Metodología empleada y manual de operación

Diseño de compiladores
Dr. Edgar E. Vallejo

Saúl de Nova Caballero (A01165957)
Diego Galíndez Barreda (A01370815)

6 de mayo de 2016

1. Metodología empleada

A continuación viene un resumen de los cambios que se hicieron a la especificación para poder cumplir todos los requerimientos.

1.1. Expresiones regulares

Para las expresiones regulares, se convirtieron todos los operadores, tipos y palabras claves del lenguaje en tokens. También se verificaron constantes de strings, enteros en hexadecimal y enteros, flotantes en notación científica y sin la notación. Se consideraron todas las cadenas que no tengan saltos de línea como válidas. Se verificó para los identificadores que fueran a lo más de 31 caracteres. En el caso de que un caracter no sea ' ', \n, \r ó \t.

Para acomodar comentarios, se utilizó la instrucción **BEGIN** para determinar los estados de flex. Se creó el estado **C_COMMENT** para determinar cuando te encuentras dentro de un comentario.

1.2. Modificaciones a la gramática

Para manejar los bloques de código en las instrucciones adecuadas, if, for, $while$, $block$, se insertaron expresiones *scopeStart* y *scopeEnd* para manejar la profundidad de las tablas de símbolos.

Para poder tener las expresiones con $^+$ y * se descompusieron las expresiones de la siguiente manera:

$Expr ::= Term^+$
$Expr ::= Term \,|\, Expr\, Term$

se cambió a:

$Expr ::= Term^*$
$Expr ::= \epsilon \,|\, Expr\, Term$

Con el propósito de generar el código intermedio correcto indexando, arreglos y *strings*, se cambió valorL y expr. El valor original era:

$Expr ::= ValorL \mid ValorL = Expr$
$ValorL ::= identificador \mid Expr\,[\,Expr\,]$

se cambió a:

$Expr ::= ValorL \mid ValorL = Expr \mid ValorL\,[\,Expr\,] \mid ValorL\,[\,Expr\,] = Expr$
$ValorL ::= identificador$

Para poder hacer los reales y los formales, se separó la gramática en 3 para poder acomodar comas, vacíos y final sin comas.

$Reals ::= Expr^{+}\,, \mid \epsilon$

se cambió a:

$Reals ::= \epsilon \mid RealsDefEnd$
$RealsDefEnd ::= expr \mid RealsDef\, expr$
$RealsDef ::= expr\,, \mid RealsDef\, expr\,,$

Para finalizar, en las funciones se nos presentó un problema. Para poder asegurarnos que los tipos de los *returns* fueran correctos tuvimos que cambiar la gramática de esto:

$DeclFuncion ::= Tipo\, identificador\,(Formales)\, BloqueInstr$
$\mid void\, identificador\,(Formales)\, BloqueInstr$

a:

$DeclFuncion ::= TipoFuncion\,(Formales)\, BloqueInstr$
$TipoFuncion ::= tipo\, identificador \mid void\, identificador$

Este cambio puede parecer minúsculo, sin embargo, nos permite que la tabla de símbolos guarde el identificador de la función para ser usado en la función de abajo.

Para manejar los *fors* se utilizó la técnica de *backpatching* y se cambió la gramática insertando expresiones vacías, cuyo único propósito es generar código. Por lo que las expresiones:

$InstrIf ::= if\,(\,Expr\,)\, Instr < else\, Instr >$
$InstrWhile ::= while\,(\,Expr\,)\, Instr$
$InstrFor ::= for\,(< expr >;\, Expr\,;< expr >)\, Instr$

se cambiaron a:

$InstrIf ::= if\,(\,\textbf{InstrIfCond}\,)\, Instr\, \textbf{InstrIfEnd}$
$InstrIf ::= if\,(\,\textbf{InstrIfCond}\,)\, Instr\, \textbf{InstrIfElse}\, else\, Instr\, \textbf{InstrIfElseEnd}$
$InstrWhile ::= \textbf{InstrWhileStart}\, while\,(\,\textbf{InstrWhileCond}\,)\, Instr\, \textbf{InstrWhileEnd}$
$InstrFor ::= for\,(< expr >\, \textbf{InstrForStart}\,;\, \textbf{InstrForCond}\,;< expr >)$
$>\qquad\qquad \textbf{InstrForLabel}\, Instr\, \textbf{InstrForEnd}$

1.3. Tabla de símbolos

Para poder manejar adecuadamente los scopes del programa fue necesario utilizar una tabla recursiva de símbolos. Se muestra el concepto en la figura 1.

