PRÁCTICA LEX

Modelos de Computación

Ignacio Vellido Expósito: ignaciove@correo.ugr.es

Descripción

Se ha creado un programa que, a partir de un fichero en código fuente, realiza una series de transformaciones de formato para que resulte cómodo de leer. Este tipo de programas son comúnmente conocidos como “prettyprinters”, “beautifiers” o “prettifiers”.

La versión realizada ha sido desarrollada para un programa escrito en lenguaje C o C++, en el que se además de las transformaciones de formato permite numerar las líneas y avisar de algunos errores simples.

Desarrollo - Declaraciones

Se añade la opción noyywrap puesto que no realizamos ninguna operación al terminar de leer el fichero.

Además se definen contadores para el número de línea y la cantidad de tabulaciones a añadir.

Se almacena el carácter tabulador, que puede ser sustituido por espacios según las argumentos al ejecutar el programa.

```
2    /*-----------Declaraciones-----------------*/
3   %option noyywrap
4
5    int num_tab = 0, num_linea = 0;
6    int println = 0, error_parentesis = 0;
7    char *tab="\t";
8
9    // Siempre queremos imprimir el salto de línea, ya que ignoramos el carácter en las reglas
10   void imprime_lines () {
11     if (error_parentesis) {
12       printf("\t// ERROR: Falta paréntesis \")\"");
13       error_parentesis = 0;
14     }
15
16     printf("\n");
17     if (println)
18       printf("%d\t", num_linea);
19   }
20
21   void imprime_tabs () {
22     int i=0;
23
24     for (i; i<num_tab; i++)
25       printf("%s", tab);
26   }
27  %%
```

Desarrollo - Declaraciones

También se definen dos funciones:

- imprime_lines: **Imprime el número de línea y muestra un error de paréntesis si se ha detectado.**

- imprime_tabs: **Añade las tabulaciones al comienzo de la línea.**

```
2    /*-----------Declaraciones-----------------*/
3   %option noyywrap
4
5    int num_tab = 0, num_linea = 0;
6    int println = 0, error_parentesis = 0;
7    char *tab="\t";
8
9    // Siempre queremos imprimir el salto de línea, ya que ignoramos el carácter en las reglas
10   void imprime_lines () {
11     if (error_parentesis) {
12       printf("\t// ERROR: Falta paréntesis \")\"");
13       error_parentesis = 0;
14     }
15
16     printf("\n");
17     if (println)
18       printf("%d\t", num_linea);
19   }
20
21   void imprime_tabs () {
22     int i=0;
23
24     for (i; i<num_tab; i++)
25       printf("%s", tab);
26   }
27  %%
```

Desarrollo - Reglas

Para evitar problemas se ignoran todos los caracteres de salto de línea que vienen en el archivo.

```
28          /*-----------------Reglas-----------------*/
29
30          /*-------Caracteres ignorados-----------------------------*/
31      \n                              // Ignora los saltos de línea
32
```

Se deja un espacio entre el else **y la llave.**

```
91          /*---------------------If/else-----------------------------*/
94      "else"/"{"                      printf("else ");
95
```

Desarrollo - Reglas

Existen reglas para dejar un espacio entre cualquier carácter y un paréntesis.

Se añade una regla en la que se comprueba la falta de paréntesis. Para ello esta mira si encuentra un ; o un { antes del cierre del paréntesis.

Para las llaves se aumenta el nº de tabulaciones y se salta a una nueva línea.

```
34          /*------------Llaves y paréntesis-------------------------*/
35  ")"/"{"                     printf(") ");                   /* Añadir espacio si ) está junto a
36                                                                 {, con / decimos que la siguiente
37                                                                 parte pertenezca al siguiente token
38                                                                 leído */
39
40  [^ ]/"("                    printf("%c ", yytext[0]);       /* Separamos el paréntesis del carácter */
41
42  [^f][^o][^r]"("/[^)]*[;{]   ECHO; error_parentesis = 1;     /* Ver falta de cierre de paréntesis */
43
44  "{"                         { num_tab++; num_linea++; ECHO;
45                                imprime_lines();
46                                imprime_tabs();
47                              }
48
```

Desarrollo - Reglas

Regla para separar el ; del }, reduciendo tabulación.

Otra para separar varias llaves seguidas en distintas líneas.

