“simpliCty” Project Proposal

Course: COMS W4115
Date: July 11, 2016

Group members:
Rui Gu - rg2970
Adam Hadar - anh2130
Zachary Moffitt - znm2104
Suzanna Schmeelk - ss4648

The language we propose to implement

We propose to implement a simplified version of C, which contains a subset of C grammar with a strict type system and uses LLVM as the backend to produce bytecode.

The sorts of programs meant to be written in our language

Our language operates best with programs that need to be Turing complete and can be defined using strict type casting and only stack-based memory management, which decreases runtime errors. The C language domain is for number crunching and embedded systems. Our programs will operate in the same domain as C with the exception that our compiler supports only a limited features of C.

Parts of our language and what they do

Supporting a subset of C grammar

Primitive types will be supported (int, float, bool, char). Pointers will not be supported (see implementation section). ‘while’, ‘for’, ‘if-else’, and ‘return’ are supported; other control flow statements are not. Function calls and recursion will also be supported. Nested functions are not allowed. Only local scope will be supported. We will not allow global variables or static variables. We will not support heap memory allocations as everything is located on the stack. We will also not support objects and structs/unions.

Strict type programming

No automatic type conversion and no type inference. Every statement will have a decided type information before actually running the program. No uninitialized variables are allowed. We don’t provide default values to primitive types.

Implementation

We will build our own compiler frontend by using OCaml. The generated AST will be hooked with LLVM APIs to construction LLVM IR. Then we utilize LLVM to generate machine dependent assembly code. We compiler solely relies on stack for memory management. Variables exist only within the scope of the function where the variables are created which requires the developer to handle the return value. By definition; when the function ends, all the allocated memory will be reclaimed.

Source Code for an Interesting Program (*Fibonacci.sct*)

```
#include stdio.sct               // includes print() and scan()
#include cast.sct                // includes int_to_string()

start()                          // control flow always begins at start()
{
    int n = 0;                   // each variable must init on separate
    int first = 0;               // lines
    int second = 1;
    int next = 0;
    int c = 0;
    txt inputReq = “Enter number of terms:\n”;

    print(inputReq);
    n = scan();

    txt output = “The first ”.int_to_string(n).” terms of Fibonacci:\n”;
    print(output);


    for (c = 0; c < n; c = c + 1)
    {
        if (c <= 1)                     // all ‘if’s must have braces
        {
            next = c;
        }
        else
        {
            next = first + second;
            first = second;
            second = next;
        }
        txt number = int_to_string(next).”\n”;
        print(number);
    }
}                                      // start() does not return a value
```