Name   : Raj Deepaknath Patil
Roll no : CS18BTECH11039

REPORT : PROGRAMMING ASSIGNMENT 0 :
        : TOY COOL PROGRAMS

Comments have been provided wherever necessary in the program itself
(especially for the incorrect programs)
This report deals with the correlation between the generated mips code and the cool source for the five trivial programs

- this is going to be more of an observational style report where I present correlated code segments and my hypotheses

The inheritance tree for all classes in the programs roots back to the Object class.
so all the fundamental classes (Int,String,Bool,IO) have atleast 3 features(a feature can be an attribute or a procedure) :

- Object.abort
- Object.type_name
- Object.copy

which are further followed by their own specific features which occur in the order of classes in the inheritance tree
For instance, in fact.cl:
The Main class's jump to identifier tag is stated as :

```
        Main_dispTab:
                .word   Object.abort
                .word   Object.type_name
                .word   Object.copy
                .word   Main.fact
                .word   Main.main
```

note that .word, .byte , .half and so on refer to the storage pattern in the memory: 32, 8 and 16 bits of data allocated for that particular feature. In this case 32 bits are allocated for the identifier of that method of the object hence this can be thought of storing the pointer to that function.
.ascii,.align also work along those lines(store info regarding storage)

An interesting thing to notice is that (as mentioned in the manual) Integers are strings of digits. Just for the sake of it, my final return in main in fib.cl is 12345 and the corresponding int_const from fib.s is as follows:

```
        int_const2:
                .word   2
                .word   4
                .word   Int_dispTab
                .word   12345
                .word   -1
```

this means that each of those single digits takes 32 bits each and not stored in the conventional c style of integers : simple but wasteful

Repeating this for string return ..
doing so in the case of bool_func.cl..: the return is "12345" and the corresponding

string literal in bool_func.s is as follows :

```
        str_const1:
                .word   4
                .word   6
                .word   String_dispTab
                .word   int_const7
                .ascii  "12345"
                .byte   0
                .align  2
                .word   -1
```

note the .ascii storage and not .word : each digit takes 8 bits(corresponding ascii's range of 0 to 255)

Note the global tags at the beginning of every assembly source :

```
        .data
        .align  2
        .globl  class_nameTab
        .globl  Main_protObj
        .globl  Int_protObj
        .globl  String_protObj
        .globl  bool_const0
        .globl  bool_const1
        .globl  _int_tag
        .globl  _bool_tag
        .globl  _string_tag
```

.globl is assembler directive that indicates the assembler that those symbols can be accessed from outside files as well

Proceeding to the general structure of a program:

- first the section of the with the .globl directives is present
- then seem to come some rudimentary routines that are used by the mips simulator during run time which are used to manage memory such as:
    - _MemMgr_INITIALIZER
    - _MemMgr_COLLECTOR
    - _MemMgr_TEST
    * these have a .globl directive to the same identifiers and probably refer to existing sub routines in the Mips simulator and have to be called external(hence .globl)
- then follow the string, int and bool constants (in that specific order). String constants include the usual strings explicitly mentioned in the program as well as identifiers (names of objects and classes) but not keywords.
- then, there is a class_nameTab which contains the pointers to the string constants to the names of all the declared and fundamental classes of the program. For instance (from indexed_string.s):

```
        class_nameTab:
                .word   str_const6
                .word   str_const7
                .word   str_const8
                .word   str_const9
                .word   str_const10
                .word   str_const11
```

these constants correspond to "Object","IO","Int","Bool","String" and "Main" in that specific order.
if you examine bool_func.s, there should be one more const corresponding to the BoolFunc class other than these fundamental classes as seen here:

```
        str_const8:
                .word   4
                .word   7
                .word   String_dispTab
                .word   int_const2
                .ascii  "BoolFunc"
                .byte   0
                .align  2
                .word   -1
        and ..

        class_nameTab:
                .word   str_const3
                .word   str_const4
                .word   str_const5
                .word   str_const6
                .word   str_const7
                        .WORD   STR_CONST8
                .word   str_const9
```

just as predicted, note the extra literal in this case

- then follows a tag for the objects (there can be multiple objects for one class as observable here)

```
        class_objTab:
                .word   Object_protObj
                .word   Object_init
                .word   IO_protObj
                .word   IO_init
                .word   Int_protObj
                .word   Int_init
                .word   Bool_protObj
                .word   Bool_init
                .word   String_protObj
                .word   String_init
                .word   BoolFunc_protObj
                .word   BoolFunc_init
                .word   Main_protObj
                .word   Main_init
```

* all of them in 32 bits so act like pointers to the

- then dispTabs for classes than contain that point the individual class's feature tags
- then the protObj that occur in the above class_objTab(already discussed before)
- and finally, now follows the core code, everything before this was about locating the correct feature, class info and so on..

For instance, here is the logicalAnd feature from BoolFunc

```
        BoolFunc.logicalAnd:
                addiu   $sp $sp -16
                sw      $fp 16($sp)
                sw      $s0 12($sp)
                sw      $ra 8($sp)
                addiu   $fp $sp 4
                move    $s0 $a0
                lw      $s1 20($fp)
                la      $t2 bool_const1
                move    $t1 $s1
                la      $a0 bool_const1
                beq     $t1 $t2 label2
                la      $a1 bool_const0
                jal     equality_test
```

these are mips assembly instructions doing the actual manipulation of registers(simulated in spim) and performing the computation

END OF REPORT