La Guía definitiva de Mini Pascal

IS744 Compiladores - Primer Semestre 2014

Última actualización: Febrero 6, 2013

Mini Pascal es el lenguaje que debe ser compilado en IS744. El nombre en realidad no significa gran cosa, excepto que, bueno, tiene la palabra PASCAL en ella y va derecho al corazón de todos los lenguajes de programación.

El lenguaje es en gran parte derivado de Pascal y lenguajes similares. Sin embargo, se ha simplificado enormemente para hacer su vida un poco más fácil. Por ejemplo, no hay punteros, registros, arreglos multi-dimensionales u objetos. Del mismo modo, el lenguaje es un poco escaso a la hora de características avanzadas. Sin embargo, usted encontrará que es lo suficientemente difícil para un proyecto de 14 semanas.

Su tarea es escribir un compilador para el lenguaje Mini Pascal. Su compilador leerá programas fuentes de Mini Pascal ".pas" y producirá código ensamblador para x86, MIPS, LLVM o SPARC que pueda ser compilado y ejecutado en una estaciones de trabajo. Su primer objetivo es producir un compilador que funcione. Sin embargo, los compiladores que generen código objeto optimizado no se recompensarán.

Un programa de ejemplo Mini Pascal

Este es un programa muy simple en Mini Pascal:

```
fun quicksort(l:int, r:int, a:int[8192])
    i:int;
    j:int;
    x:int;
    w:int;
    tmp:int;
    done:int;
begin
    i := l;
    j := r;
    x := a[(l+r)/2];
    done := 0;
    while done == 0 do
        begin
            while a[i] < x do
                i := i + 1;

            while x < a[j] do
                j := j - 1;

            if i <= j then
                begin
                    tmp := a[i];
                    a[i] := a[j];
                    a[j] := tmp;
                    i:=i+1;
```

```
                    j:=j-1
                end;

            if i>j then
                done := 1
        end;

    if l<j then
        tmp := quicksort(l, j, a);

    if i<r then
        tmp := quicksort(i, r, a)
end

fun main()
   v:int[8192];
   i:int;
   n:int;
begin
   print("Entre n: ");
   read(n);
   i := 0;
   while i<n do
     begin
        read(v[i]);
        i := i+1
     end;
   quicksort(0, n-1, v);
   i := 0;
   while i<n-1 do
     begin
        write(v[i]); print(" ");
        if 0 < v[i] - v[i+1] then
        begin
           print("Quicksort falló "); write(i); print("\n") ; return(0)
        end
        else
           i:=i+1
     end;
   write(v[i]);
   print("Éxito\n")
end
```

Estructura del Programa

Un programa Mini Pascal consiste de una o más declaraciones de función, como se muestra en el ejemplo anterior. La forma general de una declaración de función es la siguiente:

```
fun name(arguments) locals begin statements end
```

La ejecución siempre inicia en una función llamada `main`. Esta función no tiene argumentos y deberá devolver un número entero para indicar el éxito o el fracaso. Si a un programa no se le define `main`, se debe generar un error.

Declaraciones

Dentro del cuerpo de una función, pueden aparecer los siguientes tipos de declaraciones:

- `while` *`relation`* `do` *`statement`*
- `if` *`relation`* `then` *`statement`*
- `if` *`relation`* `then` *`statement`* `else` *`statement`*
- *`location`* `:=` *`expression`*
- `print(`*`literal`*`)`
- `write(`*`expression`*`)`
- `read(`*`location`*`)`
- `return` *`expression`*
- *`name`*`(`*`exprlist`*`)`
- `skip`
- `break`
- `begin` *`statements`* `end`

En cualquier lugar se espera una declaración, múltiples declaraciones pueden ser usadas siempre que estén encerradas en un bloque `begin` ... `end` y separadas por punto y coma como esto:

```
begin
   statement1;
   statement2;
   ...
   statementn
end
```

Es importante señalar que un punto y coma (;) no se utiliza en la última declaración de un bloque. Esto es porque el punto y coma se utiliza solamente para separar múltiples declaraciones y no es un finalizador de comando.

El comando `skip` se usa para denotar una declaración vacía. No hace nada. Por ejemplo, un bucle infinito, se escribe así:

```
while 1==1 do skip
```

El comando `break` se usa para salir de un bucle while. Por ejemplo:

```
while expr do
   begin
     ...
     ...
     break
     ...
   end
```

Tipos de Datos

Mini Pascal solo admite dos tipos de datos básicos

- `int` - Enteros
- `float` - Números de punto flotante

Además, los arreglos de tamaño fijo pueden especificarse como sigue:

- `int[`*`size`*`]`
- `float[`*`size`*`]`

Los arreglos son siempre limitados y es ilegal para un programa el acceso a un elemento que está fuera de rango. Si esto ocurre, un programa Mini Pascal debe generar un error en tiempo-ejecución.