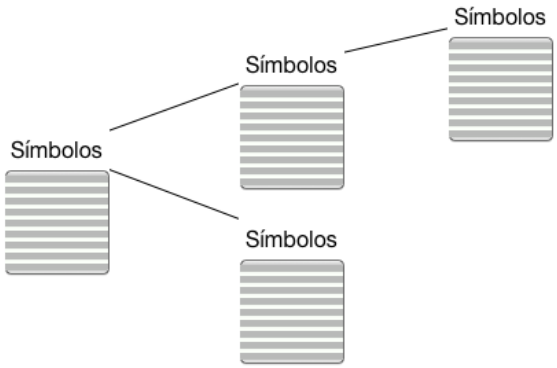

Figura 1: Tabla recursiva de símbolos

En el momento en que se cambia el scope (**instrIf**, **instrWhile**, **instrFor**, **declFunction**), se crea una nueva tabla de símbolos. Todas las tablas tienen una tabla padre, por lo que se puede regresar al scope padre.

Las tablas utilizan los hashes de los símbolos. Si hay colisiones utiliza una lista para agregar varios símbolos al mismo hash.

Los símbolos incluyen un nombre, un tipo y un contador de las veces que se accesa la variable. Tenemos una función lookup que busca recursivamente los símbolos, por lo tanto tenemos oversha-dowing.

1.4. Manejo de tipos

Para manejar los diferentes tipos que se pueden programar en cmm: *int*, *bool*, *double*, *string*[], *int*[], *bool*[], *double*[], se definieron constantes enteras para identificar los tipos.

1.5. Generación de código intermedio

Al momento de generar el código intermedio se consideraron muchas alternativas. En nuestro caso se utilizó el lenguaje intermedio de LLVM (`http://llvm.org/releases/3.6.2/docs/LangRef.html`).

1.5.1. LLVM

LLVM es una colección de herramientas que utilizan muchas empresas y lenguajes para compilar sus propios programas. Por ejemplo Apple utiliza LLVM como su infraestructura para compilar C, C++, Objective-C y Swift.

1.5.2. Runtime

Es necesario declarar un archivo que declare las implementaciones de las funciones incluidas en el lenguaje. Este archivo implementa las funciones **printInt**, **printDouble**, **printString** que utiliza internamente la función **print**. El archivo está escrito en el lenguaje intermedio.

1.6. Arquitectura

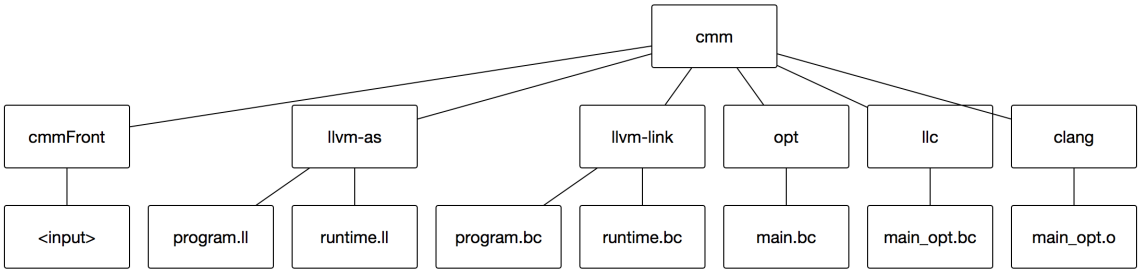

Figura 2: Arquitectura del compilador

Esta arquitectura corre en serie. Es decir el programa de hasta la izquierda recibe el archivo de input, corre el *front end* y genera el archivo **program.ll**. Las herramientas realizan lo siguiente:

1. **out/cmmFront**: Corre el *front end* que incluye el parser, el scanner y genera el código intermedio.
2. **llvm-as**: Convierte el código intermedio a bitcode.
3. **llvm-link**: *Linkea* los programas en bitcode.
4. **opt**: Optimiza el programa *linkeado* por **llvm-link**.
5. **llc**: Corre el *back end* y genera un archivo de objeto **.o**
6. **clang**: Corre el linker de otro compilador para *linkear* el **.o** con las *librerías* del sistema y generar un ejecutable.

