Inhaltsverzeichnis

1. Funktionen zur Zeichenkettenverarbeitung

1.1 Lösungsidee

Laut Angabe muss eine Pascal-Funktion DeleteSubstring mit den Übergabeparametern s und substr angelegt werden. Der Parameter s repräsentiert unseren String und substr ist ein weiterer String, welchen wir aus s entfernen wollen. Zurückgeliefert soll ein String werden, die alle vorkommenden substr aus s entfernt hat.

Es wird zuerst eine Variable newString angelegt, welche einen leeren String darstellt. Als nächstes iterieren wir über alle Characters vom String s. Dabei wird überprüft, ob der aktuelle Character des Strings inklusive aller darauffolgenden Character, ob diese genau den Substring ergeben (insgesamt ist die Länge, die überprüft wird, der aktuelle Charakter + die darauffolgenden Characters, bis wir eine Länge haben, welche auch der Länge unseres Substrings entspricht.)

Wenn der aktuelle Charakter + darauffolgenden Charakters unseren Substring entsprechen, wird zum Zähler der for-Schleife die Länge des Substrings – 1 addiert (-1 kommt davon, weil wir bereits auf den nächsten Charakter zeigen würden, welcher möglicherweise wieder gültig ist, aber durch die for-Schleife der Zähler automatisch inkrementiert wird und somit würden wir das Zeichen auslassen.).

Wenn die Bedingung nicht zutrifft, wird der aktuelle Charakter an unserem newString angehängt. Im Endeffekt machen wir hier nichts anderes, als den newString mit unserem String s zu befüllen und unseren Substring s lassen wir aus.

Für die Überprüfung, ob unser aktueller Charakter + darauffolgenden Charakters genau der Substring sind, wird eine Funktion angelegt, in welcher unser String s und der substring übergeben werden. Außerdem wird unsere aktuelle Position von der Zählschleife übergeben (weiters Startposition genannt). Dann wird über alle Zeichen vom Substring iteriert und überprüft, ob unser aktuelles Zeichen von unserem Substring auch dem entspricht, welches sich an der Position Startposition + Zählerposition – 1 befindet. (-1 kommt in diesem Fall, weil der erste Charakter eines Strings sich an der Position 1 befinden und somit die Zählschleife von dem Substring bei 1 anfängt. Wir würden somit ohne -1 den aller ersten Character überspringen).

Wenn das aktuelle Zeichen vom Substring nicht mit dem von s an der Position Startposition + Zählervariable – 1 übereinstimmt, oder wir außerhalb unseres String s sind, beenden wir die Schleife und liefern false zurück. Das Gleiche soll auch passieren, wenn wir außerhalb der Schleife sind. Wenn alle Zeichen übereinstimmen und wir am Ende unseres Substrings angekommen sind, wird true zurückgeliefert.

