Compyler. Un traductor de lenguaje natural a Python para listas

Vladislav Nikolov Vasilev
Nazaret Román Guerrero

Variables y alias

```
char * procesado;
char * elem;
int i = 0;
int j = 0;
int comparar = 0;
char * comparacion;
%}

letra           [a-zA-Z]
digito          [0-9]
espacio         [ ]
entero          \-?{digito}+
numero          {entero}(\.{digito}+)?
delimitador     "'"[^\t\n]+"'"
cadena          "'"({letra}|{digito}|{espacio})*"'"
variable        ({letra}|{digito}|_)+
crear           "crear "{variable}(" "({numero}|{cadena}|{espacio})+)?
longitud        "longitud "{variable}
imprimir        "imprimir "{variable}
recorrer        "recorrer "{variable}(" "{delimitador})?
insertar        "insertar "{variable}" "({cadena}|{numero})
borrar          "borrar "{variable}" "({cadena}|{numero})
obtener         "obtener "{variable}" "{entero}
copiar          "copiar "{variable}" "{variable}
concatenar      "concatenar "{variable}" "{variable}" "{variable}
ordenar         "ordenar "{variable}" reves"?
suma            {variable}" mas "{variable}(" mas "{variable})*
operador        "igual"|"diferente"|"menor"|"menor igual"|"mayor"|"mayor igual"
expresion       {longitud}|{suma}|{obtener}|{variable}|{ordenar}
comparar        "comparar "{expresion}" "{operador}" "{expresion}
```

Funcionalidades (I)

```
{crear}
{procesado = yytext + 6;
    char * lista = malloc(strlen(procesado));
    elem = malloc(strlen(procesado) * 2);
    for (i = 0; i < strlen(procesado) && procesado[i] != ' '; i++)
        lista[i] = procesado[i];
    if (i < strlen(procesado)) {
        procesado += i + 1;
        j = 0;
        int cambiar_espacio_coma = 0;
        for (i = 0; i < strlen(procesado); i++) {
            if (procesado[i] == ' ') {
                if (cambiar_espacio_coma != 0) {
                    elem[j] = ',';
                    j++;
                    elem[j] = ' ';
                    j++;
                    cambiar_espacio_coma = 0;
                }
            } else {
                elem[j] = procesado[i];
                j++;
                cambiar_espacio_coma = 1;
            }
        }
    }
    printf("%s = [%s]", lista, elem);}
{longitud}
{procesado = yytext + 9;
    if (comparar > 0)
        comparar--;
    printf("len(%s)", procesado);}
```

Figure 1: Expresiones para crear una lista y obtener la longitud.

Funcionalidades (II)

```
{imprimir}
{recorrer}
```

```
{procesado = yytext + 9; printf("print(%s)", procesado);}
{procesado = yytext + 9;
    char * lista = malloc(strlen(procesado));
    char * delim;
    int usar_delim = 0;
    for (i = 0; i < strlen(procesado) && procesado[i] != ' '; i++)
        lista[i] = procesado[i];
    if (strlen(procesado) >= i + 1)
        usar_delim = 1;
    procesado += i + 1;
    delim = procesado;
    printf("for item in %s:\n    print(item", lista);
    if (usar_delim)
        printf(", end = %s", delim);
    printf(")");
}
```

Figure 2: Expresiones para imprimir y recorrer una lista.

Funcionalidades (III)

```
{insertar}
{procesado = yytext + 9;
    char * lista = malloc(strlen(procesado));
    for (i = 0; i < strlen(procesado) && procesado[i] != ' '; i++)
        lista[i] = procesado[i];
    procesado += i + 1;
    elem = procesado;
    printf("%s.append(%s)", lista, elem);}
{borrar}
{procesado = yytext + 7;
    char *lista = malloc(strlen(procesado));
    for(i = 0; i < strlen(procesado) && procesado[i] != ' '; i++)
        lista[i] = procesado[i];
    procesado += i + 1;
    elem = procesado;
    printf("%s.remove(%s)", lista, elem);}
{obtener}
{procesado = yytext; procesado += 8;
    char *lista = malloc(strlen(procesado));
    for(i = 0; i < strlen(procesado) && procesado[i] != ' '; i++)
        lista[i] = procesado[i];
    procesado += i + 1;
    elem = procesado;
    if(comparar > 0) {
        printf("%s[%s]", lista, elem);
        comparar--;
    }
    else
        printf("print(%s[%s])", lista, elem);}
```

Figure 3: Expresiones para insertar, borrar y obtener un elemento de una lista.

Funcionalidades (IV)

```
{copiar}
{procesado = yytext + 7;
    char *orig = malloc(strlen(procesado));
    char *dest = malloc(strlen(procesado));
    for(i = 0; i < strlen(procesado) && procesado[i] != ' '; i++)
        orig[i] = procesado[i];
    procesado += i + 1;
    dest = procesado;
    printf("%s = %s", dest, orig);}
{concatenar}
{procesado = yytext + 11;
    char *l1 = malloc(strlen(procesado));
    char *l2 = malloc(strlen(procesado));
    char *dest = malloc(strlen(procesado));
    for(i = 0; i < strlen(procesado) && procesado[i] != ' '; i++)
        l1[i] = procesado[i];
    procesado += i + 1;
    for(i = 0; i < strlen(procesado) && procesado[i] != ' '; i++)
        l2[i] = procesado[i];
    procesado += i + 1;
    dest = procesado;
    printf("%s = %s + %s", dest, l1, l2);}
{ordenar}
{procesado = yytext + 8;
    char *lista = malloc(strlen(procesado));
    if (comparar > 0)
        comparar--;
    for(i = 0; i < strlen(procesado) && procesado[i] != ' '; i++)
        lista[i] = procesado[i];
    if (strlen(procesado) >= i + 1)
        printf("%s.sort(reverse=True)", lista);
    else
        printf("%s.sort()", lista);}
```

Figure 4: Expresiones para copiar, concatenar y ordenar listas.