1.7. Overshadowing de variables

Para manejar overshadowing de variables, generamos nombres diferentes en el archivo intermedio. Internamente en el *front end* utilizamos una estructura **t_symbol** que tiene dos cambios, **internalName** y **name**. **internalName** se asigna de acuerdo a las variables con el mismo nombre definidas en el bloque padre.

2. Manual de usuario

2.1. Compilación

Para compilar el programa tenemos un Makefile. Este archivo tiene las siguientes opciones:e

El comando **make** o **make help** muestran la documentación correspondiente de las opciones de make.

Una vez modificadas las variables, para compilar el programa utilizamos el comando: **make build**.

Para correr las pruebas del sistema utiliza el comando: **make test**.

Para limpiar los ejecutables utiliza el comando: **make clean**.

2.2. Corrimiento programa

Para compilar un programa utilizamos el comando **cmm**. El compilador tiene la siguiente ayuda:

```
usage: cmm [-h] [-o OUTPUT] [--optimize]
           [--target {assembly,executable,intermediate}]
           [--arch {mips,current}]
           source

Compiler for the cmm language. Currently uses a LLVM backend.

positional arguments:
  source                the source file to be compiled

optional arguments:
  -h, --help            show this help message and exit
  -o OUTPUT             the output destination file. By default this value is
                        result.{target_extension}
  --optimize            optimize the generated intermediate code
  --target {assembly,executable,intermediate}
                        tells the compiler which is the target it should
                        generate
  --arch {mips,current}
                        tells the compiler the target architecure you want for
                        your output. If the target doesn't generate code, this
                        flag is ignored
```

3. Código ejemplo

A continuación, colocamos un resultado de lo que genera el compilador. Incluyendo código fuente, código intermedio y código resultado.

3.1. Código fuente

```
int fact(int n) {
    int result;
    if (n == 1) {
        result = 1;
    } else {
        result = n * fact(n - 1);
    }

    return result;
}

int main() {
    int n;
    n = readInt();
    print(fact(n), "\n");

    return 0;
}
```

s7.cmm

3.2. Código intermedio

```
; ModuleID = 'main_opt.bc'
target triple = "mips-mipstechnologies"

@string_constant0 = internal constant [2 x i8] c"\0A\00"
@d = internal constant [3 x i8] c"%d\00"
@lf = internal constant [4 x i8] c"%lf\00"
@s = internal constant [3 x i8] c"%s\00"
@sn = internal constant [4 x i8] c"%s\0A\00"

; Function Attrs: nounwind readnone
define i32 @fact(i32 %_p__n) #0 {
  %1 = icmp eq i32 %_p__n, 1
  br i1 %1, label %6, label %2

; <label>:2                                       ; preds = %0
  %3 = add i32 %_p__n, -1
  %4 = call i32 @fact(i32 %3)
  %5 = mul i32 %4, %_p__n
  br label %6

; <label>:6                                       ; preds = %0, %2
  %result.0 = phi i32 [ %5, %2 ], [ 1, %0 ]
  ret i32 %result.0
}

; Function Attrs: nounwind
define i32 @main() #1 {
  %res.i = alloca i32, align 4
  %1 = bitcast i32* %res.i to i8*
  call void @llvm.lifetime.start(i64 4, i8* %1)
  %2 = call i32 (i8*, ...) @scanf(i8* getelementptr inbounds ([3 x i8], [3 x i8]* @d,
       i64 0, i64 0), i32* nonnull %res.i) #1
  %3 = load i32, i32* %res.i, align 4
  call void @llvm.lifetime.end(i64 4, i8* %1)
  %4 = call i32 @fact(i32 %3)
  %5 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([3 x i8], [3 x i8]* @d
      , i64 0, i64 0), i32 %4) #1
  %6 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([3 x i8], [3 x i8]* @s
      , i64 0, i64 0), i8* getelementptr inbounds ([2 x i8], [2 x i8]*
      @string_constant0, i64 0, i64 0)) #1
  ret i32 0
}

; Function Attrs: nounwind
define void @printInt(i32 %x) #1 {
  %1 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([3 x i8], [3 x i8]* @d
      , i64 0, i64 0), i32 %x)
  ret void
}

; Function Attrs: nounwind
declare i32 @printf(i8* nocapture readonly, ...) #1

; Function Attrs: nounwind
define void @printDouble(double %x) #1 {
  %1 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]*
      @lf, i64 0, i64 0), double %x)
```

```
52    ret void
53  }
54
55  ; Function Attrs: nounwind
56  define void @printString(i8* %s) #1 {
57    %1 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([3 x i8], [3 x i8]* @s
        , i64 0, i64 0), i8* %s)
58    ret void
59  }
60
61  ; Function Attrs: nounwind
62  define i32 @readInt() #1 {
63    %res = alloca i32, align 4
64    %1 = call i32 (i8*, ...) @scanf(i8* getelementptr inbounds ([3 x i8], [3 x i8]* @d,
         i64 0, i64 0), i32* nonnull %res)
65    %2 = load i32, i32* %res, align 4
66    ret i32 %2
67  }
68
69  ; Function Attrs: nounwind
70  declare i32 @scanf(i8* nocapture readonly, ...) #1
71
72  ; Function Attrs: nounwind
73  define double @readDouble() #1 {
74    %res = alloca double, align 8
75    %1 = call i32 (i8*, ...) @scanf(i8* getelementptr inbounds ([4 x i8], [4 x i8]* @lf
        , i64 0, i64 0), double* nonnull %res)
76    %2 = load double, double* %res, align 8
77    ret double %2
78  }
79
80  ; Function Attrs: nounwind
81  define i8* @readLine() #1 {
82    %res = alloca i8*, align 8
83    %1 = call i32 (i8*, ...) @scanf(i8* getelementptr inbounds ([4 x i8], [4 x i8]* @sn
        , i64 0, i64 0), i8** nonnull %res)
84    %2 = load i8*, i8** %res, align 8
85    ret i8* %2
86  }
87
88  ; Function Attrs: argmemonly nounwind
89  declare void @llvm.lifetime.start(i64, i8* nocapture) #2
90
91  ; Function Attrs: argmemonly nounwind
92  declare void @llvm.lifetime.end(i64, i8* nocapture) #2
93
94  attributes #0 = { nounwind readnone }
95  attributes #1 = { nounwind }
96  attributes #2 = { argmemonly nounwind }
```

