**GRIFFITH UNIVERSITY**

School of Information and Communication Technology

# 3136CIT Programming Language Implementation

Final Examination, Semester 1, 2005

**Details**

| | |
|---|---|
| Total marks: | 100 (40% of the total marks for this subject) |
| Perusal: | 15 minutes |
| Duration: | 3 hours |
| Time: | 8:30 am |
| Date: | Thursday 23 June 2005 |

**Instructions**

1. **Don't panic!**
2. Read these instructions carefully. Use the perusal time wisely.
3. **You may not write anything during perusal.**
4. **This is a closed book examination.** Written materials other than language translation dictionaries (on paper) are not permitted. Calculators, PDAs and other electronic devices are not permitted.
5. The examination has eight questions worth 100 marks in total.
6. **Write answers to all questions in the answer booklets provided.** Start the answer to each question on a new page. Please write neatly. Use blue or black ink (and definitely not red ink). Dark pencil is acceptable.
7. In general, explicitly state any assumptions you are making and show all working. Credit may be given for incomplete answers provided the work shown is understandable (and correct).

Paper number: NA 462 EOS

**3136CIT Programming Language Implementation (2005/1)**

1. (a) Briefly describe the four main tasks of a lexical analyser.

   (b) Give a regular expression for the set of strings of 0's and 1's that start with 1 and do not contain the substring 00, *e.g.*, 1, 101, 10111010.

   (c) Give a regular expression for the set of programming language identifiers, each of which starts with a letter and contains only letters and digits, *e.g.*, a, a123bc4,

   (d) Give a regular expression for the set of programming language floating point constants, where each such constant contains either a decimal point or an exponent part, *e.g.*, 123.45, 123e45, 1.23e45. The strings 123. and .45 are *not* floating point constants for the purposes of this question. Ignore the possibility of negative constants or negative exponents.

   In the last two parts, you may use the symbols LETTER and DIGIT to represent a letter or digit respectively.

   **(10 marks)**

2. (a) Briefly describe the two main tasks of a parser.

   (b) What language is described by the following context-free grammar?

   $S \rightarrow \epsilon \mid aSbS \mid bSaS$

   The terminal symbols in this grammar are $a$ and $b$; $\epsilon$ denotes the empty sequence.

   (c) Give a context-free grammar for the set of nonempty strings of $a$'s and $b$'s that are palindromes, *i.e.*, that read the same forwards and backwards, *e.g.*, $a$, $abba$, $bbababb$.

   (d) Give a context-free grammar for the set of strings of $a$'s, $b$'s and $c$'s of the form $a^i b^j c^k$, where $i + j = k$, *e.g.*, $aabbbccccc$.

   **(10 marks)**

3. Consider the following grammar for boolean expressions:

   $B \rightarrow \mathbf{true} \mid \mathbf{false} \mid B \textbf{ and } B \mid B \textbf{ or } B \mid (B)$.

   The terminal symbols in this grammar are $\mathbf{true}$, $\mathbf{false}$, $\mathbf{and}$, $\mathbf{or}$, ( and ).

   (a) Give a rightmost derivation for the following sentence in the language defined by this grammar: $\mathbf{true\ and\ false\ or\ true}$

   (b) Use this example sentence to show that the grammar is ambiguous. (Remember to state carefully why your example shows that the grammar is ambiguous.)

   (c) Give an unambiguous grammar for the same language.

   (d) What does it mean for a context-free language to be *inherently* ambiguous?

   (e) Give an example of an inherently ambiguous language.

   **(10 marks)**

4. Consider the following grammar for arithmetic expressions:

   $E \to F \mid E + F$
   $F \to \mathbf{num} \mid \mathbf{id} \mid (E)$

   The terminal symbols in this grammar are **num**, **id**, (, ) and +.

   (a) What is the start (or first) set of the nonterminal symbol $E$?

   (b) What is the follow set of the nonterminal symbol $E$?

   (c) Explain why this grammar is not LL(1).

   (d) Transform the grammar into an LL(1) grammar for the same language. **Hint** Use regular expressions on the right hand sides of rules in your transformed grammar.

   (e) Write a recursive descent parsing functions in C (or very clear pseudo-code) for the nonterminal symbols $E$ and $F$ in your transformed grammar. Your parsing functions only need to recognise whether or not an input string is a legal arithmetic expression defined by the initial grammar. You may use the global variable `token` and the (void) functions `getToken()` and `error()` in the normal way.

