SOOP

SOOP is the brainchild of Maggie Johnson and Steve Freund.

In this course, we will write a compiler for a simple object-oriented programming language called SOOP. By design, it uses syntax that should be familiar from C/C++/Java with a few tweaks, and keeps things simple by sticking to the basics rather than adopting all of the features of those languages. It has function declarations and definitions, as well as standard assignment statements and expressions, while loops, if-statements, and print statements. The supported data types are int, boolean, double, arrays, and classes with single inheritance. Classes have instance variables and methods, along with primitive access control. Global and local scoping is also supported. The target language for the SOOP compiler is Chowder, which is assembly language code for the SPIM Machine, a MIPS R2000/R3000 simulator (copyrighted by James R. Larus).

We will provide more precise information about the language as we progress through the programming projects. For now, to give you a taste of SOOP, here are some sample programs for your perusal:

The obligatory first "hello world" program

```
void main(void)
{
   Print("hello world");
}
```

SOOP programs start execution at the global function `main()` which takes no arguments. `Print()` is the name of the built-in output function in SOOP. It is a globally accessible function and accepts any number of arguments of type integer or string. SOOP is case-sensitive.

Some arithmetic

```
int a;            // global variable

void main(void)
{
   int b;         // local variables
   int a;
   double d;

   b = 3;
   a = b * 10 + 2;
   d = 0.3/3.5;
   Print(a, b);
}
```

The standard binary arithmetic and relational operators exist in SOOP with the usual associativity and precedence. For simplicity, SOOP does not allow co-mingling of types (i.e. the two operands must both be integers or both doubles).

A loop

```
void main(void)
{
   int arr[10];
   int i;

   i = 0;
   arr = NewArray(arr);

   while (i != 10) {
      arr[i] = i;
      i = i + 1;
   }
}
```

Arrays are a little quirky in SOOP. You must state the dimensions when declaring an array and must allocate space using the built-in `NewArray` function. They are indexed in the usual 0-based manner. Multi-dimensioned arrays are supported.

A function definition and call

```
int binky(int a, double b, bool c); // SOOP has a boolean type built-in

int binky(int a, double b, bool c)
{
   if (b > 0.0 && c)
      return a + 2;
   else
      return a % 2;
}

void main(void)
{
   int a;
   double d;

   a = 3;
   d = 0.15;
   binky(a * 2, d, true);
}
```

Functions must be declared before they are defined or called.

A simple class

```
class Cow {
   int height;
   int weight;
   void Moo(void) {
      Print(this.height, " ", this.weight, "\n" );
   }
}

void main(void) {
   class Cow betsy;
   betsy = New(Cow);
   betsy.weight = 100;
   betsy.height = 122;
   betsy.Moo();
}
```

Class names must always be preceded by the reserved word `class` when used as type (e.g. `Cow` by itself is not valid). Each instance variable/method can have an optional access specifier for `public` or `private`. The default is `public`. All methods are dynamically bound (virtual).

More classes, with inheritance

```
class Animal {
   int height;
   class Animal mother;

   void InitAnimal(int h, class Animal mom) {
      this.height = h;
      this.mother = mom;
   }

   int GetHeight(void) {
      return this.height;
   }

   class Animal GetMom(void) {
      return this.mother;
   }
}

class Cow : Animal {
   int spots;

   void InitCow(int h, class Animal m, int spot) {
      this.spots = spot;
      this.InitAnimal(h,m);
   }

   int GetSpots (void) {
      return this.spots;
   }
}
```

```
void main(void)
{
   class Cow betsy;
   class Animal b;
   betsy = New(Cow);

   betsy.InitCow(5, NULL, 1223);
   b = betsy;
   b.GetMom();
   Print("spots: ", betsy.GetSpots(), "    height: ", b.GetHeight());
}
```