s7.ll

3.3. Código resultado

```
    .text
    .abicalls
    .option pic0
    .section .mdebug.abi32,"",@progbits
    .nan    legacy
    .file   "main_opt.bc"
    .text
    .globl  fact
    .align  2
    .type   fact,@function
    .set    nomicromips
    .set    nomips16
    .ent    fact
fact:                                   # @fact
    .frame  $sp,32,$ra
    .mask   0x80000000,-4
    .fmask  0x00000000,0
    .set    noreorder
    .set    nomacro
    .set    noat
# BB#0:
    addiu   $sp, $sp, -32
    sw  $ra, 28($sp)            # 4-byte Folded Spill
    addiu   $1, $zero, 1
    move    $2, $1
    sw  $4, 24($sp)             # 4-byte Folded Spill
    sw  $2, 20($sp)             # 4-byte Folded Spill
    beq $4, $1, $BB0_3
    nop
# BB#1:
    j   $BB0_2
    nop
$BB0_2:
    lw  $1, 24($sp)             # 4-byte Folded Reload
    addiu   $4, $1, -1
    jal fact
    nop
    lw  $1, 24($sp)             # 4-byte Folded Reload
    mul $2, $2, $1
    sw  $2, 20($sp)             # 4-byte Folded Spill
$BB0_3:
    lw  $1, 20($sp)             # 4-byte Folded Reload
    move    $2, $1
    lw  $ra, 28($sp)            # 4-byte Folded Reload
    addiu   $sp, $sp, 32
    jr  $ra
    nop
    .set    at
    .set    macro
    .set    reorder
    .end    fact
$func_end0:
    .size   fact, ($func_end0)-fact

    .globl  main
    .align  2
    .type   main,@function
```

```
58      .set    nomicromips
59      .set    nomips16
60      .ent    main
61  main:                                   # @main
62      .frame  $sp,40,$ra
63      .mask   0x80000000,-4
64      .fmask  0x00000000,0
65      .set    noreorder
66      .set    nomacro
67      .set    noat
68  # BB#0:
69      addiu   $sp, $sp, -40
70      sw  $ra, 36($sp)            # 4-byte Folded Spill
71      lui $1, %hi(d)
72      addiu   $1, $1, %lo(d)
73      addiu   $5, $sp, 32
74      move     $4, $1
75      sw  $1, 28($sp)             # 4-byte Folded Spill
76      jal scanf
77      nop
78      lw  $4, 32($sp)
79      sw  $2, 24($sp)             # 4-byte Folded Spill
80      jal fact
81      nop
82      lw  $4, 28($sp)             # 4-byte Folded Reload
83      move     $5, $2
84      jal printf
85      nop
86      lui $1, %hi(s)
87      addiu   $4, $1, %lo(s)
88      lui $1, %hi(string_constant0)
89      addiu   $5, $1, %lo(string_constant0)
90      sw  $2, 20($sp)             # 4-byte Folded Spill
91      jal printf
92      nop
93      addiu   $1, $zero, 0
94      sw  $2, 16($sp)             # 4-byte Folded Spill
95      move     $2, $1
96      lw  $ra, 36($sp)            # 4-byte Folded Reload
97      addiu   $sp, $sp, 40
98      jr  $ra
99      nop
100     .set    at
101     .set    macro
102     .set    reorder
103     .end    main
104 $func_end1:
105     .size   main, ($func_end1)-main
106
107     .globl  printInt
108     .align  2
109     .type   printInt,@function
110     .set    nomicromips
111     .set    nomips16
112     .ent    printInt
113 printInt:                               # @printInt
114     .frame  $sp,32,$ra
115     .mask   0x80000000,-4
116     .fmask  0x00000000,0
```

```
117     .set    noreorder
118     .set    nomacro
119     .set    noat
120 # BB#0:
