CPSC 320 Winter 2001

Midterm I—Wednesday, 7 February 2001

Name(Printed) : ____________________

Signature : ____________________

StudentNumber : ____________________

ACRE	AREA	BALE	BAND	BARD	BASS
BETA	BIRD	BLOT	BOOK	BREW	CAMP
CHIP	CLAN	COAT	COIL	CORN	CROW
CURL	DARK	DEER	DOSE	DROP	DUCK
DUSK	FARE	FILM	FLAX	GAZE	GIFT
GOLD	GULF	HINT	HORN	HULL	IBOU
INCH	IRIS	ISLE	KERN	KILN	KITE
LANE	LARK	LENS	LOFT	LURE	MALT
MANX	MESH	MINK	MOTH	MOVE	MUSK
NAVY	NEWT	NOON	OATS	OBOE	OPAL
PARK	PINE	POET	RAFT	REED	RING
RUBY	RUFF	SEAM	SEED	SHOP	SILK
SINE	SNIP	SOAP	STUB	TASK	TAXI
TEAM	TELL	TEXT	TIDE	TILT	TOIL
TOME	TOUR	TURN	VANE	VISA	WALL
WICK	WOLF	WRIT	YARD		

Question	Score
1	/3
2	/3
3	/2
4	/7
5	/7
6	/3
7	/3
8	/2
9	/2
10	/5
11	/3
12	/3
13	/4
14	/3
Total	/50

Instructions

- Write the word marked above on each page of questions of your exam. Do not put any other identifying marks on any page of your exam. Failure to put the circled word on a page of your exam may result in no marks being awarded for that page.
- *Read each question carefully. Ask yourself what the point of the question is.* Answer each question. *Check to make sure that you have answered the question asked.*
- Answer all questions on the exam sheet. If you do some of your work on the back of a page, clearly indicate to the marker what work corresponds with which question.
- This is a **50** minute exam. This exam contains **7** pages of questions not including this cover page. Make sure that you have all of them.
- Partial marks shall be awarded for clearly identified work.
- This exam counts as **20%** of your total grade.

Programming Language Principles

(3) **1.** In C and C++ macro-expansion occurs after comment and blank removal, but early in the parsing process. Comment on how C-style macro expansion affects the *writeability* and *reliability* of C and C++ code.

Syntax

(3) **2.** In C++ "`list<list<int>>`" results in a syntax error because of how `>>` is parsed. Explain what goes wrong, and how this relates to how language parsing works in general.

(2) **3.** C++, like most modern computer languages, is primarily *free-form.* What features, if any, are *not* free-form?

4. One common form of ambiguity relates to the precedence of operators in expressions. Consider the expression

$$\mathtt{x} \times \mathtt{y} + \mathtt{z} \times \mathtt{w} \qquad (1)$$

(2) (a) Given the grammar

$$\begin{array}{lcl} \langle \mathit{expr} \rangle & ::= & \langle \mathit{expr} \rangle + \langle \mathit{expr} \rangle \quad | \quad \langle \mathit{expr} \rangle \times \langle \mathit{expr} \rangle \\ & | & \langle \mathit{variable} \rangle \\ \langle \mathit{variable} \rangle & ::= & \mathbf{w} \mid \mathbf{x} \mid \mathbf{y} \mid \mathbf{z} \end{array}$$

show two different ways to parse (1) as an $\langle \mathit{expr} \rangle$.

(3) (b) Given the (*unusual!*) grammar

$$\begin{array}{lcl} \langle \mathit{expr} \rangle & ::= & \langle \mathit{term} \rangle \times \langle \mathit{expr} \rangle \quad | \quad \langle \mathit{term} \rangle \\ \langle \mathit{term} \rangle & ::= & \langle \mathit{term} \rangle + \langle \mathit{variable} \rangle \quad | \quad \langle \mathit{variable} \rangle \\ \langle \mathit{variable} \rangle & ::= & \mathbf{w} \mid \mathbf{x} \mid \mathbf{y} \mid \mathbf{z} \end{array}$$

what parsings of expression (1) as an $\langle \mathit{expr} \rangle$ are possible?

