Objectives

The purpose of this homework is to translate a given source of C- - to the appropriate assembly looking intermediate representation. The translation will be built by traversing over the abstract syntax tree in the same way that we performed the semantic analysis. While the translation will assume that the given source is semantically correct the semantic analysis component will be important and it will be executed prior to translation. This is due to the fact that in translation we will take advantage of the annotations of the abstract syntax tree that we have achieved during the semantic analysis phase. Each basic node class will provide the implementation of two methods IRbuildstatic and IRgenerate that will facilitate the translation. The translation, as in the case of analysis, will occur in two stages, the static data building stage (that is rather short as it doesn’t need to traverse the whole abstract syntax tree) and the IR generation phase. In the generation phase the code will be synthesized in a bottom up fashion in the form of basic code blocks.

Handout

To obtain your assignment, go to the class website in the handouts section and download hw5-packet.zip. Similar to previous homeworks, this archive is a template project that you can import into Eclipse. It contains support code and basic structure. It is up to you to “fill in” the blanks. It includes plenty of support code as well as the C- -grammar. The project now includes two new packages called ir and irgen that contain a number of support classes. In particular ir contains a number of classes that extend an abstract class Instruction. All IR instructions are formed with members of the ir package.

The irgen package on the other hand contains a collection of support classes that will be used in IR code generation. In particular it contains the abstract class Location and is used for the operands of ir instructions.

The class Location is extended in a number of classes including Temporary, Immediate, RelativeSt, RelativeCl, Indexed and Label. When locations are used as operands their semantics are as follows:

- **Temporary** locations : think of them as CPU registers. You are allowed an unlimited number of them for this homework.

- **Immediate** locations : they are constants.

- **RelativeSt** locations : they provide a setOffset() method and they are offset locations from a memory location signified as [fp + offset] (this notation will denote the contents of that memory location). In practice fp is the frame pointer.

- **RelativeCl** locations : they provide a setOffset() method and they are offset locations from a memory location signified as [self + offset] (this notation will denote the contents of that memory location). In practice self is the memory address where the current object is stored.
Indexed locations: they are composite locations of the general form base (offset) where base is a location that signifies the base address and offset an offset to be added. Note that you may have compositions of such locations as in [self + 4](8) which means the address that is found after you add 8 to the contents of the address self +4.

Label locations: these are labels used for branching and beginning of methods.

The IR generation will be building an object Block that will contain a vector of instructions. This will be the main output of your translation engine. Note that the block of instructions will be attached to a class object called Code that may contain many code blocks. At present it may contain at most two, the translated program and the block required to spawn the main method of C++. program.

The convention regarding the main method is that it should take no arguments and should always return type int. No system calls are implemented whatsoever (such as printing for example) except a call to malloc that allocates memory for the objects as we need it.

One of the delicate aspects of this homework is to handle the labels when composing instruction vectors that use branching (such as for, while blocks etc). In all IR generation there is a label called firstlabel that is flowing down as an inherited attribute during your AST traversal. Whenever you can find a place to put it you will have to assign it to a certain instruction. When this is not possible you will simply pass it back upwards using the exitpoint attribute of the block you are returning.

Compared to hw4, the package sem has been upgraded from the previous version of hw4 to account for the additional functionality that is required at this stage.

To Do

Inside the package ast you will find all abstract syntax tree class that are referenced in parser.cup. In most classes the instantiation of the method IRbuildstatic and IRgenerate is missing; you will have to provide these methods. I have provided some comments inside each file that explain how you can solve them.

In the zip file you will also find a collection of 16 test files named test1.cmm to test16.cmm written in C++. These have to be translated correctly as shown in the next pages.

Remark 1. Feel free to toss away the grammar or any code that I provide to you and substitute it for the one you have built from homeworks #3-#4. Only minor modifications to your code will be required to do this.

Remark 2. There are a number of additional things one can do here and there to optimize the generation of the IR code. Any documented improvements will receive extra credit.

