Languages And Compilers Design's Project

Perna Mirco

A.A. 2024/2025

1 Introduction

The project involves developing a program using FLEX and BISON, aimed at producing a ranking of student admissions to degree courses based on entrance test scores and diploma grades.

2 Input and Output

2.1 Input

The input provided to the program is divided into three sections, separated by the delimiter strings %% and $$, each placed on a separate line. The first section specifies the academic year. The second section contains the criteria used by each degree program to determine the student ranking. The third part of the input file contains the data of individuals applying for university admission, whom we will refer to as candidate for simplicity. For each candidate, the following information is provided: full name, tax code, date of birth, diploma grade, entrance test score, and the list of degree programs for which they wish to apply for and in which they must therefore be included in the ranking. The format of the three sections is shown in the following table.

Section	**Format**
Academic Year	**AAAA/AA**
Degree program criteria	**Corso_di_laurea → n_posti; perc_voto_diploma; perc_voto_test.**
Candidate data	**Cognome e nome: Cognome Nome,** **Codice Fiscale: CF,** **Data di nascita: data_nascita,** **Voto di diploma: voto_diploma,** **Voto del test: voto_test.** **Lista corsi di Laurea: (cdl_1, cdl_2, . . . , cdl_k)** **!!!**

Table 1: Format of the three sections

2.2 Input example

2021/22
%%
Medicina → 500; 50; 50.
Biologia → 200; 75; 25.
Fisica → 50; 70; 30.
Chimica → 50; 35; 65.
$$$
Nome e Cognome: Maria Rosselli,
Codice Fiscale: RSSMRA77L07A546F,
Data di Nascita: 25/06/2004,
Voto di Diploma: 100,
Voto di Test: 88.
Iscrizione corsi di Laurea: (Medicina, Biologia).
!!!
Nome e Cognome: Francesco Amato,
Codice Fiscale: MTAFNC70H01H070L,
Data di Nascita: 09/09/2003,
Voto di Diploma: 80,
Voto di Test: 82.
Iscrizione corsi di Laurea: (Ingegneria Informatica, Fisica).
!!!

2.3 Output

The output consists, for each degree program considered, of a list of students along with their evaluation, sorted in descending order of score, and containing at most as many students as the number of available admissions. If two students have the same score, priority is given to the younger one. Here an example of output:

nome_cdl → (nome_1 cognome_1: valutazione_1) → (nome_2 cognome_2: valutazione_2)...

The candidate's evaluation is carried out using the following formula.

valutazione = (**voto_diploma** × **perc_voto_diploma**)/100 + (**voto_test** × **perc_voto_test**)/100

2.4 Output example

Medicina→ (Antonio Giusti: 98) → (Maria Rosselli: 94) → (Monica Ferrante: 91)
Biologia → (Maria Rosselli: 97) → (Antonio Giusti, 97) → (Monica Ferrante: 95,5) → (Giorgia Conte: 76,25)
Fisica→ (Marta Scuderi: 86,4) →v (Francesco Amato: 80,6) →(Giuseppe Genovese: 77,1)
Chimica →Monica Ferrante: 88,3)→(Giorgia Conte: 83,05)

3 Lexical analysis

In this phase, the lexical analyzer, or scanner, scans the sequence of characters that constitutes the source code, groups them into lexemes, and produces a sequence of tokens corresponding to these lexemes. Flex is used to implement the lexical analyzer. Flex is a tool used in compiler construction to generate lexical analyzers. It takes as input a file that specifies the lexicon of a certain language, usually in the form of regular expressions, including other auxiliary functions, token definitions, etc., and outputs code that implements the scanner.

3.1 Tokens

The following table shows the tokens that make up the input file, along with the regular expressions used to define their patterns.