Una más para cuando hay una sentencia que no es else **a continuación de una llave.**

```
50  ";}"                          { printf(";");
51
52                                  num_tab--; num_linea++;
53                                  imprime_lines();
54                                  imprime_tabs();
55
56                                  printf("} ");
57                                }
58
59  "}"                           { num_tab--; num_linea++;       /* Para cuando hay } encadenados */
60                                  imprime_lines();
61                                  imprime_tabs();
62
63                                  ECHO; printf(" ");
64                                }
65
66  "}"/[^ ][^e][^l][^s][^e]      { num_tab--; num_linea++;       /* Para cuando hay } no seguido
67                                                                  por un else */
68                                  imprime_lines();
69                                  imprime_tabs();
70
71                                  ECHO;
72
73                                  imprime_lines();             /* Dejamos una línea adicional */
74                                  imprime_tabs();
75                                }
```

Desarrollo - Reglas

Con los switch **se separa la palabra reservada** case **de la variable que acompaña (en el caso de que no exista un espacio) y se imprimen las sentencias con una tabulación adicional hasta que se encuentra con la palabra** break**.**

```
77          /*--------------------Switch-------------------------------*/
78      :                           { ECHO; num_linea++;
79                                    imprime_lines();
80                                    imprime_tabs();
81                                  }
82
83      case[^ ]                    { printf("case ");              /* Añadimos espacio tras "case" */
84                                    num_tab++;
85                                  }
86
87      case                        ECHO; num_tab++;
88
89      break                       ECHO; num_tab--;
90
```

Desarrollo - Reglas

En el caso de operadores, se le añade un espacio entre las variables que los rodean en el caso de que no estuvieran.

Algunas de estas reglas se podrían agrupar en una pero por visibilidad se han mantenido separadas.

```
96          /*--------------Operadores AND y OR-------------------------*/
97          /* Se separan los operadores de las variables.
98             Se imprime el carácter anterior junto a un espacio para
99             la parte izquierda, y al contrario con la derecha       */
100 [^ ]/"||"                       printf("%s ", yytext);
101 "||"/[^ ]                       printf("|| ");
102
103 [^ ]/"&&"                       printf("%s ", yytext);
104 "&&"/[^ ]                       printf("&& ");
105
106         /*--------------Operadores de comparación------------------*/
107         /* Se separan los operadores de las variables              */
108 [^ ]/"=="                      printf("%s ", yytext);
109 "=="/[^ ]                      printf("== ");
110
111 [^ ]/"!="                      printf("%s ", yytext);
112 "!="/[^ ]                      printf("!= ");
113
114 [^ ]/"<="                      printf("%s ", yytext);
115 "<="/[^ ]                      printf("<= ");
116
117 [^ ]/">="                      printf("%s ", yytext);
118 ">="/[^ ]                      printf(">= ");
119
120 [^ ]/"<"                       printf("%s ", yytext);
121 "<"/[^ ]                       printf("< ");
122
123 [^ ]/">"                       printf("%s ", yytext);
124 ">"/[^ ]                       printf("> ");
125
```

Desarrollo - Reglas

Cuando se encuentra un ; se aumenta el número de líneas y se salta a la siguiente.

En el caso delos bucles for**, se imprimen tal cuál para que no existan problemas con los ; que lo forman.**

```
126         /*---------------Saltos de línea----------------------------*/
127 "for"[^;]*;[^;]*;[^)]*/")"      ECHO;                          /* El for se copia tal cual */
128
129 ;                               { num_linea++; ECHO;
130                                   imprime_lines();
131                                   imprime_tabs();
132                                 }
133
```

Desarrollo - Reglas

El resto de caracteres no captados por ninguna regla se imprimen tal cuál.

Cuando se llega al final del fichero, puesto que en los ejemplos se muestra por terminal, se añade una línea adicional por visibilidad.
También se comprueba la falta de llaves, que se da cuando el número de tabulaciones es mayor que cero.

```
134         /*-------------------------------------------------------------*/
135     .                           ECHO;                       /* El resto de caracteres solo se imprimen */
136
137     <<EOF>>                     { printf("\n");             // Carácter de fin de archivo, añadimos
138                                                             // salto de línea para la lectura en terminal
139
140                                   // Comprobar falta de llaves {} = num_tab al final es distinto de 0
141                                   if (num_tab != 0)
142                                     printf("Error: Se ha llegado al final del fichero a falta de }\n");
143
144                                   return 0;
145                                 }
```

Desarrollo Procedimientos

En la función principal se comprueba el número de argumentos y se genera un vector con el número de espacios indicados en el argumento del programa.

```
132  int main (int argc, char **argv) {
133    if (argc == 3 || argc == 4) {
134      yyin = fopen(argv[1], "rt"); // Se abre fichero para lectura en modo texto
135      if (!yyin) { // Error
136        printf("No se pudo abrir el fichero %s\n", argv[1]);
137        return 0;
138      }
139
140      // Si queremos espacios se crea un array con el número indicado y se le asigna a tab
141      if (argc == 4) {
142        int tam = atoi(argv[3]);
143        char *aux = malloc(tam*sizeof(aux));;
144
145        int i=0;
146        for (i; i<tam; i++)
147          aux[i] = ' ';
148
149        tab = aux;
150      }
151    }
152    else {
153      printf("Uso: ./beautifier <archivo.cpp> <y/n> <tabulacion>\n");
154      exit(1);
155    }
156
157    if (argv[2][0] == 'y') {  // Comprobando si se quieren líneas
158      println = 1;
159      num_linea++;
160      printf("%d\t", num_linea);
161    }
162
163    //------
164    yylex();
165    //------
166
167    // Liberamos la memoria reservada si queríamos espacios
168    if (argc == 4) {
169      free(tab);
170    }
171    return 0;
172  }
```