1.2 Code

```
PROGRAM DeleteSubstringProgram;

CONST
  stringStartIndex = 1;

FUNCTION SubstringIsNextIndexes (s, substr : STRING; startPosition : INTEGER) : BOOLEAN;
  VAR
    i        : INTEGER;
    result   : BOOLEAN;
    position : INTEGER;
BEGIN
  result := FALSE;

  i := stringStartIndex;

  WHILE i <= Length(substr) DO BEGIN
    position := startPosition + i - 1;

    IF (s[position] <> substr[i]) OR (position > Length(s))
      THEN BEGIN
        i := Length(substr) + 1;
        result := FALSE;
      END;

    IF (i = Length(substr)) AND (s[position] = substr[i])
      THEN result := TRUE;

    Inc(i);
  END;

  SubstringIsNextIndexes := result;
END;



FUNCTION DeleteSubstring (s, substr : STRING) : STRING;
  VAR
    newString         : STRING;
    i                 : INTEGER;
BEGIN
  newString         := '';

  i := stringStartIndex;

  WHILE i <= Length(s) DO BEGIN
    IF (SubstringIsNextIndexes(s, substr, i)) THEN i := i + Length(substr) - 1
    ELSE newString := newString + s[i];

    Inc(i);
  END;

  DeleteSubstring := newString;
END;
```

```
PROCEDURE Test (inputString, substringToDelete, expectedOutput : STRING);
  VAR
    result : STRING;
BEGIN
  WriteLn('Input     : ', '"', inputString, '"');
  WriteLn('Substring : ', '"', substringToDelete, '"');
  WriteLn('Expected  : ', '"', expectedOutput, '"');
  result := DeleteSubstring(inputString, substringToDelete);
  WriteLn('Output    : ', '"', result, '"');
  if result = expectedOutput THEN
    WriteLn('Status    : OK')
  ELSE
    WriteLn('Status    : FAILED');
  WriteLn('--------------------------------------')

END;



VAR
  i : INTEGER;
  longString : STRING;
BEGIN
  Test('helloWorldHelloHello', 'llo', 'heWorldHeHe');
  Test('helloWorldHelloHell', 'llo', 'heWorldHeHell');
  Test('helloWorldHelloHello', '', 'helloWorldHelloHello');
  Test('Hallo, ich bin Mario!', 'Mario', 'Hallo, ich bin !');
  Test('Hallo, ich bin Mario!', ' ', 'Hallo,ichbinMario!');
  Test('Hallo, ich bin Mario!', ',', 'Hallo ich bin Mario!');
  Test('Hallo, ich bin Mario!', ' Mario!', 'Hallo, ich bin');
  Test('Zwischen zwei Zwetschgenzweigen schweben zwei zwitschernde Schwalben.',
       'zw',
       'Zwischen ei Zwetschgeneigen schweben ei itschernde Schwalben.');
  Test('', 'llo', '');
  Test('', '', '');
  Test(' ', ' ', '');

  longString := '';
  WriteLn('Very long string:');
  FOR i := 0 TO 100 DO longString := longString + 'Zwischen zwei Zwetschgenzweigen schweben
zwei zwitschernde Schwalben.' + 'Hello';
  WriteLn('Length: ', Length(longString));
  WriteLn('String: ', longString);
  WriteLn('Output: ');
  WriteLn(DeleteSubstring(longString, 'Hello'));
END.
```

Zusätzliche Anmerkung: Es wurde bewusst anstatt Zählschleifen für die Bildung des Strings ohne Substring verwendet, damit der Substring übersprungen werden kann und das mag der Compiler nicht so ganz.

1.3 Testfälle

```
// Normaler Test (im Text und am Ende enthalten)
Input     : "helloWorldHelloHello"
Substring : "llo"
Expected  : "heWorldHeHe"
Output    : "heWorldHeHe"
Status    : OK
-------------------------------------
// Im Text enthalten und am Ende angedeutet, aber nicht vollständig
Input     : "helloWorldHelloHell"
Substring : "llo"
Expected  : "heWorldHeHell"
Output    : "heWorldHeHell"
Status    : OK
-------------------------------------
// Ohne Angabe eines Substrings
Input     : "helloWorldHelloHello"
Substring : ""
Expected  : "helloWorldHelloHello"
Output    : "helloWorldHelloHello"
Status    : OK
-------------------------------------
// Mario entfernen
Input     : "Hallo, ich bin Mario!"
Substring : "Mario"
Expected  : "Hallo, ich bin !"
Output    : "Hallo, ich bin !"
Status    : OK
-------------------------------------
// Leerzeichen entfernen
Input     : "Hallo, ich bin Mario!"
Substring : " "
Expected  : "Hallo,ichbinMario!"
Output    : "Hallo,ichbinMario!"
Status    : OK
-------------------------------------
// Beistrich (Sonderzeichen entfernen)
Input     : "Hallo, ich bin Mario!"
Substring : ","
Expected  : "Hallo ich bin Mario!"
Output    : "Hallo ich bin Mario!"
Status    : OK
-------------------------------------
```

```
// „ Mario!“ entfernen (Leerzeichen + String + Sonderzeichen)
Input     : "Hallo, ich bin Mario!"
Substring : " Mario!"
Expected  : "Hallo, ich bin"
Output    : "Hallo, ich bin"
Status    : OK
-------------------------------------
// Ersten zwei Zeichen von einem Satz
Input     : "Zwischen zwei Zwetschgenzweigen schweben zwei zwitschernde
Schwalben."
Substring : "zw"
Expected  : "Zwischen ei Zwetschgeneigen schweben ei itschernde Schwalben."
Output    : "Zwischen ei Zwetschgeneigen schweben ei itschernde Schwalben."
Status    : OK
-------------------------------------
// Angabe eines Substrings, aber s ist leer
Input     : ""
Substring : "llo"
Expected  : ""
Output    : ""
Status    : OK
-------------------------------------
// Substrings und s sind leer
Input     : ""
Substring : ""
Expected  : ""
Output    : ""
Status    : OK
-------------------------------------
// s und substring sind ein Leerzeichen
Input     : " "
Substring : " "
Expected  : ""
Output    : ""
Status    : OK
-------------------------------------
```

```
// Spielen mit langen Strings (es wurde mit einer for-Loop versucht den String
// Zwischen zwei Zwetschgenzweigen schweben zwei zwitschernde Schwalben.Hello
// 100 Mal an einer Variable vom Typ string anzuhängen. Da der String aber nur
// 255 Characters haben kann ist alles darüber hinaus im String nicht mehr
// bekannt.
Very long string:
Length: 255
Length: Zwischen zwei Zwetschgenzweigen schweben zwei zwitschernde
Schwalben.HelloZwischen zwei Zwetschgenzweigen schweben zwei zwitschernde
Schwalben.HelloZwischen zwei Zwetschgenzweigen schweben zwei zwitschernde
Schwalben.HelloZwischen zwei Zwetschgenzweigen s
Output:
Zwischen zwei Zwetschgenzweigen schweben zwei zwitschernde Schwalben.Zwischen
zwei Zwetschgenzweigen schweben zwei zwitschernde Schwalben.Zwischen zwei
Zwetschgenzweigen schweben zwei zwitschernde Schwalben.Zwischen zwei
Zwetschgenzweigen s
```