   **(15 marks)**

5. Consider the following grammar for balanced parenthesis strings, *e.g.*, (), (()())().

   $S \to \epsilon \mid (S)S$

   Now, consider the following SLR(1) parsing table for this language of balanced parenthesis strings.

| **State** | **Input** | | | **Goto** |
|---|---|---|---|---|
| | ( | ) | $ | S |
| 0 | Shift 2 | Reduce ($S \to \epsilon$) | Reduce ($S \to \epsilon$) | 1 |
| 1 | | Shift 3 | Accept | |
| 2 | Shift 2 | Reduce ($S \to \epsilon$) | Reduce ($S \to \epsilon$) | 3 |
| 3 | | Shift 4 | | |
| 4 | Shift 2 | Reduce ($S \to \epsilon$) | Reduce ($S \to \epsilon$) | 5 |
| 5 | | Reduce ($S \to (S)S$) | Reduce ($S \to (S)S$) | |

   (a) Each state consists of a set of "items" of the form $A \to X_1 \ldots X_m \,.\, X_{m+1} \ldots X_n$. Carefully describe what each such item means.

   (b) Assume the parsing stack contains only states during parsing. Carefully define the effect of "Shift $m$" and "Reduce ($A \to X_1 \ldots X_n$)" actions on the parsing stack.

   (c) Given the input ()()$, trace the behaviour of this parser. Initially, the parsing stack contains state 0. At each step, show the parsing stack, the remaining input, and the action taken.

   (d) In this parser, state 0 consists of the following items:
   $S' \to . \, S\$$
   $S \to .$
   $S \to . \, (S)S$
   (Here, $S'$ is a new nonterminal symbol.) Which items does state 2 consist of?

(e) Under what conditions is a grammar SLR(1)?

**(15 marks)**

6. (a) Briefly describe the two main tasks of a semantic analyser.

   (b) What are the two main data structures required by a semantic analyser to perform these tasks?

   (c) Define the "scope" of a binding (of an identifier to an entity).

   (d) Give one example of a typical *synthesized* attribute.

   (e) Describe a typical data structure used to implement a symbol table, and justify two main features of the data structure.

   (f) Consider the following grammar for variable declarations:

   $decs \rightarrow type\ vars$
   $type \rightarrow$ **int** | **double**
   $vars \rightarrow$ **id** | **id ,** $vars$

   The terminal symbols in this grammar are **int**, **double**, **id** and **,**.

   Suppose we wish to assign a value $integer$ or $real$ to each **id** in a variable declaration. Give an attribute grammar that assigns a type attribute to each $decs$, $type$, $vars$ and **id** node in the syntax tree for a declaration.

**(15 marks)**

7. Answer **either** part (a) **or** part (b) below (but not both).

   (a) Carefully describe a mechanism for dynamic storage management, on a heap, capable of implementing C `malloc()` and `free()` functions. In your description, indicate how these two functions should be implemented, discuss the extent to which internal or external fragmentation may occur, and consider the efficiency of your implementation.

   (b) Carefully describe the structure and contents of an activation record (or stack frame) for a procedural language with nested scopes, *i.e.*, with nonlocal variables. Assuming the existence of registers *sp* (stack pointer) and *fp* (frame pointer), carefully describe what operations must be performed on procedure call and on procedure return. Also indicate how both local and nonlocal variables are accessed.

**(10 marks)**

8. (a) Briefly describe three different, possible target languages for code generation.

   (b) Give a sequence of three-address code instructions that is equivalent to the following (TINY) source code fragment. Avoid any unnecessary unconditional jumps in your solution.

```
read x;
if x > 0 then
    fact := 1;
    repeat
        fact := fact * x;
        x := x - 1
    until x = 0;
    write fact
end
```

(c) Consider the class of integer expressions constructed from identifiers, integer constants, addition and assignment. The assignment operator assigns the value of its second operand to its first operand (an identifier), and returns the value of the second operand.

Write a (void) function `genCode()` in C (or very clear pseudo-code) that takes a syntax tree representing such an expression as an argument and prints an equivalent sequence of three-address code instructions. For example, given a tree corresponding to the expression `a+(b=a+2)`, `genCode()` might output the following sequence:

```
t1 = a + 2
b = t1
t2 = a + t1
```

(d) Name two important optimisation techniques and give a clear example of each one.

**(15 marks)**



**End of examination**