Por el momento, Mini Pascal no tiene soporte para caracteres o cadenas. Por supuesto, algo relacionado con su aplicación podría hacer una buena pregunta de examen.

Asignación

La asignación a una ubicación se hace de la siguiente manera:

- *`name`* `:=` *`expression`*
- *`name`*`[`*`index`*`]` `:=` *`expression`*

El lado izquierdo o bien puede ser el nombre de una variable o una entrada a un arreglo. Los índices del arreglo deben ser enteros. Las expresiones deben ser del mismo tipo de la variable del lado izquierdo. De lo contrario, un error de tipo debe ser generado.

Expresiones

Las siguientes operaciones aritméticas son soportadas por Mini Pascal:

- *`expr`* `+` *`expr`*
- *`expr`* `-` *`expr`*
- *`expr`* `*` *`expr`*
- *`expr`* `/` *`expr`*
- `-` *`expr`*
- `+` *`expr`*
- `(`*`expr`*`)`
- *`name`*`(`*`exprlist`*`)`
- *`name`*
- *`name`*`[`*`expr`*`]`
- *`number`*
- `int(`*`expr`*`)`
- `float(`*`expr`*`)`

El tipo de una expresión está determinado por los tipos de sus operandos. Es ilegal mezclar operandos de diferente tipo. Sin embargo, los operadores `int()` y `float()` pueden ser usados para realizar una conversión explícita de tipo.

Un *number* es una literal que aparece en un programa y puede ser un número entero tal como "34" o un número de punto flotante como "2,8162". El tipo de un número está determinado por su sintaxis (por ejemplo, si tiene un punto decimal, debe ser punto flotante). Además, una literal de número entero puede ser promovido a punto flotante, si se utiliza como parte de una expresión de punto flotante.

Un índice de un arreglo debe ser siempre un número entero. Si se trata de otro tipo (es decir, punto flotante), un error de tipo debe ser generado.

La expresión `name(exprlist)` denota una llamada a función. En este caso, *exprlist* es una lista separada por comas como argumentos de la función, cada uno de los cuales puede ser una expresión.

Expresiones Relacionales

Las siguientes expresiones relacionales pueden ser usadas en declaraciones condicionales y en bucles while.

- `expression < expression`
- `expression <= expression`
- `expression > expression`
- `expression >= expression`
- `expression == expression`
- `expression != expression`

Además, las expresiones relacionales pueden ser combinadas usando `and`, `or` y `not` como esto:

- `relation and relation`
- `relation or relation`
- `not relation`
- `(relation)`

Por ejemplo:

```
if (n >= 2) and (n < 10) then statement
```

Funciones

Las funciones se definen usando la sintaxis

- `fun name ( arglist ) locals begin statements end`

Por ejemplo:

```
fun foo(n:int, f:float, d:int[32])
    temp1:int;
    temp2:float;
    p:int[32];
    begin
        statement1;
        statement2;
```

```
        ...
        statementn
    end
```

Cada argumento de la función debe tener un nombre y un tipo como se muestra. Múltiples argumentos son separados por comas y una función puede aceptar ningún argumento. *Locals* deben ser declaradas antes del primer `begin` y puede incluir declaraciones de variables y/o declaraciones de funciones. Cada *local* es terminado por un punto y coma.

Una característica de Mini Pascal es que soporta definiciones de funciones anidadas. Funciones anidadas son siempre definidas en la sección *locals* de una función como esto:

```
fun foo(n:int)
    i:int;
    j:int;

    /* Una función anidada */
    fun bar(x:int, y:int)
        a:int;
        begin
             ...
        end;

    /* Inicio de la función foo */
    begin
        j := bar(n,i)
        ...
    end
```

Las funciones anidadas son solamente visibles a las instrucciones de la función en la que fue definida. Es decir, la función `bar()` anterior es solamente visible dentro de la función `foo()` y no en otra parte. Además, las funciones anidadas usan alcance léxico para enlazar a nombres de variables. Esto significa que la función `bar()` anterior se le permite accesar las variables `i`,`j` y `n` que se definen para `foo()` en adición a las variables `x`, `y` y `a` que son recibidas como parámetros o definidas como variables locales. Si una función anidada redefine una variable ya utilizada en una función externa, la nueva definición es usada (por ejemplo, si `bar()` también definió una variable `i`, esa varible debe ser diferente de la variable `i` definida en `foo()`).