2. Feldverarbeitung mit offenen Feldparametern: Schnittmenge

2.1 Lösungsidee

Gefordert ist eine PROCEDURE Intersect mit den Übergabeparametern a1, a2, n1, n2 und den Übergangsparametern a3 und n3. Die Parameter a1 und a2 stellen Felder (ARRAY OF INTEGER) dar, bei welchen die Schnittmengen gebildet wird und die Schnittmengen sollen in aufsteigender Reihenfolge in den Parameter a3 gespeichert werden, welches ebenfalls ein ARRAY OF INTEGER ist. In den Feldern a1 und a2 wird angenommen, dass dort ganzzahlige Positive zahlen gespeichert werden. Parameter n1, n2 und n3 geben an, wie viele Zahlen sich in a1, a2 und a3 befinden.

Wir gehen davon aus, dass a3 bereits definiert wurde, also auch die Größe bereits festgelegt wurde.

Zuerst müssen wir überprüfen, ob a1 und a2 genauso viele Zahlen beinhaltet, wieviel auch in n1 und n2 angegeben wurden. Danach wird überprüft, ob a1 und a2 bereits aufsteigend sortiert sind. Wenn einer der Bedingungen nicht zutrifft, haben wir einen Fehlerfall und n3 wird auf -1 gesetzt. Das Feld a3 wird nicht befüllt.

Bei der Überprüfung, ob ein Feld sortiert ist, wird über alle Elemente iteriert und überprüft, ob das aktuelle Element kleiner oder gleich dem nächsten Element ist.

Nun iterieren wir über jedes Element im Feld a1. Für jedes Element i im Feld a1 wird nochmals über jedes Element j im Feld a2 iteriert. Wenn wir eine Übereinstimmung von i und j haben, dann wird dieser Wert in das Feld a3 abgelegt, es wird von vorne begonnen. Der Wert für n3 wird initial auf 0 gesetzt und bei jedem Mal, wenn ein Wert eingefügt ist, wird n3 inkrementiert. Es kann passieren, dass wir einen Überlauf bekommen, da a3 möglicherweise kleiner definiert ist, als die Felder a1 und a2, aber mehr Werte übereinstimmen, als Platz in a3 vorhanden ist. In diesem Fall wird n3 ebenfalls auf -1 gesetzt.