(2) (c) What does it mean for a grammar to be unambiguous?

5. This question is about the rôle of context-free grammars in modern programming languages.

(2) (a) What is a context-free grammar?

(2) (b) Most programming languages are nearly, but not entirely, context-free. What makes most of them non context-free?

(2) (c) What small modifications allow an essentially context-free parser to be used with most programming languages?

(1) (d) Is the language for which we defined a formal semantic model context-free?[1]

(3) **6.** Explain some of the principal ways that the grammar of FORTRAN differs from that of most modern computer languages.

Formal Denotational Semantics

(3) **7.** Let p stand for the program "`x:=y+3.0; y:= 2.0; z:=x-y .`"; then:

(a) $\mathcal{C}[\![p]\!] = \mathcal{C}[\![\textbf{x:=y+3.0}]\!] \circ \mathcal{C}[\![\textbf{y:= 2.0}]\!] \circ \mathcal{C}[\![\textbf{z:=x-y}]\!]$.

(b) $\mathcal{C}[\![p]\!] = \mathcal{C}[\![\textbf{z:=x-y}]\!] \circ \mathcal{C}[\![\textbf{y:= 2.0}]\!] \circ \mathcal{C}[\![\textbf{x:=y+3.0}]\!]$.

(c) none of the above.

Briefly justify your answer.

[1]In fact, I never entirely specified the actual grammar. Given a likely grammar for the language, is it context-free?

(2) **8.** When we added semantics for expressions we gave the rule

$$\mathcal{E}[\![r]\!](s) = \iota(r) \qquad \text{where } r \text{ is a floating point literal} \qquad (2)$$

(a) What is the point of the function ι?

(b) Why does the state s appear on the left hand side of (2)?

(2) **9.** Give an equation defining $\mathcal{E}[\![e_1 + e_2]\!]$, where e_1 and e_2 are meta-syntactic variables standing for arbitrary expressions.

Informal Semantics

10. Some of the following questions are about the notion of binding.

(2) (a) To what, in general, does binding refer?

(2) (b) Binding times may be either static or dynamic. What are some further subdivisions of static binding times suggested in the book?

(1) (c) What features in C++ use dynamic binding?

(3) **11.** In C++ functions may be templated on their arguments. For instance, in the code

```
1  template<class Q>
2  void swap(Q& q1, Q& q2) { Q& q_temp(q1) ; q1 = q2; q2 = q_temp;}
3
4  int main()
5  {
6     int i1, i2; double d1, d2;
7     swap(i1, i2);
8     swap(d1, d2); return 0;
9  }
```

the function `swap` on line 8 is bound to `swap<int>`, and the function `swap` on line 9 is bound to `swap<double>`. Describe as specifically as possible when the name of a templated function is bound to its instance.

(2) **12.** (a) What does *strict* or *applicative* evaluation order of expressions mean?

```
#include <iostream>
using namespace std ;
int x = 3, y=4 ;
void h()
{
    char z[] = "6" ;
    cout << x << ", " << y << ", " << z << endl ;
    return ;
}
void g() { int y = x; h(); return; }
int main() { int x = 2+y++; g(); return 0; }
```

Figure 1: Code for Question 13.

(1) (b) Can short-circuit operators (such as '`&&`' in C++ or '`and also`' in ADA and ML) be simulated by functions with strict evaluation rules?

13. Given the C++-like code shown in Figure 1:

(2) (a) what would be printed in a language that uses *static* scoping? Explain.

(2) (b) what would be printed in a language that uses *dynamic* scoping? Explain.

```
int f(bool flag=false)
{
    static int n = 0 ;
    if (flag) n = 0 ;
    return ++n ;
}

int g(int,int,int x) { return x ; }
```

Figure 2: Code for Question 14

14. The following question is about the order of evaluation in C++. Assume that we have the functions `f` and `g` defined as shown in Figure 2 (page 7)

(1) (a) Explain why

```
cout << (f(true),f(false),f(false)) << endl
```

is well-defined.

(2) (b) Explain why

```
cout << g(f(true),f(false),f(false)) << endl
```

is not well-defined.