program for the Examination Management System

Rishikesh Evane

16-11-2022

Objectives of Project

The objective of the project is to An examination management system is used to manage the complete examination process of an institute. It includes all exam-related activities

Function Description

`add()` :

This function that get the data from the user and update the list of the students. While adding the student into the list, check for the uniqueness of the Roll Number of the student.

`eligibleStudents()` :

This function shows the previous attendance percentage required for exams and get the data from the user and update the eligibility for the exams. It also updates the fee status required for the eligibility of exams by iterating over the List of the student records and for every student check the attendance percentage is above the percentage required and fee status of the student.

`execute()` :

This function will shows the available choices for the software and will perform the below functionality using Switch Statements. .

Functionality: `Add Student Details`

Eligible Students

`printStudents()` :

This function iterate over the list of students and print the details of the student. .

`deleteRecord()` :

This function get the student roll number to delete the student record and update the student's list. .

Profiling Report

```
Flat profile:

Each sample counts as 0.01 seconds.
 no time accumulated

  %   cumulative   self              self     total           
 time   seconds   seconds    calls  Ts/call  Ts/call  name    
  0.00      0.00     0.00        1     0.00     0.00  add
  0.00      0.00     0.00        1     0.00     0.00  eligibleStudents
  0.00      0.00     0.00        1     0.00     0.00  execute

 %         the percentage of the total running time of the
```

```
time       program used by this function.

cumulative a running sum of the number of seconds accounted
 seconds   for by this function and those listed above it.

 self      the number of seconds accounted for by this
seconds    function alone.  This is the major sort for this
           listing.

calls      the number of times this function was invoked, if
           this function is profiled, else blank.

 self      the average number of milliseconds spent in this
ms/call    function per call, if this function is profiled,
   else blank.

 total     the average number of milliseconds spent in this
ms/call    function and its descendents per call, if this
   function is profiled, else blank.

name       the name of the function.  This is the minor sort
           for this listing. The index shows the location of
   the function in the gprof listing. If the index is
   in parenthesis it shows where it would appear in
   the gprof listing if it were to be printed.
```

```
     Call graph (explanation follows)


granularity: each sample hit covers 4 byte(s) no time propagated

index % time    self  children    called     name
                0.00    0.00       1/1           main [82]
[2]      0.0    0.00    0.00       1         add [2]
                0.00    0.00       1/1           execute [4]
-----------------------------------------------
                0.00    0.00       1/1           execute [4]
[3]      0.0    0.00    0.00       1         eligibleStudents [3]
-----------------------------------------------
                                   1             execute [4]
                0.00    0.00       1/1           add [2]
[4]      0.0    0.00    0.00       1+1       execute [4]
                0.00    0.00       1/1           eligibleStudents [3]
                                   1             execute [4]
-----------------------------------------------

 This table describes the call tree of the program, and was sorted by
 the total amount of time spent in each function and its children.

 Each entry in this table consists of several lines.  The line with the
 index number at the left hand margin lists the current function.
 The lines above it list the functions that called this function,
 and the lines below it list the functions this one called.
 This line lists:
     index A unique number given to each element of the table.
Index numbers are sorted numerically.
The index number is printed next to every function name so
it is easier to look up where the function is in the table.

     % time This is the percentage of the ‘total’ time that was spent
in this function and its children.  Note that due to
different viewpoints, functions excluded by options, etc,
these numbers will NOT add up to 100%.

     self This is the total amount of time spent in this function.

     children This is the total amount of time propagated into this
function by its children.

     called This is the number of times the function was called.
If the function called itself recursively, the number
only includes non-recursive calls, and is followed by
a ‘+’ and the number of recursive calls.

     name The name of the current function.  The index number is
printed after it.  If the function is a member of a
cycle, the cycle number is printed between the
function’s name and the index number.


 For the function’s parents, the fields have the following meanings:

     self This is the amount of time that was propagated directly
from the function into this parent.

     children This is the amount of time that was propagated from
the function’s children into this parent.

     called This is the number of times this parent called the
function ‘/’ the total number of times the function
was called.  Recursive calls to the function are not
included in the number after the ‘/’.

     name This is the name of the parent.  The parent’s index
number is printed after it.  If the parent is a
member of a cycle, the cycle number is printed between
the name and the index number.

 If the parents of the function cannot be determined, the word
 ‘<spontaneous>’ is printed in the ‘name’ field, and all the other
 fields are blank.

 For the function’s children, the fields have the following meanings:

     self This is the amount of time that was propagated directly
from the child into the function.

     children This is the amount of time that was propagated from the
child’s children to the function.

     called This is the number of times the function called
this child ‘/’ the total number of times the child
was called.  Recursive calls by the child are not
listed in the number after the ‘/’.
```

```
     name This is the name of the child.  The child's index
number is printed after it.  If the child is a
member of a cycle, the cycle number is printed
between the name and the index number.

 If there are any cycles (circles) in the call graph, there is an
 entry for the cycle-as-a-whole.  This entry shows who called the
 cycle (as parents) and the members of the cycle (as children.)
 The `+' recursive calls entry shows the number of function calls that
 were internal to the cycle, and the calls entry for each member shows,
 for that member, how many times it was called from other members of
 the cycle.
```

```
Index by function name

   [2] add                      [3] eligibleStudents        [4] execute
```