2.2 Code

```
PROGRAM IntersectionCheck;

CONST
  errorCase = -1;

FUNCTION IsSorted (field: ARRAY OF INTEGER; n : INTEGER) : BOOLEAN;
  VAR
    result  : BOOLEAN;
    i       : INTEGER;
    endIndex: INTEGER;
BEGIN
  result := TRUE;

  i := Low(field);
  endIndex := Low(field) + n - 1;

  WHILE i < endIndex DO BEGIN
    IF field[i] > field[i + 1] THEN BEGIN
      result := FALSE;
      i := High(field);
    END;

    Inc(i);
  END;
  IsSorted := result;
END;


PROCEDURE AppendIntersectionItem (value : INTEGER; VAR field: ARRAY OF
INTEGER; VAR n : INTEGER);
  VAR
    startIndex  : INTEGER;
    i           : INTEGER;

BEGIN
  startIndex := Low(field);
  i := startIndex + n;
  IF i > High(field) THEN
    n := errorCase
  ELSE BEGIN
    field[i] := value;
    Inc(n);
  END;
END;


PROCEDURE ComputeIntersetcion (
      a1 : ARRAY OF INTEGER;      n1: INTEGER;
```

```
      a2 : ARRAY OF INTEGER;      n2: INTEGER;
  VAR a3 : ARRAY OF INTEGER; VAR  n3: INTEGER
);
  VAR
    i, j          : INTEGER;
    currentValue  : INTEGER;
    endA1         : INTEGER;
    endA2         : INTEGER;

BEGIN
  i := Low(a1);
  endA1 := Low(a1) + n1 - 1;

  endA2 := Low(a2) + n2 - 1;

  WHILE i <= endA1 DO BEGIN
    j := Low(a2);
    WHILE j <= endA2 DO BEGIN

      currentValue := a1[i];
      IF a1[i] = a2[j] THEN AppendIntersectionItem(currentValue, a3, n3);
      IF n3 = errorCase THEN BEGIN
        i := endA1 + 1;
        j := endA2 + 1;
      END;
      Inc(j);
    END;

    Inc(i);
  END;
END;


PROCEDURE Intersect (
      a1 : ARRAY OF INTEGER;      n1: INTEGER;
      a2 : ARRAY OF INTEGER;      n2: INTEGER;
  VAR a3 : ARRAY OF INTEGER; VAR  n3: INTEGER
);
  VAR
    validA1Length       : BOOLEAN;
    validA2Length       : BOOLEAN;
BEGIN
  n3 := 0;
  validA1Length := n1 <= Length(a1);
  validA2Length := n2 <= Length(a2);

  IF IsSorted(a1, n1) AND IsSorted(a2, n2) AND validA1Length AND validA2Length
THEN BEGIN
    ComputeIntersetcion(a1, n1, a2, n2, a3, n3);
```

```
  END ELSE IF (n1 = 0) OR (n2 = 0) THEN
    n3 := 0
  ELSE
    n3 := errorCase;
END;



PROCEDURE PrintField (field : ARRAY OF INTEGER; n : INTEGER);
  VAR
    i : INTEGER;
    firstIndex : INTEGER;

BEGIN
  WriteLn('n: ', n);

  WriteLn('Items in field:');
  IF n > 0 THEN BEGIN
    firstIndex := Low(field);
    Write(field[firstIndex]);
    FOR i := firstIndex + 1 TO firstIndex + n - 1 DO
      Write(', ', field[i]);
  END;
  WriteLn;
END;



PROCEDURE Test;
  VAR
    // a<number of items>
    a0 : ARRAY OF INTEGER;
    a2 : ARRAY [1..2] OF INTEGER;
    a3 : ARRAY [1..3] OF INTEGER;
    a4 : ARRAY [1..4] OF INTEGER;
    a5 : ARRAY [1..5] OF INTEGER;
    a6 : ARRAY [1..6] OF INTEGER;
    n : INTEGER;

BEGIN
  a6[1] := 1;
  a6[2] := 2;
  a6[3] := 3;