121     addiu   $sp, $sp, -32
122     sw  $ra, 28($sp)            # 4-byte Folded Spill
123     lui $1, %hi(d)
124     addiu   $1, $1, %lo(d)
125     sw  $4, 24($sp)             # 4-byte Folded Spill
126     move    $4, $1
127     lw  $5, 24($sp)             # 4-byte Folded Reload
128     jal printf
129     nop
130     sw  $2, 20($sp)             # 4-byte Folded Spill
131     lw  $ra, 28($sp)            # 4-byte Folded Reload
132     addiu   $sp, $sp, 32
133     jr  $ra
134     nop
135     .set    at
136     .set    macro
137     .set    reorder
138     .end    printInt
139 $func_end2:
140     .size   printInt, ($func_end2)-printInt
141
142     .globl  printDouble
143     .align  2
144     .type   printDouble,@function
145     .set    nomicromips
146     .set    nomips16
147     .ent    printDouble
148 printDouble:                                # @printDouble
149     .frame  $sp,24,$ra
150     .mask   0x80000000,-4
151     .fmask  0x00000000,0
152     .set    noreorder
153     .set    nomacro
154     .set    noat
155 # BB#0:
156     addiu   $sp, $sp, -24
157     sw  $ra, 20($sp)            # 4-byte Folded Spill
158     lui $1, %hi(lf)
159     addiu   $4, $1, %lo(lf)
160     mfc1    $6, $f13
161     mfc1    $7, $f12
162     jal printf
163     nop
164     sw  $2, 16($sp)             # 4-byte Folded Spill
165     lw  $ra, 20($sp)            # 4-byte Folded Reload
166     addiu   $sp, $sp, 24
167     jr  $ra
168     nop
169     .set    at
170     .set    macro
171     .set    reorder
172     .end    printDouble
173 $func_end3:
174     .size   printDouble, ($func_end3)-printDouble
175
```

```
176     .globl  printString
177     .align  2
178     .type   printString,@function
179     .set    nomicromips
180     .set    nomips16
181     .ent    printString
182 printString:                               # @printString
183     .frame  $sp,32,$ra
184     .mask   0x80000000,-4
185     .fmask  0x00000000,0
186     .set    noreorder
187     .set    nomacro
188     .set    noat
189 # BB#0:
190     addiu   $sp, $sp, -32
191     sw  $ra, 28($sp)            # 4-byte Folded Spill
192     lui $1, %hi(s)
193     addiu   $1, $1, %lo(s)
194     sw  $4, 24($sp)             # 4-byte Folded Spill
195     move     $4, $1
196     lw  $5, 24($sp)             # 4-byte Folded Reload
197     jal printf
198     nop
199     sw  $2, 20($sp)             # 4-byte Folded Spill
200     lw  $ra, 28($sp)            # 4-byte Folded Reload
201     addiu   $sp, $sp, 32
202     jr  $ra
203     nop
204     .set    at
205     .set    macro
206     .set    reorder
207     .end    printString
208 $func_end4:
209     .size   printString, ($func_end4)-printString
210
211     .globl  readInt
212     .align  2
213     .type   readInt,@function
214     .set    nomicromips
215     .set    nomips16
216     .ent    readInt
217 readInt:                                   # @readInt
218     .frame  $sp,32,$ra
219     .mask   0x80000000,-4
220     .fmask  0x00000000,0
221     .set    noreorder
222     .set    nomacro
223     .set    noat
224 # BB#0:
225     addiu   $sp, $sp, -32
226     sw  $ra, 28($sp)            # 4-byte Folded Spill
227     lui $1, %hi(d)
