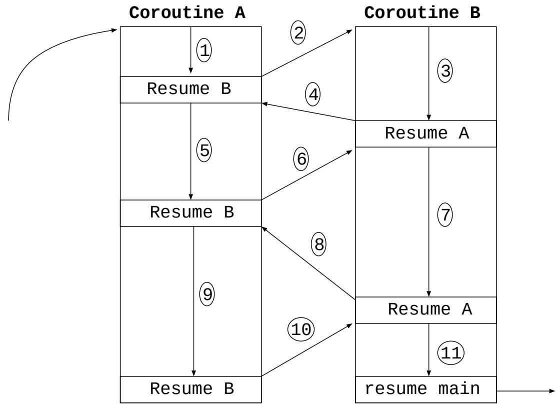
Programming Languages

Qualifiers Examination
Fall 2000

December 27, 2000

Introduction

A number of programming languages, such as *Modula 2*, introduced the notion of *coroutine*. A coroutine is a particular type of subroutine, whose execution can be repeatedly stopped and resumed. When a coroutine receives control from another subprogram, it executes partially and then it suspends and returns control. At a later point the calling program may *resume* execution of the coroutine from the point at which the execution was previously suspended. The following figure shows the flow of control between two coroutines.



Question 1 [25pts]

Describe the traditional implementation scheme used to support standard (possibly recursive) subroutines in imperative programming languages. Describe how the traditional implementation scheme used to support execution of subroutines in traditional imperative programming languages needs to be modified to support the execution of coroutines. Please use pictures and diagrams to explain your idea.

Question 2 [25pts]

Assume that you want to compile the language described earlier to C. In particular assume that you have been given access to a C compiler which provides the following two features:

- a new data type, called `Code *` which is used to declare pointers to instructions in the program;
- a predefined variable, called `cp`, of type `Code *`, which represents the current instruction pointer (i.e., it contains the address of the next instruction to be executed);
- the ability to apply the `&` operator to the name of a subroutine to obtain the address of the first instruction in the subroutine;
- the ability to perform a `goto` using as argument a variable of type `Code *`

Describe how you would compile the `resume` instructions using these novel features of C.

Question 2 [10pts]

Consider a standard imperative language with subroutines with the following syntax:

```
Program       ::=  Declarations Body
Declarations  ::=  Decl
               |   Decl Declarations
Decl          ::=  subroutine (Id) Body
Body          ::=  begin Commands end ;
Commands      ::=  Command ;
               |   Command Commands
Command       ::=  Id := Expression
               |   while Boolean do Commands endwhile
               |   resume Id
Expression    ::=  Expression + Expression
               |   Expression - Expression
               |   Identifier
               |   Numeral
Boolean       ::=  Expression = Expression
               |   Expression > Expression
```

For example, the following is a simple program according to the grammar presented:

```
subroutine A
   begin
     i := 5;
     j := 10;
     result := 0;
     resume B;
     result := result + res;
     i := result;
     j := 10;
     resume B;
     result := result + res;
     resume main;
   end
```

```
subroutine B
   begin
     res := j;
     while i > 0 do
        res := res * j;
        i := i - 1;
     endwhile
     resume A;
     while i > 0 do
        res := res * (i+j);
        i := i - 1;
     endwhile
     resume A;
   end
```

```
begin
  resume A;
end
```

In this language, the `resume` command is used to both start the execution of a coroutine as well as transfer control back and forth between coroutines. Note the use of `resume main` to terminate the coroutine and return control to the main program.

Describe (even if just at an intuitive level) how you can modify the traditional denotational semantics to accommodate for the presence of coroutines. You *do not* need to worry about error checking in the denotational semantics.