Hand-in

Your solution should be an archive of the whole project (that should compile correctly and perform the translation as good as you can for as many of the test files as possible) and should include the following:

1. All the classes of the package ast appropriately completed.

You submit your homework by midnight on the due date as a zip archive exported from eclipse. The e-mail address to submit is c244fa06@cse.uconn.edu.

Have fun!
In this section you can find the compilation of all 16 assigned test C- - files. Note that a solution that slightly deviates from those examples can still be correct.

parsing [test1.cmm]
D0:  
L1
L2
L3

L0:  
param 8
call malloc
get return T5
mov 0(T5),D0
param T5
invoke (T5,2,0)
get return T6
set return T6
ret

L1:  
prologue
set return [fp+4]
epilogue

L2:  
prologue
neg T0,[fp-8]
add T1,[fp-8],1
mul T2,12,T1
add T3,T0,T2
mov [fp+4],T3
set return [fp+4]
epilogue

L3:  
prologue
param 17
invoke (self,1,1)
get return T4
set return T4
epilogue

*******************************************************************************

parsing [test2.cmm]
D0:  
L1

L1:  
prologue
and T0,0,1
or T1,1,T0
not T2,T1
mov [fp-12],T2
epilogue

*******************************************************************************

parsing [test3.cmm]
D0:  
L1
D1:  
L2

L1:  
prologue
param 16
call malloc
get return T0
mov 0(T0),D1
param T0
invoke (T0,0,0)
mov [self+4],T0
mov [fp-8],[self+4](12)
epilogue
L2: prologue
mov [self+4],1
mov [self+8],0
mov [self+12],3
epilogue

**********************************************
parsing [test4.cmm]
D0: L1
D1: L2

L1: prologue
mov [fp-8],3
param 16
call malloc
get return T1
mov 0(T1),D1
param [fp-8]
add T0, [fp-8],1
param T0
param T1
invoke (T1,0,2)
mov [self+4],T1
mov [fp-8],[self+4](12)
epilogue
L2: prologue
add T2, [fp+4],[fp+8]
mov [self+8],T2
epilogue

**********************************************
parsing [test5.cmm]
D0: L1
D1: L2

L1: prologue
mov [fp-8],3
param 12
call malloc
get return T1
mov 0(T1),D1
param [fp-8]
add T0, [fp-8],1
param T0
param T1
invoke (T1,0,2)
mov [self+4],T1
param 13
invoke ([self+4],1,1)
get return T2
mov [fp-8],T2
param 14
param 15
param 16
invoke ([self+4],2,3)
get return T3
mov [fp-8],T3
epilogue

L2: prologue
add T4,[fp+4],[fp+8]
mov [self+8],T4
epilogue

L3: prologue
set return 2
epilogue

L4: prologue
set return 3
epilogue

**********************************************
parsing [test6.cmm]
D0: L1

L1: prologue
add T0,1,[fp+4]
mov [self+4],T0
add T1,2,[fp+8]
sub T2,T1,[self+4]
mul T2,T2,4
add T3,T2,8
param T3
call malloc
get return T4
mov 0(T4),T2
mov 4(T4),[self+4]
mov [fp-8],T4
sub T5,2,[fp-8](4)
mul T5,T5,4
add T5,T5,8
set return [fp-8](T5)
epilogue

**********************************************
parsing [test7.cmm]
D0: L1

L1: prologue
mov [fp-8],13
add [fp-8],[fp-8],1
sub [fp-8],[fp-8],1
set return [fp-8]
epilogue

**********************************************
parsing [test8.cmm]
D0: L1
    L2

L1:  prologue
    mov [fp-8],1
    set return [fp-8]
    epilogue

L2:  prologue
    jumpz [fp+4],L3
    mov [fp-8],1
    jump L4

L3:  mov [fp+4],1
    add [fp-8],[fp-8],1

L4:  set return [fp-8]
    epilogue

**********************************************
parsing [test9.cmm]
D0: L1

L1:  prologue
    mov [self+4],2
    mov [fp-8],0

L2:  lt T0,[fp-8],[fp+4]
    jumpz T0,L3
    mul T1,[self+4],2
    mov [self+4],T1
    add [fp-8],[fp-8],1
    jump L2