228     addiu   $4, $1, %lo(d)
229     addiu   $5, $sp, 24
230     jal scanf
231     nop
232     lw  $1, 24($sp)
233     sw  $2, 20($sp)             # 4-byte Folded Spill
234     move     $2, $1
```

```
235     lw  $ra, 28($sp)            # 4-byte Folded Reload
236     addiu   $sp, $sp, 32
237     jr  $ra
238     nop
239     .set    at
240     .set    macro
241     .set    reorder
242     .end    readInt
243 $func_end5:
244     .size   readInt, ($func_end5)-readInt
245
246     .globl  readDouble
247     .align  2
248     .type   readDouble,@function
249     .set    nomicromips
250     .set    nomips16
251     .ent    readDouble
252 readDouble:                                 # @readDouble
253     .frame  $sp,40,$ra
254     .mask   0x80000000,-4
255     .fmask  0x00000000,0
256     .set    noreorder
257     .set    nomacro
258     .set    noat
259 # BB#0:
260     addiu   $sp, $sp, -40
261     sw  $ra, 36($sp)            # 4-byte Folded Spill
262     lui $1, %hi(lf)
263     addiu   $4, $1, %lo(lf)
264     addiu   $5, $sp, 24
265     jal scanf
266     nop
267     ldc1    $f0, 24($sp)
268     sw  $2, 20($sp)             # 4-byte Folded Spill
269     lw  $ra, 36($sp)            # 4-byte Folded Reload
270     addiu   $sp, $sp, 40
271     jr  $ra
272     nop
273     .set    at
274     .set    macro
275     .set    reorder
276     .end    readDouble
277 $func_end6:
278     .size   readDouble, ($func_end6)-readDouble
279
280     .globl  readLine
281     .align  2
282     .type   readLine,@function
283     .set    nomicromips
284     .set    nomips16
285     .ent    readLine
286 readLine:                                   # @readLine
287     .frame  $sp,32,$ra
288     .mask   0x80000000,-4
289     .fmask  0x00000000,0
290     .set    noreorder
291     .set    nomacro
292     .set    noat
293 # BB#0:
```

```
294     addiu   $sp, $sp, -32
295     sw  $ra, 28($sp)            # 4-byte Folded Spill
296     lui $1, %hi(sn)
297     addiu   $4, $1, %lo(sn)
298     addiu   $5, $sp, 24
299     jal scanf
300     nop
301     lw  $1, 24($sp)
302     sw  $2, 20($sp)             # 4-byte Folded Spill
303     move     $2, $1
304     lw  $ra, 28($sp)            # 4-byte Folded Reload
305     addiu   $sp, $sp, 32
306     jr  $ra
307     nop
308     .set    at
309     .set    macro
310     .set    reorder
311     .end    readLine
312 $func_end7:
313     .size   readLine, ($func_end7)-readLine
314
315     .type   string_constant0,@object # @string_constant0
316     .section    .rodata,"a",@progbits
317     .align  2
318 string_constant0:
319     .asciz  "\n"
320     .size   string_constant0, 2
321
322     .type   d,@object               # @d
323     .align  2
324 d:
325     .asciz  "%d"
326     .size   d, 3
327
328     .type   lf,@object              # @lf
329     .align  2
330 lf:
331     .asciz  "%lf"
332     .size   lf, 4
333
334     .type   s,@object               # @s
335     .align  2
336 s:
337     .asciz  "%s"
338     .size   s, 3
339
340     .type   sn,@object              # @sn
341     .align  2
342 sn:
343     .asciz  "%s\n"
344     .size   sn, 4
345
346
347     .section    ".note.GNU-stack","",@progbits
348     .text
```

s7.s