University of Algarve

Faculty of Sciences and technology

Masters in Informatics Engineering

DLPC

Nome: Diogo Fonseca

Número: 79858

Project “mini-c”

The mini-c programming language was made to resemble the C programming language, however, the biggest difference is it’s aggressive implicit promotion/conversion of values.

Algebraic expressions

The programming language can perform the common mathematical operations, such as division, multiplication, addition and subtraction (“/”, “*”, “+” and “-” respectufully).

Variable declaration

Furthermore, the language employs strong explicit typing, with global (outside of functions) and local (inside functions) variables. Meaning that a type must be explicitly stated at variable initialization. The semantics are close to equal to C's, expliciting the type, the name, and then equaling it to an expression followed by a semicolon.

Printing

The language benefits of a special printing function "print()" which will print it's argument expression correctly regardless of type.

Functions

Functions have any number of arguments (as they are stack-based), which will act as local variables for the function, when calling a function in an expression (or in an instruction by itself) the compiler will guarantee that the arguments given and the arguments at the function declaration match up. This does not include type checking against the declaration due to the aggressive promotion system employed (on purpose), which was one of the focuses of the compiler. Variables within a scope have priority over variables with a lower scope. Furthermore the program has to have a main function in order to compile without errors. Return statements are also mandatory, except for void functions. A special care was employed in order to maintain stack alignment throughout the whole process.

Type system

The types that exist are:

- **int** – a signed integer of 32 bits.
- **long** – a signed integer of 64 bits.
- **float** – a single precision floating point of 32 bits
- **double** – a double precision floating point of 64 bits

The type system, before compiling, checks all types and annotates the tree with the types of expressions (which can be functions, arithmetic operations, or even variables, …)
Because of this type annotation, on compile-time, an aggressive promotion/demotion system is employed to the variables.

Promotion system

The promotion system employs the following hierarchy:

int32 → int64 → float32 → float64

in an expression, when two of these types must forego an joint operation between the two, the lower on the hierarchy will convert to the higher on the hierarchy. As an examlple, if we were to multiply an int32 by a float32, the int32 would be converted to a float32 right before multiplication. The same applies when passing arguments to a function.

Demotion system

The demotion system is employed only when either declaring a variable or assigning a new value to a variable. The expression, no matter the type, will lawfully obey the type of the variable, and always convert to the variable's type after it has it's final value. It always truncates whenever need be, trying to preserve sign. Every conversion was carefully crafted and tested. The promotion and demotion systems are a defining (and perhaps dangerous) part of this language.

Compiling errors

The compiler will exit compilation and throw a descriptive error whenever the code isn’t up to the rules of the language, many errors were searched for, such as not having a main function, not having a return on a non-void function, wrong parameters, etc…

Running

To run the program, ocaml packages (pam) must be installed, as well as ocamellex and menhir. Other than that, when running “make”, the “test.minic” program will be compiled into “test.s” and then executed.

Conclusions

There was also some minor optimizations to the assembly, but nothing too great, if the assembly is to be inspected there are a lot of potential optimizations to be had.
The language also lacks if statements, which makes the language not turing complete. Furthermore, it doesn’t have loops either, which would have been a major helping hand after the language had if statements.
Both if statements and loops were sadly not implemented due to a lack of time.