intermediate helper symbols if necessary. [6]

d) Give the remaining Bison-like grammar for recognising regular expressions. [7]

e) The regular expression `^ftp://([a-z]+):([a-z0-9_]+)@[.]+$` is designed to match URLs containing a user name and password. Draw a DFA for recognising this pattern. [6]

f) The URL regex contains two capture groups. Assume there is an additional DFA annotation called `append[n]`, which pushes the current input character onto the end of capture group n. Where should the annotations be added to your DFA in order to capture the groups? [5]

3. The following AST models a simple language where all statements are also expressions:

```
struct Expr {};

struct Num      : Expr{ int value; };

struct Add      : Expr{ Expr *left; Expr *right;  };

struct VarRef   : Expr{ string id; };

struct VarDecl  : Expr{ string id; Expr *init; Expr *body; };

struct Assign   : Expr{ string target; Expr *source;  };

struct Sequence : Expr{ vector<Expr*> body;  };

struct While    : Expr{ Expr *cond; Expr *body; };

struct Func     : { string name;  vector<string> args;  Expr *body; };
```

All expressions return a value. Statement-like expressions (VarDecl, Assign, Sequence, While) return the value of the last evaluated sub-expression. While loops execute while the condition evaluates to a non-zero value.

An example function for multiplication is:

```
Func[ multiply, [a,b],
  VarDecl[ res, 0,
    Sequence[
      While[ b,
        Sequence[
          Assign[ b, Add[ VarRef[ b ], Num[ -1 ] ] ],
          Assign[ res, Add[ VarRef[ res ], VarRef[ a ] ] ]
        ],
      ],
      res
    ]
  ]
]
```

a) Translate the example function to C. State any assumptions needed. [5]

b) The AST needs to be compiled to MIPS assembly. Define a function calling convention which can support this language, and describe a function call as seen by both caller and callee. (This does *not* have to follow the GNU ABI, and generality is more important than efficiency.) [6]

c) Give a general MIPS assembly template for the code emitted for a While loop. The template should follow your calling convention and the semantics of the language. [6]

d) A virtual function called `codeGen` is going to be added to the Expr node. Give a function prototype (i.e. arguments and return type) for the `Expr::codeGen` function, and add minimal comments to explain how it works. If necessary, helper classes or function declarations can be used. [6]

e) Give C++ code for the implementation of `While::codeGen`. [7]