Imperial College London
Department of Computing

Compilers (221)

Exercises – Runtime Organisation

Check your answers with the tutorial helpers during tutorials and with each other on Piazza.

1. Given classes A, B and C below draw the memory layout after execution of the following sequence of assignments: L3

```
class A {
   int x
   A next
   method p(A t) { print 1; t.q(this); }
   method q(A t) { print 2; t.q(this); }
}
class B extends A {
   int y
   method q(A t) { print 3, t.q(this); }
}
class C extends B {
   int z
   method q(A t) { print 4; t.q(this); }
}

A a = new A()
B b = new B()
C c = new C()
a.next = b
b.next = c
c.next = a
```

After execution of the sequence of assignments what would `c.next.next.q(c)` print?

2. Consider a compiler and runtime system that implements immutable strings as memory blocks managed by a garbage collector. Discuss the time and memory issues that the following program might give rise to: L3

```
string line = randomChar;            // initialise line to a random character
for(int k=1; k<10000; k++) {
      line = line ++ randomChar();   // append random character to line
      process(line);                 // do something with line.
}
```

3. In some programming languages variables are not initialised by default and any attempt to read an un-initialised variable is considered to be an error. For example: L3

```
int x, y[100];    // not initialised
print x;          // error
print y[expr];    // error
```

Describe how such errors could be detected. Consider un-initialised object attributes, array elements, statically-allocated variables, stack-allocated variables and heap-allocated variables. Comment on the costs of your detection mechanism(s).

4.	In programming languages that support nested procedures, procedure parameters can be represented by a pair of values: the address of the procedure and a pointer to the appropriate stack frame. Consider the following program which has a nested procedure beta and a procedure parameter `proc`:
	`def alpha(num, proc):`
	`  def beta(): print(num)`
	`  if num==1:`
	`  alpha(2,beta)`
	`  else:`
	`  proc()`
	`def gamma(): skip`
	`alpha(1, gamma)`
	For this program, draw and explain the state of the stack when procedure `beta` is called. Explain why the output of the program is 1. Assume parameters are passed via the stack.
		L4

5.	Some OO languages have the concept of interface types and interface variables. Devise a scheme for implementing such variables and show the memory layout for the following:
	`interface F {`
	`  method p()`
	`  method q()`
	`  method r()`
	`}`
	`class A implements F {`
	`  int x`
	`  method m() { ... }`
	`  method n() { ... }`
	`  method p() { ... }`
	`  method q() { ... }`
	`  method r() { ... }`
	`}`
	`class B extends A implements F {`
	`  int y`
	`  int z`
	`  method o() { ... }`
	`  method p() { ... }`
	`  method q() { ... }`
	`  method r() { ... }`
	`}`
	`A a = new A()`
	`B b = new B()`
	`F f = a`
		L5

6.	Continuing the previous question now translate the following statements into Intel IA-32 assembly language. Assume that the addresses of a, b and f are already in registers `eax`, `ebx`, `ecx`:
	`f.r()`
	`f = b`
	`a = (A) f`
		L3

7.	Some languages allow the programmer to omit variable declarations entirely. Other languages require the programmer to declare all variables, but not to declare their types. Still other languages require the programmer to declare both variables and their types. Give a short argument in favour of each approach. Which argument do you find most convincing? Why?	L3