  a6[4] := 5;
  a6[5] := 8;
  a6[6] := 13;


  a5[1] := 1;
```

```
  a5[2] := 3;
  a5[3] := 5;
  a5[4] := 7;
  a5[5] := 9;

  WriteLn('Normaler Testfall (Länge von a3 = 3)');
  Intersect(a6, 6, a5, 5, a3, n);
  WriteLn('A1:');
  PrintField(a6, 6);
  WriteLn;
  WriteLn('A2:');
  PrintField(a5, 5);
  WriteLn;
  WriteLn('Result:');
  PrintField(a3, n);
  WriteLn('-----------------------------------------');
  WriteLn('Überlauf von A3');
  Intersect(a6, 6, a5, 5, a2, n);
  WriteLn('A1:');
  PrintField(a6, 6);
  WriteLn;
  WriteLn('A2:');
  PrintField(a5, 5);
  WriteLn;
  WriteLn('Result:');
  PrintField(a2, n);
  WriteLn('-----------------------------------------');
  a6[3] := 11;
  WriteLn('Unsortiertes A1');
  Intersect(a6, 6, a5, 5, a3, n);
  WriteLn('A1:');
  PrintField(a6, 6);
  WriteLn;
  WriteLn('A2:');
  PrintField(a5, 5);
  WriteLn;
  WriteLn('Result:');
  PrintField(a3, n);
  WriteLn;
  WriteLn('-----------------------------------------');
  a5[3] := 100;
  WriteLn('Unsortiertes A1 und A2');
  Intersect(a6, 6, a5, 5, a3, n);
  WriteLn('A1:');
  PrintField(a6, 6);
  WriteLn;
  WriteLn('A2:');
  PrintField(a5, 5);
  WriteLn;
```

```
  WriteLn('Result:');
  PrintField(a3, n);
  WriteLn;
  WriteLn('-------------------------------------------');
  a6[3] := 3;
  WriteLn('Unsortiertes A2');
  Intersect(a6, 6, a5, 5, a3, n);
  WriteLn('A1:');
  PrintField(a6, 6);
  WriteLn;
  WriteLn('A2:');
  PrintField(a5, 5);
  WriteLn;
  WriteLn('Result:');
  PrintField(a3, n);
  WriteLn;
  WriteLn('-------------------------------------------');
  a5[3] := 5;
  a6[3] := 3;
  WriteLn('Array A3 ohne items');
  Intersect(a6, 6, a5, 5, a0, n);
  WriteLn('A1:');
  PrintField(a6, 6);
  WriteLn;
  WriteLn('A2:');
  PrintField(a5, 5);
  WriteLn;
  WriteLn('Result:');
  PrintField(a0, n);
  WriteLn;
  WriteLn('-------------------------------------------');
  WriteLn('Array A1 ohne items');
  Intersect(a4, 0, a5, 5, a0, n);
  WriteLn('A1:');
  PrintField(a4, 0);
  WriteLn;
  WriteLn('A2:');
  PrintField(a5, 5);
  WriteLn;
  WriteLn('Result:');
  PrintField(a0, n);
  WriteLn;
  WriteLn('-------------------------------------------');
  WriteLn('Array A2 ohne items');
  Intersect(a5, 5, a4, 0, a0, n);
  WriteLn('A1:');
  PrintField(a5, 5);
  WriteLn;
  WriteLn('A2:');
```

```
  PrintField(a4, 0);
  WriteLn;
  WriteLn('Result:');
  PrintField(a0, n);
  WriteLn;
  WriteLn('-----------------------------------------');
  WriteLn('Array A1 und A2 ohne items');
  Intersect(a4, 0, a4, 0, a0, n);
  WriteLn('A1:');
  PrintField(a5, 5);
  WriteLn;
  WriteLn('A2:');
  PrintField(a4, 0);
  WriteLn;
  WriteLn('Result:');
  PrintField(a0, n);
  WriteLn;
  WriteLn('-----------------------------------------');


END;


BEGIN
  Test();
END.
```