Code in C Language

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int option = 0;
int i = 0;
int n = 0;
int j = 0;
float present = 75.00;
char money = 'P';
float tdays = 1;

// Structure of Student
struct student {
    char name[20];
    int rno;
    char fees;
    float days;
    float attend;
} s[50];

// Functions
void add(struct student s[]);
void eligibleStudents(struct student s[]);
void execute();
void printStudents(struct student s[]);
void deleteRecord(struct student s[]);

// Function to execute the software
// for the student examination
// registration system
void execute()
{
    printf(
        " Enter the serial number"
        "to select the option \n");
    printf(" 1. To show Eligible"
           "candidates \n");
    printf(" 2. To delete the "
           "student record \n");
    printf(" 3. To change the "
           "eligibility criteria \n");
    printf(" 4. Reset the "
           "eligibility criteria \n");
    printf(" 5. Print the list "
           "of all the student \n");
    printf(" Enter 0 to exit \n");

    scanf("%d", &option);

    // Switch Statement for choosing
    // the desired option for the user
    switch (option) {
    case 1:
        eligibleStudents(s);
        execute();
        break;
    case 2:
        deleteRecord(s);
        execute();
        break;

    case 3:
        printf("Old Attendance "
               "required = %f",
               present);
        printf(
            "\n Enter the updated "
            "attendence required \n");
        scanf("%f", &present);
        printf("fees status required"
               " was %c \n",
               money);
        printf("Enter the new fees "
               "status 'P' for paid 'N' "
               "for not paid and "
               "'B' for both \n");
        scanf("%c", &money);
        printf("Eligibility Criteria updated \n");
        execute();
        break;

    case 4:
        present = 75.00;
        money = 'P';
        printf("Eligibility creitria reset \n");
        execute();
        break;

    case 5:
        printStudents(s);
        execute();
        break;

    case 6:
        execute();
        break;

    case 0:
        exit(0);

    default:
        printf("Enter number only from 0-4 \n");
        execute();
    }
}

// Function to print the students record
void printStudents(struct student s[])
{
    // Loop to iterate over the students
    // students records
    for (i = 0; i < n; i++) {

        printf("Name of student %s \n",
               s[i].name);
        printf("Student roll number = %d \n",
               s[i].rno);
        printf("Student fees status = %c \n",
               s[i].fees);
        printf("Student number of days "
               "present = %f \n",
               s[i].days);
        printf("Student attendence = %f \n",
               s[i].attend);
    }
}

// Function to Student Record
void deleteRecord(struct student s[])
{
    int a = 0;
    printf("Enter the roll number of "
           "the student to delete it ");
    scanf("%d", &a);

