Лабораторная работа 6 Автоматизированное тестирование примера

Автоматизированное тестирование с применением условной компиляции

Программа PassStats требует выполнения с различными наборами данных для ее тестирования. После любых изменений в коде мы должны снова полностью протестировать программу. Это регрессионное тестирование используется для проверки того, что изменения не привели к появлению новых ошибок в системе. Это может быть утомительно, и программист, скорее всего, будет использовать подмножество исходных тестов и упускать из виду некоторые возможные последствия измененного кода. Следующая программа TestStats имеет встроенную процедуру тестирования, которая заставляет процедуру CalcStats обрабатывать тестовые файлы, сравнивает результаты с ожидаемыми для этого файла и выводит результаты теста. Ожидаемые результаты для tn.csv (где n-цифра) находятся в файле expectedn.csv. Каждая строка в ожидаемом файле содержит strMalePassRate, strFemalePassRate или error,n (где n-номер ожидаемого сообщения об ошибке).

Код, написанный ниже, создает исполняемый файл, который выполняет вышеуказанное тестирование, поскольку он имеет директиву компилятора {$DEFINE DEBUG}. Чтобы получить исполняемый файл для выполнения обычных функций, как программа PassStats, вы должны раскомментировать эту директиву. Обычно мы делаем это, вставляя точку перед знаком доллара. Тогда знак доллара $ больше не следует сразу за скобкой, поэтому директива превращается в комментарий. Регрессионное тестирование становится легким!

Скриншот вывода следует за кодом.

Задание:

1. **Составить программу, проводящую в автоматизированном режиме тестирование разработанного на предыдущем занятии приложения**

   Добавить в свою программу процедуры автоматизированного тестирования с применением условной компиляции, выполнить свою программу в обычном режиме и в режиме автоматизированного тестирования.

2. Составить отчёт, в котором представить описание хода выполнения задания, листинг разработанной программы, скриншоты выполнения программы в обычном и тестовом режиме.

   В электронную среду представить отчёт и архив с проектом Lazarus

Program **PassStats** requires runs with different data sets to test it. After any changes in the code we should test the program fully again. This *regression testing* is used to check that changes have not introduced new faults into the system. This can be tedious and the programmer is likely to use a subset of the original tests and to overlook some

possible consequences of the changed code. The following program **TestStats** has a test procedure that repeatedly (i) instructs procedure **CalcStats** to process a test file, (ii) compares the results with those expected for that file and (iii) outputs the test results. The expected results for tn.csv (where n is a digit) are in file expectedn.csv. Each line in an expected file comprises either strMalePassRate,strFemalePassRate or error,n (where n is the number of the expected error message).

The code as written below creates an executable file that performs the above testing, because it has the {$DEFINE DEBUG} compiler directive. To obtain an executable to perform like Program **PassStats**, you must comment out this directive. Conventionally, we do this by inserting a full stop before the dollar sign. The $ no longer follows immediately after the brace, so the directive is changed into a comment. Regression testing becomes easy!

A screenshot of the output follows the code.

