Exam 1, Computer science 356, Fall 2014

Time allowed: 75 minutes. Total points: 75.

Name: Solution

Question 1. (30 points) — 1 point for each answer. Circle "T" for true, and "F" for false.

In C, the keyword `for` is both a token and a lexeme.	T	F
Let L be a language that is not ambiguous. Then every string in L has a unique derivation.	T	F
Let L' be an ambiguous language, and suppose string $s \in L'$. Then s has at least two distinct parse trees.	T	F
In C++, output parameters can be implemented using both pointers and references.	T	F
In C, `malloc` allocates memory on the heap.	T	F
While a C++ program is running, the operating system performs garbage collection on behalf of the program.	T	F
The EBNF rule $\langle x \rangle \rightarrow \langle y \rangle [(a\|b)] \langle z \rangle$ is equivalent to the BNF rule $\langle x \rangle \rightarrow \langle y \rangle \langle z \rangle \mid \langle y \rangle a \langle z \rangle \mid \langle y \rangle b \langle z \rangle$	T	F
Consider the C function `f()` defined by `void f(double *y) { double *z = y; double w = *z; }` The function `f()` causes a memory leak.	T	F
The function `f()` in the previous question could cause a segmentation fault due to an illegal memory access.	T	F
In C, `x[4]` and `(*x+4)` always produce the same result.	T	F
In C, suppose that a union `u` contains exactly 2 fields: a float and a double. Then `sizeof(u)` equals `sizeof(float)+sizeof(double)`.	T	F

Consider the following sequence of C++ statements:
`int** x = new int*[3];`
`x[0] = new int(5);`
`x[1] = new int(8);`
`delete x[0];`
`delete x[1];`
`delete x[2];`
`delete [] x;`
This code produces a memory leak.
	T	F

The code in the previous question could cause a segmentation fault due to an illegal memory access.	T	F

The following C++ snippet is an example of ad hoc polymorphism:
`int countObjects(Bird bird) { ... }`
`int countObjects(Fish fish) { ... }`
	T	F

The snippet in the previous question is an example of overloading.	T	F

According to the technical definition of *language* used in this course, the set of all integers (written in decimal notation, as in "`5523`") is a language.	T	F
Every language can be represented by a BNF grammar.	T	F
Suppose the language L can be represented by two distinct BNF grammars. Then at least one of these grammars is ambiguous.	T	F
In an attribute grammar, the value of an intrinsic attribute typically depends on the value of inherited attributes.	T	F
In C, "`int* x;`" and "`int *x;`" mean the same thing.	T	F
In C++, structs can be placed either on the stack or on the heap.	T	F
Suppose we are compiling our C++ on a 64-bit machine. (So pointer data types occupy 64 bits.) Consider a function with the signature `int g(char* c)`. The total size of the parameter passed to this function is 8 bytes.	T	F
As in the previous question, assume we are using a 64-bit machine. This time consider a function with the signature `int g(char &c)`. The total size of the parameter passed to this function is 8 bytes.	T	F
Consider the C program given in Figure 1 below. The output of this program is "`cbc`".	T	F
Again consider the C program given in Figure 1, and assume it is compiled for a machine on which `int`s occupy four bytes. Then the size of the data type `U` is five bytes.	T	F
Consider the following snippet of C code: `int a = 6; int b = 8; int &c = a; int d = c; c = 3;`. After this snippet has been executed, the value of `a` is 3.	T	F
Consider the same snippet as in the previous question. After the snippet has been executed, the value of `d` is 3.	T	F
In C++, declaring a function f virtual in a base class imposes a small performance penalty on any functions that override f in derived classes.	T	F
In C++, declaring a function f virtual in a base class imposes a small performance penalty on calls to f by instances of the base class.	T	F
In Java, suppose a class `Wombat` has been defined with a default constructor. Then the snippet "`Wombat w = new Wombat()`" causes a wombat object to be placed on the stack.	T	F

```
#include <stdio.h>
typedef union {
  char x;
  int y;
  char z[5];
} U;
int main(int argn, char** argv) {
  U u;
  u.y = 54321;
  u.z[0] = 'a';
  u.z[1] = 'b';
  u.x = 'c';
  printf("%c%c%c\n", u.z[0], u.z[1], u.x);
}
```

Figure 1: A C program referred to by the true/false questions.