Zusätzliche Anmerkung: Es wurde bewusst anstatt Zählschleifen für die Bildung der Schnittmengen verwendet, da wir bei einem Fehlerfall die Zählvariable manipulieren, damit unnötige Schleifendurchläufe vermieden werden und das mag der Compiler nicht so ganz.

2.3 Testfälle

```
Normaler Testfall (Länge von a3 = 3)
A1:
n: 6
Items in field:
1, 2, 3, 5, 8, 13

A2:
n: 5
Items in field:
1, 3, 5, 7, 9

Result:
n: 3
Items in field:
1, 3, 5
-------------------------------------------
Überlauf von A3
A1:
n: 6
Items in field:
1, 2, 3, 5, 8, 13

A2:
n: 5
Items in field:
1, 3, 5, 7, 9

Result:
n: -1
Items in field:
-------------------------------------------
Unsortiertes A1
A1:
n: 6
Items in field:
1, 2, 11, 5, 8, 13

A2:
n: 5
Items in field:
1, 3, 5, 7, 9

Result:
n: -1
Items in field:
-------------------------------------------
```

```
Unsortiertes A1 und A2
A1:
n: 6
Items in field:
1, 2, 11, 5, 8, 13

A2:
n: 5
Items in field:
1, 3, 100, 7, 9

Result:
n: -1
Items in field:
-------------------------------------------
Unsortiertes A2
A1:
n: 6
Items in field:
1, 2, 3, 5, 8, 13

A2:
n: 5
Items in field:
1, 3, 100, 7, 9

Result:
n: -1
Items in field:
-------------------------------------------
Array A3 als ARRAY OF INTEGER definiert
A1:
n: 6
Items in field:
1, 2, 3, 5, 8, 13

A2:
n: 5
Items in field:
1, 3, 5, 7, 9

Result:
n: -1
Items in field:
-------------------------------------------
```

```
Array A1 ohne items
A1:
n: 0
Items in field:


A2:
n: 5
Items in field:
1, 3, 5, 7, 9

Result:
n: 0
Items in field:
-----------------------------------------
Array A2 ohne items
A1:
n: 5
Items in field:
1, 3, 5, 7, 9

A2:
n: 0
Items in field:


Result:
n: 0
Items in field:
-----------------------------------------
Array A1 und A2 ohne items
A1:
n: 5
Items in field:
1, 3, 5, 7, 9

A2:
n: 0
Items in field:


Result:
n: 0
Items in field:
-----------------------------------------
```

Zusätzliche Anmerkung: Mit diesem Programm ist es möglich, auch die Schnittmengen von zwei Feldern mit ganzen Zahlen zu bilden, wenn diese auch aufsteigend sortiert sind. Laut Angabe befinden sich in den Feldern a1 und a2 nur positive ganze Zahlen, es wurde also hierfür keine Validierung eingebaut.

3. Roulette

3.1 Lösungsidee

a) Es ist eine Funktion BenefitForLuckyNr gefordert, welche die Übergabeparameter luckyNr und bet als INTEGER hat. luckyNr stellt unsere Zahl dar, auf welche wir setzen. Wir gehen davon aus, dass die Zahl zwischen 0 und 36 ist. Der Parameter bet stellt unseren Geldbetrag dar, welchen wir wetten. Zuerst generieren wir uns eine zufällige Zahl zwischen 0 und 36, welche das Ergebnis des Rouletts darstellt. Wenn unsere luckyNr der Zufallszahl entspricht, liefern wir unseren Wettbetrag multipliziert mit 36 zurück (= wir haben gewonnen), ansonsten wir der Wert 0 zurückgeliefert (= wir haben verloren). Es wird also ein Wert vom Typ INTEGER zurückgeliefert.