Funcionalidades (V)

```
{suma}
```

```
{procesado = yytext;
    char * suma = malloc(strlen(procesado));
    int salta_palabra = 0;
    j = 0;

    if (comparar > 0)
        comparar--;
    for(i = 0; i < strlen(procesado); i++) {
        if (procesado[i] == ' ') {
            salta_palabra = !salta_palabra & 0x1;
            if(salta_palabra) {
                suma[j] = ' ';
                j++;
                suma[j] = '+';
                j++;
                suma[j] = ' ';
                j++;
            }
        } else {
            if (!salta_palabra) {
                suma[j] = procesado[i];
                j++;
            }
        }
    }
    printf("%s", suma);}
```

```
{comparar}
```

```
{comparar = 2; printf("comparacion = "); yyless(9);}
```

Figure 5: Expresiones para sumar y comparar listas.

Funcionalidades (VI)

```
{operador}
```

```
{procesado = yytext;
    i = 0;
    if(procesado[i] == 'i')
        printf(" == ");
    else if(procesado[i] == 'd')
        printf(" != ");
    else if(procesado[i] == 'm') {
        if(procesado[i+1] == 'e' && yyleng == 5)
            printf(" < ");
        else if(procesado[i+1] == 'e' && yyleng > 5)
            printf(" <= ");
        else if(procesado[i+1] == 'a' && yyleng == 5)
            printf(" > ");
        else
            printf(" >= ");}}
```

```
{variable}
```

```
{if (comparar > 0)
        comparar--;
printf("%s", yytext);}
```

```
"mostrar comparacion"
```

```
{printf("print(comparacion)");}
```

```
.
```

```
{}
```

Figure 6: Expresiones para procesar un operador, una variable, mostrar el resultado de una comparación y regla por defecto.

Ejemplo: salida por defecto

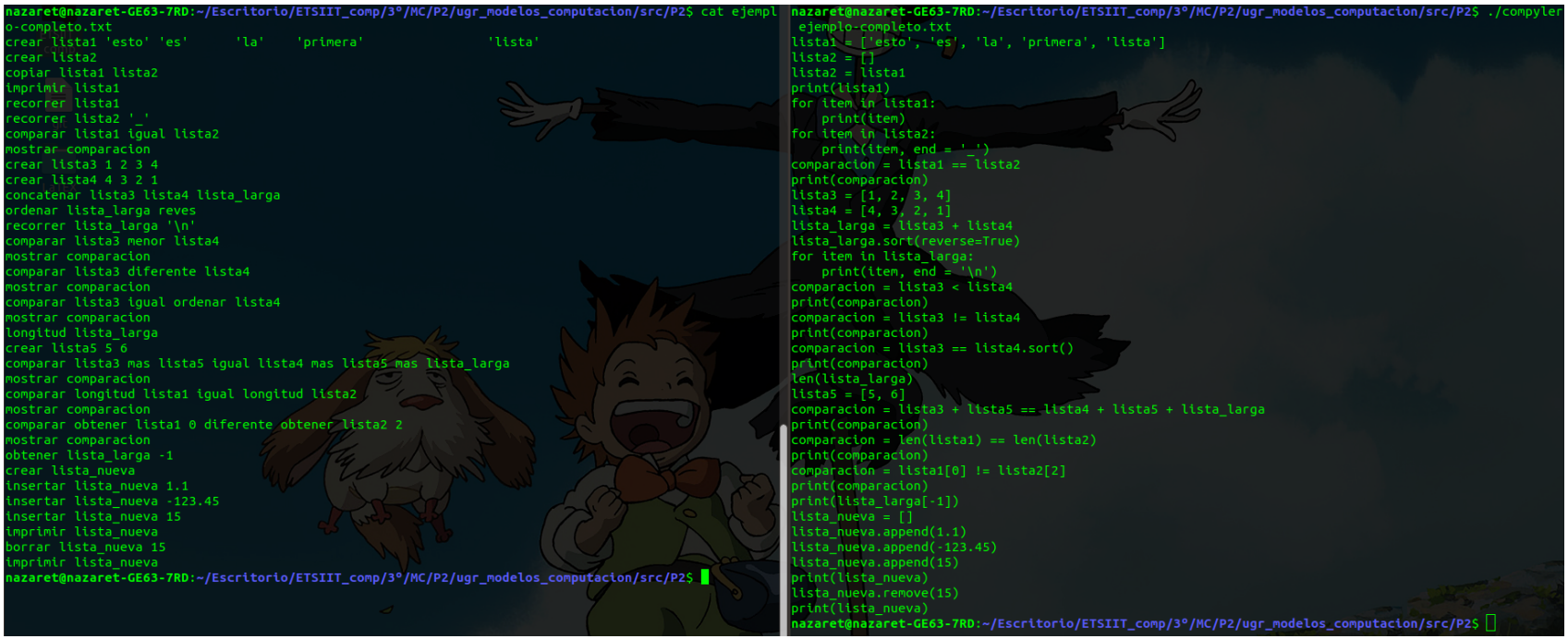

Figure 7: Salida por pantalla de la funcionalidad.

Ejemplo: salida a un archivo .py

Figure 8: Salida redirigida a un archivo .py y ejecutado en python.