Assignment 3
ICS|E 141 & Inf4mtx 101: Programming Languages

Guidelines

Semantic translation is the last phase in our study of programming language basics. SIMPLESEM is the example architecture we will use for our study. Recall that SIMPLESEM is a very basic processor based on the Von Neumann model of the fetch-execute cycle. SIMPLESEM was designed to give very practical and real experience translating the semantics of high-level programming languages to a simple, but powerful, processor. Virtually every programming language implementer (whether the language is imperative, logical, or functional in nature) must map the semantics of the source language onto a Von Neumann machine. Implementing semantic translation helps you understand how a compiler writer & compiler complete the translation of high level programming language mechanisms into a series of low level instructions.

Assignment

Note: for this assignment you will create and submit several .txt files. When asked to write a complete program, use the same file format as the Program<#>.S files in previous assignments.

1. SIMPLESEM Template (15 points)

Create a SIMPLESEM code- template (similar to the if/while constructs we created in class) for a switch-case statement. Place your implementation in a file named: **`Template.txt`**

2. Implementing C1 semantics in Simplesem (20 points)

Write a complete SIMPLESEM program for the following C1 program. Place the implementation in a file named: **C1.txt**

```
int a = 3, b = 1, c = -1;
main()
{
 //Note: you must issue explicit instructions
 //for all initializations
  while( a >  c)
  {
    if (a == 0)
    {
       print(b);
    }
    else
    {
       b = b+a;
    }
    a = a - 1;

  }
  print(a,b,c);
}
```

3. Implementing C2 semantics in Simplesem (20 points)

(a) Write a complete SIMPLESEM code for the following program using the C2 language paradigm. Place the implementation in a file named: **`C2.txt`**

```
int n=0, m=0; //Note: you must issue explicit
              //instructions for all initializations
gcd()
{
   while( m!=n)
   {
     if(n>m)
       n = n-m;
     else
     m = m-n;
   }
}
main()
{
  get(n, m);
  gcd( );
  print(n);
}
```

4. Implementing C3 semantics in Simplesem (20 points)

Write a complete SIMPLESEM program for the code below using the C3 language paradigm. Place the implementation in a file named: **`C3.txt`**

```
int global;
int c3()
{
  int n = global;
  global = global - 1;
  if (n == 1)
    return 1;
  else
    return 2*c3() + 1;
}
   main(){
       get(global);
       print(c3());
  }
```

5. C3 with Parameter Passing Semantics (20 points)

Assuming we were to extend the C3 programming language to allow the passing of parameters, how would the call/return sequence change?

Perform the following:

1) Give a template call/return sequence
2) Write a complete program using your templated call/return sequence for each of the parameter passing methodologies listed below:
   A) pass-by-value
   B) pass-by-reference

Place the implementation in the files named: **`C3P_template.txt, C3P_reference.txt, and C3P_value.txt`**

```
int n;

int fib(int n)
{
   int local;

   if (n <= 2) return 1;
   else
   {
      local = fib(n-2);
      return fib(n-1) + local;
   }
}

main()
{
   get(n);
   fib(n);
}
```