b) Hier wird eine Prozedur Test gewählt, welche auch dann für die Frage d angewendet werden kann. Es wird ein INTEGER luckyNrMode übergeben. Wenn dieser einen Wert zwischen 0 und 36 ist, wird als luckyNr diese Zahl übernommen. Bei -1 werden als luckyNr Zufallszahlen generiert.
Es ist gefordert, dass unser Programm so lange Roulette spielt, bis kein Geld mehr übrig ist. Dabei wird pro Spiel immer wieder mit 1 gesetzt. Das Startkapital beträgt 1000. Es wird also ein budget von 1000 festgelegt und initial werden ebenfalls Variablen angelegt, welche unsere gewonnenen und verlorenen Spiele speichern. Ebenfalls benötigen wir eine Variable, in welcher unser höchster Kontostand, welchen wir jemals hatten, gespeichert wird.
Nun wird gespielt! Es wird mit einer WHILE Loop so lange gespielt, bis wir kein Geld mehr haben. Es wird für jede Runde 1 von budget subtrahiert. Wenn wir gewinnen, wird der Gewinn zum budget addiert. Wenn wir das Spiel gewonnen haben, wird gamesWon inkrementiert, wenn wir verloren haben, wird gamesLost inkrementiert. Wenn unser budget größer ist, als unser letzter höchster Kontostand, ist dies unser neuer höchster Kontostand.
Am Schluss werden unsere verlorenen Spiele, gewonnenen Spiele und unser höchster Kontostand, den wir jemals hatten, ausgegeben.
Anmerkung: Für die Variablen budget, gamesLost, gamesWon und maxMoney wurde ein LONGINT verwendet, da es mit einer sehr geringen Wahrscheinlichkeit trotzdem möglich ist, über den Wertebereich des INTEGER hinauszugehen. Es wurde hier mit dem Hintergedanken gearbeitet, die Wahrscheinlichkeit eines Programmabsturzes zu minimieren. Die Variable win wurde als INTEGER gewählt, da immer nur 1€ gesetzt und der Wert für win maximal nur 36 haben kann.

c) Es ist eine Funktion BenefitForEvenNr gefordert, welche den Übergabeparameter bet als INTEGER hat. Diese stellt unseren Wettbetrag dar. Zuerst generieren wir eine zufällige Zahl zwischen 0 und 36, welche das Ergebnis des Rouletts darstellt. Danach ermitteln wir, ob die Zufallszahl gerade oder ungerade ist, dabei wird der Rest aus der Zufallszahl durch 2 ermittelt. Wenn diese 0 ist, ist die Zahl gerade, wenn sie 1 ist, dann ungerade. Der Rest kann nur 0 oder 1 sein. Wenn nun der Rest 0 ist und das Ergebnis des Rouletts nicht 0 ist, dann liefert die Funktion das doppelte von unserem Einsatz zurück (=Gewinn), ansonsten wird 0 zurückgeliefert (=Verlust). 0 wird übrigens beim Roulette weder als gerade noch ungerade gesehen. Der Rückgabewert ist vom Typ INTEGER.

d) Punkt b wird um einen luckyNrMode -2 erweitert, welche nur auf gerade Zahlen setzt.

Zu den Hinweisen:

- Random ist laut der Dokumentation vom Freepascal als Funktion definiert, deswegen kann diese Funktion aus meiner Sicht auch in Funktionen verwendet werden. Dies bleibt jedoch ein Streitthema, da man auch argumentieren könnte, dass man hier Seiteneffekte haben könnte. Ich weiß nicht, wie die Funktion im Hintergrund aufgebaut ist, daher nehme ich an, dass es im Hintergrund zu keinen Seiteneffekten kommt.
- Randomize ist richtig fies, da dies keine Funktion, sondern eine Prozedur ist laut der Doku vom Freepascal. Dadurch, dass es geheißen hat, in Funktionen sollen wir keine Prozeduren aufrufen, wird Randomize am Beginn eines Testfalls initialisiert.