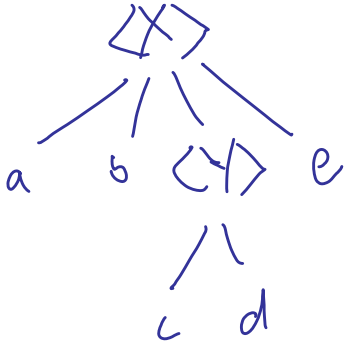
Question 2. (5 points) In his 1968 letter to the editor of CACM, Dijkstra suggests that `goto` statements could be useful for "alarm exits". What did he mean by this?

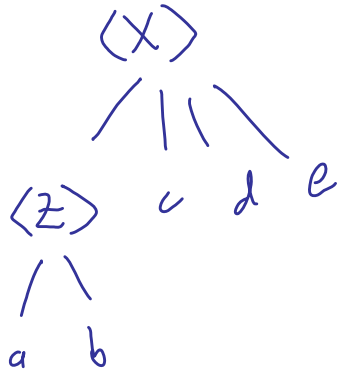
"Alarm exits" are exceptional error conditions, for which it makes sense to jump directly to some cleanup code (via 'goto') and then exit.

Question 3. (10 points) Prove that the following grammar is ambiguous.

$$\langle \mathtt{X} \rangle \rightarrow \mathtt{a\,b}\,\langle \mathtt{Y} \rangle\,\mathtt{e} \mid \langle \mathtt{Z} \rangle\,\mathtt{c\,d\,e}$$
$$\langle \mathtt{Y} \rangle \rightarrow \mathtt{c\,d} \mid \mathtt{d\,e}$$
$$\langle \mathtt{Z} \rangle \rightarrow \mathtt{a\,b} \mid \mathtt{b\,c}$$

To prove ambiguity, we must exhibit a string in the language with 2 distinct parse trees. Consider the string "abcde". It has the 2 parse trees shown below, so the claim is proved.





Question 4. (10 points) In your own words, explain the circumstances under which it is a good idea to declare a C++ member function `virtual`, and give a brief reason for your answer.

Any member function that could be overridden in a subclass should be virtual. This ensures the correct behavior of pointers to the base class: if B is derived from A, the code A* b = new B(); b -> f() will call B's version of f, provided f is virtual.

Question 5. (5 points) Consider the C program below. Fill in each of the spaces marked "___" with exactly one of the symbols `x`, `y`, `z`, so that the code will compile without errors. (You may *not* use other expressions like `&x` or `*x`.)

```
void f1(int x) {  x++; }
void f2(int* x) {  x++; }
void f3(int* x) { (*x)++; }
void f4(int& x) {  x++; }
void f5(int** x) { (*x)++; }
int main(int argn, char* argv[])
{
  int x;
  int* y = &x;
  int** z = &y;
  x = 5; f1(_x_); cout << x << endl;
  x = 5; f2(_y_); cout << x << endl;
  x = 5; f3(_y_); cout << x << endl;
  x = 5; f4(_x_); cout << x << endl;
  x = 5; f5(_z_); cout << x << endl;
}
```

Question 6. (5 points) What is the output of the program in the previous question? (You should assume all the spaces marked "___" have been filled in appropriately.)

5
5
6
6
5

Question 7. (a) (5 points) What is the output of the program printed below?

3
25
25

(b) (5 points) By writing directly on the code, fix all memory leaks in the program below.

```
#include <iostream>
#include <string>
using namespace std;

class A { public:
  int* x;
  A(string s) {
    x = new int;
    *x = s.length();
  }
  virtual ~A() {}
  virtual int getX() { return *x; }
};

class B : public A { public:
  int** yy;
  int z;
  B(int z) : A("confused!") {
    this->z = z;
    yy = new int*[z];
    for(int i=0; i<z; i++){
      yy[i] = new int[z];
      yy[i][0] = 25;
    }
  }
  ~B() {
    for(int i=0; i<z; i++) {
      delete [] yy[i];
    }
  }
  int getX() { return yy[0][0]; }
};

int main(int argn, char* argv[])
{
  A* a = new A("abc");
  A* b1 = new B(5);
  B* b2 = new B(5);
  cout << a->getX() << endl;
  cout << b1->getX() << endl;
  cout << b2->getX() << endl;
  delete a;
}
```

delete x;

delete [] yy;

delete b1;
delete b2;

Bonus question. *(This question is worth only two points of extra credit. Do not attempt it unless you have finished and checked your answers to all other questions.)*

Explain in words what output would be produced by the extra line of code

```
cout << y << endl;
```

at the end of the `main()` function in Question 5.

It would print the value of a memory address equal to &x + sizeof(int).