Token	Regular expression
ACADEMIC_YEAR	`[0-9]{4}/[0-9]{2}`
ARROW	`->`
CANDIDATE_BIRTHDATE _PROPERTY	`Data di Nascita`
CANDIDATE_END_PROPERTY	`!!!`
CANDIDATE_COURSE _PROPERTY_END	`)`
CANDIDATE_COURSE _PROPERTY_START	`(`
CANDIDATE_COURSE_PROPERTY	`Iscrizione corsi di Laurea`
CANDIDATE_FISCAL_CODE _PROPERTY	`Codice Fiscale`
CANDIDATE_HIGH_SCHOOL _VOTE_PROPERTY	`Voto di Diploma`
CANDIDATE_NAME_PROPERTY	`Nome e Cognome`
CANDIDATES_SECTION	`$$$`
CANDIDATE_TEST_VOTE _PROPERTY	`Voto di Test`

DATE_VALUE	`(0[1-9]|1[0-9]|2[0-9]|3[01])/(0[1-9]|1[0-2])/[0-9]{4}`
DEGREE_COURSE	`([A-Z][a-z]*)([ ][A-Z][a-z]*)*`
DEGREE_COURSES_SECTION	`%%`
DIVEDER	`;`
DIVEDER	`,`
END_CANDIDATE_GENERAL _INFORMATION	`.`
END_COURSE_REQUIREMENTS	`.`
EQUAL_TO	`:`
FISCAL_CODE	`[A-Z]{6}[0-9]{2}[A-Z][0-9]{2}[A-Z][0-9]{3}[A-Z]`
INT	`0|[1-9][0-9]*`
STRING_VALUE	`([A-Z][a-z]*)([ ][A-Z][a-z]*)*`

Table 2: Tokens and their corresponding regular expressions

4 Syntactic analysis

In this phase, the parser uses tokens ,the names of the tokens, to define the grammatical structure of the token sequence, that is, it determines the structure of the individual instructions and checks whether the rules of the language syntax are respected. The tokens are the terminal symbols of the grammar that defines the language.To implement the syntactic analyzer Bison is used, which takes as input a file that specifies the syntax of a given language, expressed in the form of context-free grammar rules. The generator produces an output code that implements the behavior of the parser.

4.1 Context-free grammar

A context-free grammar (CFG) is a quadruple G=(T,N,S,P) where:

- T is the alphabet of terminal symbols
- N is the alphabet of variables or non-terminal symbols
- $S \in$N is the axiom
- P is the set of productions or grammatical rules of the form A $\rightarrow$ a , where $A \in$N and $a \in (N \cap \text{T})^*$

As mentioned earlier, terminal symbols are the tokens defined and generated during lexical analysis. As for the non-terminal symbols, they are the following:

- Birthdate
- Candidate
- Candidate_Properties
- Candidates_Section
- Courses
- Course_Selected
- Degree_Course
- Degree_Courses_Section
- Fiscal_Code
- High_School_Vote
- Main (axiom)
- Test_Vote

The grammar used must be a LALR(1) grammar, which is a grammar that can be parsed by an LALR(1) parser (which uses 1 token lookahead). The following is the grammar used:

1. Main→ ACADEMIC_YEAR DEGREE_COURSES_SECTION Degree_Courses_Section CANDIDATES_SECTION Candidates_Section
2. Degree_Courses_Section→ Degree_Courses_Section Degree_Course | ϵ
3. Degree_Course → DEGREE_COURSE ARROW INT DIVEDER INT DIVEDER INT END_COURSE_REQUIREMENTS
4. Candidates_Section:→ Candidates_Section Candidate_Properties | ϵ
5. Candidate_Properties→ Candidate Fiscal_Code Birthdate High_School_Vote Test_Vote Selected_Courses CANDIDATE_END_PROPERTY
6. Candidate→CANDIDATE_NAME_PROPERTY EQUAL_TO STRING_VALUE DIVEDER
7. Fiscal_Code→CANDIDATE_FISCAL_CODE_PROPERTY EQUAL_TO FISCAL_CODE DIVEDER
8. Birthdate →CANDIDATE_BIRTHDATE_PROPERTY EQUAL_TO DATE_VALUE DIVEDER
9. High_School_Vote→CANDIDATE_HIGH_SCHOOL_VOTE_PROPERTY EQUAL_TO INT DIVEDER
10. Test_Vote→CANDIDATE_TEST_VOTE_PROPERTY EQUAL_TO INT END_CANDIDATE_GENERAL_INFORMATION
11. Selected_Courses→ CANDIDATE_COURSE_PROPERTY EQUAL_TO CANDIDATE_COURSE_PROPERTY_START Courses CANDIDATE_COURSE_PROPERTY_END END_CANDIDATE_GENERAL_INFORMATION
12. Courses→ Course_Selected | Courses DIVEDER Course_Selected
13. Course_Selected →STRING_VALUE