L3:  set return [self+4]
    epilogue

**********************************************
parsing [test10.cmm]
D0: L1

L1:  prologue
    mov [self+4],2
    mov [fp-8],0

L2:  lt T0,[fp-8],[fp+4]
    jumpz T0,L3
    mul T1,[self+4],2
    mov [self+4],T1
    add [fp-8],[fp-8],1
    jump L2

L3:  set return [self+4]
    epilogue

**********************************************
parsing [test12.cmm]
D0: L1
D1: L1
    L2
D2: L2
    L1
    L3


L1:   prologue
    mov [self+4],0
epilogue
L2:   prologue
    mov [self+8],0
epilogue
L3:   prologue
    mov [self+12],0
epilogue
******************************************************************************
parsing [test13.cmm]
D0:   L1
D1:   L1
   L2
D2:   L2
   L1
   L3
L1:   prologue
    mov [self+4],1
epilogue
L2:   prologue
    mov [self+4],1
    mov [self+8],2
epilogue
L3:   prologue
    mov [self+4],1
    mov [self+8],2
    mov [self+12],3
epilogue
******************************************************************************
parsing [test14.cmm]
D0:   L1
D1:   L1
   L2
D2:   L2
   L1
   L3
   L4
L1:   prologue
    mov [self+4],1
    param 20
    call malloc
    get return T0
    mov 0(T0),D2
    param T0
    invoke (T0,2,0)
    mov [self+8],T0
    param 7
    invoke ([self+8],3,1)
    get return T1
    mov [self+4],T1
epilogue
L2: prologue
  mov [self+4],1
  mov [self+12],2
epilogue
L3: prologue
  mov [self+16],156
epilogue
L4: prologue
  mov [self+4],1
  mov [self+12],2
  mov [self+16],[fp+4]
epilogue
*

parsing [test15.cmm]
D0: L1
  L2
  L3

L1: prologue
  mov [fp-8],11
  eql T0,[self+4],1
  jumpz T0,L4
  mov [fp-12],17
  eql T1,[fp-12],2
  jumpz T1,L4
  mov [fp-16],19
L4: set return 1
  epilogue
L2: prologue
  param 13
  invoke (self,0,1)
  get return T2
  mov [fp+4],T2
  epilogue
L3: prologue
  mov [self+4],0
  epilogue

L0: param 8
  call malloc
  get return T3
  mov 0(T3),D0
  param T3
  invoke (T3,2,0)
  get return T4
  set return T4
  ret
*

parsing [test11.cmm]
D0: L1

L1: prologue
  mov [self+4],2
mov [fp-8],0
L2:  lt T0,[fp-8],[fp+4]
    jumpz T0,L3
    mov [self+4],11
    add [fp-8],[fp-8],1
    jump L2
L3:  gt T1,[fp-8],[fp+4]
    jumpz T1,L4
    mov [self+4],12
    jump L3
L4:  mov [fp-8],0
L5:  lt T2,[fp-8],[fp+4]
    jumpz T2,L6
    mov [self+4],13
    add [fp-8],[fp-8],1
    jump L5
L6:  set return [self+4]
    epilogue
***********************************************************************
parsing [test16.cmm]
D0:  L1
L1:  prologue
    mov [fp-8],11
L2:  lt T0,[fp-12],11
    jumpz T0,L9
    mov [fp-16],17
L3:  gt T1,[fp-16],12
    jumpz T1,L8
    mov [fp+4],0
L4:  lt T2,[fp+4],7
    jumpz T2,L7
    eql T3,[self+4],278
    jumpz T3,L5
    mov [self+4],1
    jump L6
L5:  mov [self+4],2
L6:  add [fp+4],[fp+4],1
    jump L4
L7:  sub [fp-16],[fp-16],1
    jump L3
L8:  jump L2
L9:  set return 134
    epilogue
***********************************************************************