Листинг с аналогом программы.

```
program TestStats1;

  {$APPTYPE CONSOLE}
  {$DEFINE DEBUG} //Insert a full stop before the $ to compile the program
without the test.
  //Reads scores for males and females from a csv file
  //and outputs pass rate for each.
uses
  SysUtils, Strutils;
const
  PASSMARK = 40;
  ERROR_MESSAGES : array [0..6] of string = ('',
                                             'File not found',
                                             'File empty',
                                             'Comma should follow letter',
                                             'First letter must be M or F',
                                             'Mark must be between 0 and 100
inclusive',
                                             'An integer must be after the
comma');

var
  intCurrentScore, intMales, intFemales, intMalePasses, intFemalePasses,
intLine, intError, intCount : integer;
  strCurrentScore, strMalePassRate, strFemalepassRate : string;
  charCurrentGender : char;
  rMalePassRate, rFemalePassRate : real;
  ErrorFound : Boolean;

procedure CalcStats(strMarksFile : string);
var
  Marks, Results : Text;
  strCurrentLine :  string;
  CommaPos, ErrorCode : integer;

procedure OutPutError;
begin
  ErrorFound := True;
  write(ERROR_MESSAGES[intError]);
  writeln(Results, 'error' + ',' + intToStr(intError));
  if intLine = 0 then
    writeln
  else
    writeln(' at line ', intLine);
end;  //nested proc

begin
```

```
  intError := 0;
  intLine := 0;
  assignFile(Results, 'results.csv');
  rewrite(Results);

  if not fileExists(strMarksFile) then
    begin
      intError := 1;
      OutPutError;
    end
  else
    begin
      intMales := 0;
      intFemales := 0;
      intMalePasses := 0;
      intFemalepasses := 0;
      intCurrentScore := 0;
      ErrorFound := False;
      assignFile(Marks, strMarksFile);
      reset(Marks);
      if eof(Marks) then
        begin
          intError := 2;
          OutPutError;
        end
      else
        begin
          while not eof(Marks) do
            begin
              repeat
                readln(Marks, strCurrentLine);
                inc(intLine);
              until ((strCurrentLine <> ',') and (strCurrentLine <> ''))  or
eof(Marks);
              //Blank line or comma may be at the end of the file
              if not ((strCurrentLine = '') or (strcurrentline = ',')) then
                begin
                  CommaPos := pos(',', strCurrentline);
                  if (CommaPos <> 2) then
                    begin
                      intError := 3;  //Comma should follow letter
                      OutPutError;
                    end
                  else
                    begin
                      charCurrentGender := LeftStr(strCurrentLine, 1)[1];
                      charCurrentGender :=  UpCase(charCurrentGender);
                      if not (charCurrentGender in ['M', 'F']) then
                        begin
                          intError := 4;
                          OutPutError;
                        end
                      else
                        begin
                          strCurrentScore := rightStr(strCurrentLine,
length(strCurrentLine) - 2);
                          val(strCurrentScore, intCurrentScore, ErrorCode);
                          if not (ErrorCode = 0) then
                            begin
                              intError := 6;
                              OutPutError;
                            end
                          else
                            begin
```

```
                                if (intCurrentScore < 0) or (intCurrentScore >
100) then
                                  begin
                                    intError := 5;
                                    OutPutError;
                                  end
                                else  //no error detected
                                  begin
                                    if charCurrentGender = 'M' then
                                      begin
                                        inc(intMales);
                                        if intCurrentscore >= PASSMARK then
                                          inc(intMalePasses);
                                      end
                                    else //females
                                      begin
                                        inc(intFemales);
                                        if intCurrentscore >= PASSMARK then
                                          inc(intFemalePasses);
                                      end;
                                  end; //if (intCurrentScore < 0) or
(intCurrentScore > 100)
                              end; //if not (ErrorCode = 0)
                            end; //if not (charCurrentGender in ['M', 'F'])
                        end; //if CommaPos <> 2
                    end;  //if not ((strCurrentLine = '') or (strcurrentline =
','))
            end; //while
          if not ErrorFound then
            begin
              if intMales = 0 then
                begin
                  writeln('No males');
                  strMalePassRate := 'NoMales';
                end
              else
                begin
                  rMalePassRate := intMalePasses * 100 / intMales;
                  strMalePassRate := FloatToStrf(rMalePassRate, ffFixed, 6, 2);
                  writeln('Male Pass Rate (%): ', strMalePassRate);
                end;
              if intFemales = 0 then
                begin
                  writeln('No females');
                  strFemalePassRate := 'NoFemales';
                end
              else
                begin
                  rFemalePassRate := intFemalePasses * 100 / intFemales;
                  strFemalePassRate := FloatToStrf(rFemalePassRate, ffFixed, 6,
2);
                  writeln('Female Pass Rate (%): ', strFemalePassRate);
                end;
              writeln(Results, strMalePassRate + ',' + strFemalePassRate);
            end; //if not ErrorFound
          end; //if eof(Marks)
      end; //if not fileExists(FileName)
    closeFile(Results);
    if fileExists(strMarksFile) then
      closeFile(Marks);
end;

{$IFDEF  DEBUG}
procedure Test(File_No: integer);
```

```
var
  intLineNo : integer;
  strExpected, strResults, strTestfile, strExpectedFile : string;
  ExpectedFile, ResultsFile : text;
  ProgError : Boolean;

begin
  ProgError := False;
  strTestFile := 't' + intToStr(File_No) + '.csv';
  strExpectedFile := 'expected' + intToStr(File_No) + '.csv';
  writeln(#13#10'Output from test file ', File_No, ':');
  CalcStats(strTestFile);
  if fileExists(strExpectedFile) then
    assignFile(ExpectedFile, strExpectedFile);
  if fileExists('results.csv') then
    assignFile(ResultsFile, 'results.csv');
  reset(ExpectedFile);
  reset(ResultsFile);
  //Compare files line by line
  intLineNo := 0;
  while not eof(ExpectedFile) and not eof(ResultsFile) do
    begin
      readln(ExpectedFile, strExpected);
      readln(ResultsFile, strResults);
      inc(intLineNo);
      if strExpected <> strResults then
        begin
          ProgError := True;
          writeln('                                MISMATCH at Line ',
intLineNo);
          writeln('Expected: ',strExpected, ' Found: ', strResults);
        end;
      if eof(ExpectedFile) <> eof(ResultsFile) then
        begin
          ProgError := True;
          writeln('                                WRONG number of lines in
results file');
        end;
    end;
  if ProgError = False then
    writeln('                                          No programming error
detected by file ', File_No);
  closeFile(ExpectedFile);
  closeFile(ResultsFile);
end; //proc
{$ENDIF}

begin
  {$IfDEF  DEBUG}
  for intCount := 1 to 9 do
    Test(intCount);
  {$ELSE}
    CalcStats('marksheet.csv');
  {$ENDIF}
  readln;
end.
```

Пример скриншота выполнения программы в тестовом режиме.