TDT4205 Problem Set 4
Spring 2012

PART 1 - Theory due Wed Mar. 7th, 20:00.
PART 2 - Programming due Wed Mar. 14th, 20:00.

Handing in part 2 before the deadline Wed Mar. 14th will give you 10% extra credit (total of 110%). Handing it in on the following Thursday before 20:00 will result in 100% (no extra credit). Handing it in on Friday before 2pm gives 10% penalty (90% total). After 2pm, no credit will be given.

Part 1 needs to be passed in order for Part 2 to be evaluated!
ALL answers are to be submitted to *itslearning*

ALL OF THIS ASSIGNMENT IS TO BE DONE INDIVIDUALLY. Cheating (”koking”), including using partial solutions from other students, will cause a failing grade for the whole course. We will be checking for this using a plagerism detecting as well as looking for suspiciously similar submissions.

All submitted source code MUST be able to compile and run on asti. **This assignment counts towards your final grade.** Please read the assignment guidelines on itslearning before starting to work on the assignment. Requests for clarifications can be posted on the itslearning forum.

What to turn in
When turning in assignments, please turn in these files:

- (your_username)_answers.pdf : Answers to non-programming questions (Part 1)
- (your_username)_assembly.s : Solution to the assembly programming problem (Part 1)
- (your_username)_code.zip,tar.gz,tgz : All your code for this assignment, including makefiles and other necessary code (Part 2)

PART 2 - Programming: Symbol table implementation

Task 2.1 (30%)

Implement the following auxiliary functions in the provided `src/symtab.c`:

- `symtab_init`
- `symtab_finalize`
- `strings_add`
- `strings_output`
- `scope_add`
- `scope_remove`
- `symbol_insert`
- `symbol_get`

These functions provide the initialization, population, lookup and removal of `symbol_t` structures (as defined in `include/symtab.h`), linking them to syntax tree nodes through the `node_t` element `entry`.

`symtab_init` is called once before tree traversal, and should set up dynamic storage for a table of scopes, list of `symbol_t` structures, and a table of strings. Pointers, sizes and limits of these arrays are already declared, but their size will be dynamically managed. The corresponding `symtab_finalize` is called at the end of execution, to remove the tables and their contents.

`strings_add` appends a string to the table, resizing it when appropriate, and returns the index of the added string. The function `strings_output` should dump the contents of the table to a provided output stream. As the output from this function will ultimately form the initialized data segment of our generated code, it should adhere to the format

```
.data
.INTEGER: .string "%d "
.STRING0: .string "Hello, world!"
.STRING1: .string "Some other string..."
(etc.)
.globl main
```

that is, the first two and last lines are fixed, the lines between represent the contents of the string table, written as `.STRING<index>:  .string <the text>`.

The functions `scope_add` and `scope_remove` add and remove hash tables from a stack of tables representing the nested scopes at a given depth in the syntax tree. The functions `symbol_insert` and `symbol_get` insert and retrieve `symbol_t` structures stored in this stack, respectively.

Task 2.2 (30%)

Using the functions implemented in the previous task, implement the function `bind_names` in src/tree.c so that it recursively traverses the syntax tree, and

- populates the symbol table(s) when encountering declarations (functions, variables, string constants).
- resolves symbol references by setting the `entry` pointer when encountering uses/invocations.

The `depth` element of the `symbol_t` struct is only relevant for variable and parameter declarations, and is set to the number of scopes the declaration is nested inside.

The `stack_offset` element of the `symbol_t` struct is only relevant for variable and parameter declarations, and is computed as follows:

- *Parameters* to functions have positive, decreasing offset values. The *first* parameter has the greatest offset, the *final* parameter has offset 8, and each parameter has an offset 4 smaller than its predecessor.
- *Local variables* in functions have decreasing offsets, beginning at -4, and proceeding in multiples of -4 in the order of their declaration. You do NOT need to take arrays into consideration yet; this will be given in a later assignment. For now, treat array declarations like regular variable declarations.

The `n_args` element of the `symbol_t` struct is only relevant for function declarations, and is set to the number of formal arguments to the function. (Identifiers representing function calls can simply be linked to the resulting symbol table entry, it is not yet necessary to check that the number of arguments at invocation is correct).

The purpose of returning the index of added strings in the `strings_add` function, is that the string referred to by a `TEXT` node can be moved into the symbol table entry, and referred through the entry element. Its index in the string table will come in handy later, and can therefore replace the literal string as the `data` element of such nodes.