    // Loop to iterate over the students
    // records to delete the Data
    for (i = 0; i <= n; i++) {
        // Condition to check the current
        // student roll number is same as
        // the user input roll number
        if (s[i].rno == (a)) {

            // Update record at ith index
            // with (i + 1)th index
            for (j = i; j < n; j++) {
                strcpy(s[j].name, s[j + 1].name);
                s[j].rno = s[j + 1].rno;
                s[j].fees = s[j + 1].fees;
                s[j].days = s[j + 1].days;
                s[j].attend = s[j + 1].attend;
            }
            printf("Student Record deleted");
        }
    }
}

// Function to print the student
// details of the eligible students
void eligibleStudents(struct student s[])
{
    printf("______"
           "______"
           "_____"
           "_____ \n");
    printf("Qualified student are = \n");

    // Iterate over the list
    // of the students records
    for (i = 0; i < n; i++) {
        // Check for the eligibility
        // of the student
        if (s[i].fees == money || 'B' == money) {
            if (s[i].attend >= present) {
                printf("Student name = %s \n",
                       s[i].name);
                printf("Student roll no. = %d \n",
                       s[i].rno);
                printf(" Student fees = %c \n",
                       s[i].fees);
                printf(" Student attendence = %f \n",
                       s[i].attend);
            }
        }
    }
}

// Function to add the students record
void add(struct student s[50])
{
    printf("Enter the total ");
    printf("number of working days \n");
    scanf("%f", &tdays);

    printf("Enter the number");
    printf("of students \n");
    scanf("%d", &n);

    for (i = 0; i < n; i++) {

        printf("Student number %d \n",
               (i + 1));

        printf("Enter the name of"
               " the student \n");
        scanf("%s", s[i].name);

        printf("Enter the roll number \n");
        scanf(" %d", &s[i].rno);

        printf("Enter the fees of the"
               "student 'P' for paid "
               ", 'N' for not paid \n");
        scanf(" %c", &s[i].fees);

        printf("Enter the number of"
               "days the student was "
               "present \n");
        scanf("%f", &s[i].days);

        s[i].attend = (s[i].days
                       / tdays)
                      * 100;
        printf("student attendence = %f \n",
               s[i].attend);
    }
    execute();
}

// Driver Code
int main()
{
    printf("Welcome to Student "
           "database registration \n");

    printf("Enter 0 to exit \n");
    printf("Enter 1 to add student"
           " record \n");
    scanf("%d", &option);

    // Switch Statements
    switch (option) {
    case 0:
        exit(0);

    case 1:
        add(s);
        break;

    default:
        printf("Only enter 0 or 1");
        execute();
    }
    return 0;
}
```

Output of Code in C Language

```
Welcome to Student database registration
Enter 0 to exit
Enter 1 to add student record
1
Enter the total number of working days
7
Enter the numberof students
2
Student number 1
Enter the name of the student
raj
Enter the roll number
34
Enter the fees of thestudent 'P' for paid , 'N' for not paid
P
Enter the number ofdays the student was present
6
student attendence = 85.714287
Student number 2
Enter the name of the student
om
Enter the roll number
65
Enter the fees of thestudent 'P' for paid , 'N' for not paid
N
Enter the number ofdays the student was present
4
student attendence = 57.142860
 Enter the serial numberto select the option
 1. To show Eligiblecandidates
 2. To delete the student record
 3. To change the eligibility criteria
 4. Reset the eligibility criteria
 5. Print the list of all the student
 Enter 0 to exit
1
----------------------
Qualified student are =
Student name = raj
Student roll no. = 34
 Student fees = P
 Student attendence = 85.714287
 Enter the serial numberto select the option
 1. To show Eligiblecandidates
 2. To delete the student record
 3. To change the eligibility criteria
 4. Reset the eligibility criteria
 5. Print the list of all the student
 Enter 0 to exit
```