El tipo de retorno de una función se deduce por el tipo de la expresión pasada a la instrucción `return`. Es ilegal para el retorno de la función dos tipos diferentes desde diferentes instrucciones `return`. Por ejemplo:

```
fun foo(x:int)
    begin
        if x > 0 then
            return(3.4)
        else
            return(3)    /* Error! Tipo de retorno incoherente */
    end
```

Convenios en paso de parámetros

Cuando tipos de datos simples (int, float) se pasan como argumentos de función, siempre son pasados por valor.

Esto significa que se realiza una copia y que una función puede modificarlos sin afectar sus valores en la función de llamada. Cuando se pasan arreglos como argumentos, se pasan por referencia. Esto significa que los cambios en un arreglo se verá reflejado en la función actual como en la función de llamada.

I/O

I/O en Mini Pascal es bastante flojo. Unas pocas funciones simples de I/O son soportadas por el propio lenguaje.

- `print(`*`string`*`)`
- `write(`*`expression`*`)`
- `read(`*`location`*`)`

La instrucción `print` solamente se usa para imprimir literales en la salida estándar. Por ejemplo:

```
print("Hola Mundo\n");
print("Por favor, escriba algo:");
```

La instrucción `write` es usada para imprimir el valor de una expresión en la salida estándar. Sus argumentos pueden ser un valor de tipo entero o punto flotante.

La instrucción `read` es usada para leer un valor desde la entrada estándar dentro de una localización dada. Una localización puede ser el nombre de una variable o una ubicación en un arreglo. El funcionamiento de la recepción de entrada no válida no está definido (por ejemplo, si un usuario escribe un número de punto flotante, donde se espera un número entero).

Comentarios y Formatos

Los comentarios son denotados por texto encerrado en /* ... */. Los comentarios pueden abarcar varias líneas, pero no pueden anidarse.

Mini Pascal no es sensible a *whitespace* o *indentación*. Un programa puede ser especificado en una línea si se desea. Del mismo modo, cualquier instrucción puede abarcar múltiples líneas sin un caracter especial de continuación de línea.

Literales

Variables y nombres de funciones (identificadores) puede consistir de letras, números y el caracter guión bajo (_). Un nombre no puede empezar con un dígito. No hay límite en la longitud de un nombre.

Una literal entera consiste de uno o más dígitos y no lo precede un signo + o - (estos son manejados en las reglas de una expresión). Es ilegal que un entero contenga ceros a la izquierda. Por ejemplo, "01234" es ilegal. Por supuesto, un número entero puede tener el valor "0".

Hay varias maneras de especificar un número de punto flotante:

- 123.456

- 0.234
- 1.23e+04
- 1.45e-07
- 1.23e89
- 2e+14

en general un número de punto flotante debe de empezar con uno o más dígitos seguidos por un pubto decimal o una "e" para indicar notación científica. Si aparece más de un dígito, no se permite ceros a la izquierda (por ejemplo, "0123.45" y "0123e45" son ilegales). Después del punto decimal, uno o más dígitos pueden aparecer. Además, la notación científica e[+-]nnnnn puede ser usada como se muestra. Para la notación científica el signo es opcional y se asume que es positivo si es omitido.

Literales de cadena (*string*) son solamente usadas en instrucciones `print`. Una cadena está encerrada entre comillas dobles (") y puede contener cualquier combinación de letras, dígitos y caracteres especiales. Una cadena no puede abarcar múltiples líneas. Sin embargo, cadenas adyacentes son concatenadas para formar una sola cadena. Por ejemplo, escribir "Hola" "Mundo" es lo mismo que "HolaMundo". Además, Mini Pascal reconoce los siguientes caracteres de códigos de escape.

- \" - El caracter de doble-comilla
- \n - El caracter newline (0x0a hex)
- \\ - El caracter backslash

Varios

Mini Pascal no soporta variables globales o datos localizados en montículo--todo es almacenado en la pila. Por lo tanto, no tendrá que preocularse por la asignación dinámica de memoria, la implementación de su propia versión de malloc(), ni nada por el estilo. Sin embargo, se deberá resolver la complejidad de tener ámbitos de funciones anidadas.

Para este proyecto, los programas sólo pueden aparecer en un único archivo fuente. No necesita preocuparse por múltiples archivos o preprocesamiento.

Resumen del Proyecto

Esta es una vista general de las etapas implicadas en la realización del proyecto:

- Escriba un escáner que convierta un archivo fuente dentro de una lista de símbolos (tokens).
- Escriba una gramática formal BNF para Mini Pascal.
- Escriba un analizador (parser) que convierta esta gramática en un árbol de sintaxis abstracto (AST).
- Escriba un módulo que realice análisis semántico y comprobación de tipos.
- Producir código intermedio.
- Convertir el código intermedio a lenguaje ensamblador para x86, MIPS, LLVM o SPARC.
- Optimización de todo.
- Sentarse y mirar asombrado su compilador.