5 Symbol table

All the information are stored in the Symbol table (ST). The most efficient data structure for representing a symbol table is the hash table, which is a table indexed by a hash function.

5.1 Hash function

The identifiers of the entities are strings; for this reason, a hash function was chosen that converts them to a natural number. To convert a string into a natural number, each character can be mapped to a value between 0 and 127, such as its ASCII code. Then, based on the character's position from right to left (starting from zero), each value is multiplied by 128 raised to the power of its position. In this way, the string is interpreted as a number expressed in a base-128 positional numeral system.The hash function is shown below.

```
#define HASHSIZE 101
unsigned int hash(char *id) {
        int h=0;
        int a = 128;
        for(;*id!='\0';id++){
                h=(a*h+*id)%HASHSIZE;
                }
        return h;
}
```

5.2 Check

Validation checks have been added to all insertion operations.

```
void check_course_data(course *new_course)
{
        if(new_course->amount_of_place_available<1 ||  new_course->
            amount_of_place_available>999)
        {
                printf("The number of available seats for course %s is not
                    between 1 and 999.\n",new_course->name);
                exit(1);
        }

        if(new_course->high_school_vote<0 ||  new_course->high_school_vote
            >100)
        {
                printf("The high school grade percentage for course %s is not
                    between 0 and 100.\n", new_course->name);
                exit(1);
        }

        if(new_course->test_vote<0 ||  new_course->test_vote>100)
        {
                printf("The test grade percentage for course %s is not between
                     0 and 100.\n", new_course->name);
                exit(1);
        }

        if(new_course->high_school_vote+new_course->test_vote > 100)
        {
                printf("The sum of diploma grade percentage and test grade
                    percentage for course %s is larger than 100\n", new_course
                    ->name);
                exit(1);
        }
}
```

```
27
28 int insert_course(char *name_course, int amount_of_place_available, int
       high_school_vote, int test_vote){
29
30         if(lookup_course(name_course)!=NULL){
31                 printf("'%s' has just been insterted into the symbol table\n",
                        name_course);
32                 return 0;
33         }
34
35         unsigned int hash_index =  hash(name_course);
36
37         course *new_course;
38
39         if ((new_course = (course *)malloc(sizeof(*new_course)))==NULL)
40                 return 0;
41
42         new_course->name = name_course;
43         new_course->amount_of_place_available = amount_of_place_available;
44         new_course->high_school_vote = high_school_vote;
45         new_course->test_vote = test_vote;
46         new_course->next = hash_table_courses[hash_index];
47
48         check_course_data(new_course);
49
50         hash_table_courses[hash_index] = new_course;
51
52         return 1;
53 }
54
55 int insert_candidate_high_school_vote(int high_school_vote)
56 {
57         if (candidate_to_insert ==NULL)
58                 return 0;
59
60         if(high_school_vote < 0 || high_school_vote > 100)
61         {
62                 printf("Student %s's high school score is not between 0 and
                        100.\n", candidate_to_insert->name);
63                 exit(1);
64         }
65
66         candidate_to_insert->high_school_vote = high_school_vote;
67
68         return 1;
69 }
70
71 int insert_candidate_test_vote(int test_vote)
72 {
73         if (candidate_to_insert ==NULL)
74                 return 0;
75
76         if(test_vote < 0 || test_vote > 100)
77         {
78                 printf("Student %s's test score is not between 0 and 100.\n",
                        candidate_to_insert->name);
79                 exit(1);
80         }
81         candidate_to_insert->test_vote = test_vote;
82         return 1;
83 }
```