Ejecución

Existen diversas formas de ejecutar el Makefile según lo que se desea que realice:

- run-no-lineas: **Ejecución normal sin números de línea.**
- run-lineas: **Muestra numeración en cada línea.**
- ejecuta: **Igual que la opción anterior**
- run-espacios: **Se utiliza una tabulación de dos espacios.**
- run-errores: **Se utiliza un ejemplo con errores para demostrar que los detecta.**

También se puede ejecutar el programa siguiendo el comando:

./beautifier <archivo> <y/n> [espacios]

Donde <y/n> indica si se desea numeración y [espacios] **el número de espacios deseados en la tabulación (por defecto utiliza** \t**).**

Ejecución

Para una entrada

```
int foo(int k){if(k<1||k>2){printf("out of range\n");
printf("this function requires a value of 1 or 2\n");}else{
printf("Switching\n");switch(k){case 1:printf("1\n");break;case
2:printf("2\n");break;}}}
```

Producimos la salida

```
nacho@nacho-GL73-8RD:~/Documents/flex$ make run-no-lineas
./beautifier ejemplo-beautifier.cpp n
int foo (int k) {
        if (k < 1 || k > 2) {
                printf ("out of range\n");
                printf ("this function requires a value of 1 or 2\n");
        } else {
                printf ("Switching\n");
                switch (k) {
                        case 1:
                                printf ("1\n");
                                break;
                        case 2:
                                printf ("2\n");
                                break;
                }
        }
}
```

Ejecución – Líneas numeradas

Para una entrada

```
int foo(int k){if(k<1||k>2){printf("out of range\n");
printf("this function requires a value of 1 or 2\n");}else{
printf("Switching\n");switch(k){case 1:printf("1\n");break;case
2:printf("2\n");break;}}}
```

Producimos la salida

```
nacho@nacho-GL73-8RD:~/Documents/flex$ make run-lineas
./beautifier ejemplo-beautifier.cpp y
1       int foo (int k) {
2               if (k < 1 || k > 2) {
3                       printf ("out of range\n");
4                       printf ("this function requires a value of 1 or 2\n");
5               } else {
6                       printf ("Switching\n");
7                       switch (k) {
8                               case 1:
9                                       printf ("1\n");
10                                      break;
11                              case 2:
12                                      printf ("2\n");
13                                      break;
14                      }
15              }
16      }
```

Ejecución – Espacios

Para una entrada

```
int foo(int k){if(k<1||k>2){printf("out of range\n");
printf("this function requires a value of 1 or 2\n");}else{
printf("Switching\n");switch(k){case 1:printf("1\n");break;case
2:printf("2\n");break;}}}
```

Producimos la salida

```
nacho@nacho-GL73-8RD:~/Documents/flex$ make run-espacios
./beautifier ejemplo-beautifier.cpp n 2
int foo (int k) {
  if (k < 1 || k > 2) {
    printf ("out of range\n");
    printf ("this function requires a value of 1 or 2\n");
  } else {
    printf ("Switching\n");
    switch (k) {
      case 1:
        printf ("1\n");
        break;
      case 2:
        printf ("2\n");
        break;
    }
  }
}
```

Ejecución – Errores

Para una entrada

```
int foo(int k){if(k<1||k>2){printf("out of range\n");
printf("this function requires a value of 1 or 2\n");}else{
printf("Switching\n";switch(k){case 1:printf("1\n");break;case
2:printf("2\n");break;}}
```

A la que se le ha quitado un paréntesis y una llave

Producimos la salida

```
nacho@nacho-GL73-8RD:~/Documents/flex$ make run-errores
./beautifier ejemplo-errores.cpp n
int foo (int k) {
        if (k < 1 || k > 2) {
                printf ("out of range\n");
                printf ("this function requires a value of 1 or 2\n");
        } else {
                printf("Switching\n";   // ERROR: Falta paréntesis ")"
                switch (k) {
                        case 1:
                                printf ("1\n");
                                break;
                        case 2:
                                printf ("2\n");
                                break;
                }
        }
Error: Se ha llegado al final del fichero a falta de }
```

PRÁCTICA LEX

Modelos de Computación

Ignacio Vellido Expósito: ignaciove@correo.ugr.es