3.2 Code

```
PROGRAM Roulette;

CONST
  numbersOfRoulette = 37;
  betOnEven         = -2;
  betOnRandom       = -1;


FUNCTION BenefitForEvenNr (bet : INTEGER) : INTEGER;
  VAR
    randomNumber  : INTEGER;
    win           : INTEGER;
    restOfDivision: INTEGER;
BEGIN
  // Get a random number between 0 and 36
  randomNumber := Random(numbersOfRoulette);

  restOfDivision := randomNumber MOD 2;
  win := 0;

  IF (randomNumber <> 0) AND (restOfDivision = 0) THEN
    win := bet * 2;

  BenefitForEvenNr := win;
END;


FUNCTION BenefitForLuckyNr (luckyNr, bet: INTEGER) : INTEGER;
  VAR
    win           : INTEGER;
    randomNumber  : INTEGER;
BEGIN

  // Get a random number between 0 and 36
  randomNumber := Random(numbersOfRoulette);
  win := 0;

  IF luckyNr = randomNumber THEN
    win := bet * 36
  ELSE
    win := 0;
  BenefitForLuckyNr := win;
END;
```

```
FUNCTION PlaySingleGame (luckyNrMode, bet : INTEGER) : INTEGER;
  VAR
    win : INTEGER;
    luckyNr : INTEGER;
BEGIN
  IF luckyNrMode = betOnEven THEN
    win := BenefitForEvenNr(bet)

  ELSE BEGIN
    IF luckyNrMode = betOnRandom THEN
      luckyNr := Random(numbersOfRoulette)
    ELSE
      luckyNr := luckyNrMode;
    win := BenefitForLuckyNr(luckyNr, bet)
  END;

  PlaySingleGame := win
END;


PROCEDURE Test (luckyNrMode : INTEGER);
  VAR
    budget    : LONGINT;
    gamesLost : LONGINT;
    gamesWon  : LONGINT;
    maxMoney  : LONGINT;
    win       : INTEGER;
BEGIN
  budget    := 1000;
  gamesLost := 0;
  gamesWon  := 0;
  maxMoney  := 1000;

  // Randomize ist echt ein leger, da es als procedure definiert ist
  Randomize();

  IF (-2 <= luckyNrMode) AND (luckyNrMode <= 36) THEN BEGIN
    WHILE (budget > 0) DO BEGIN
      budget := budget - 1;
      win := PlaySingleGame(luckyNrMode, 1);

      IF win > 0 THEN BEGIN
        budget := budget + win;
        gamesWon := gamesWon + 1;
      END
      ELSE
        gamesLost := gamesLost + 1;
    END;
  END;
```

```
  WriteLn('Games lost: ', gamesLost);
  WriteLn('Games won : ', gamesWon);
  WriteLn('Max. money: ', maxMoney);
END;


VAR
  i : INTEGER;
BEGIN
  WriteLn('Testcase : Bet on even numbers:');
  Test(betOnEven);
  WriteLn('------------------------------');

  WriteLn('Testcase : Bet on random lucky numbers:');
  Test(betOnRandom);
  WriteLn('--------------------------------------');

  FOR i := 0 TO 36 DO BEGIN
    WriteLn('Testcase : Bet on lucky number ', i, ':');
    Test(i);
    WriteLn('--------------------------------------');

  END;
END.
```

3.3 Testfälle

```
Testcase : Bet on even numbers:
Games lost: 21596
Games won : 20596
Max. money: 1000
------------------------------
Testcase : Bet on random lucky numbers:
Games lost: 7090
Games won : 174
Max. money: 1000
-------------------------------------
Testcase : Bet on lucky number 0:
Games lost: 41005
Games won : 1143
Max. money: 1000
-------------------------------------
Testcase : Bet on lucky number 1:
Games lost: 18185
Games won : 491
Max. money: 1000
-------------------------------------
Testcase : Bet on lucky number 2:
Games lost: 57770
Games won : 1622
Max. money: 1000
-------------------------------------
Testcase : Bet on lucky number 3:
Games lost: 113455
Games won : 3213
Max. money: 1000
-------------------------------------
Testcase : Bet on lucky number 4:
Games lost: 12620
Games won : 332
Max. money: 1000
-------------------------------------
Testcase : Bet on lucky number 5:
Games lost: 22875
Games won : 625
Max. money: 1000
-------------------------------------
Testcase : Bet on lucky number 6:
Games lost: 22560
Games won : 616
Max. money: 1000
-------------------------------------
```

```
Testcase : Bet on lucky number 7:
Games lost: 7405
Games won : 183
Max. money: 1000
-------------------------------------
Testcase : Bet on lucky number 8:
Games lost: 50070
Games won : 1402
Max. money: 1000
-------------------------------------
Testcase : Bet on lucky number 9:
Games lost: 37575
Games won : 1045
Max. money: 1000
-------------------------------------
Testcase : Bet on lucky number 10:
Games lost: 3555
Games won : 73
Max. money: 1000
-------------------------------------
Testcase : Bet on lucky number 11:
Games lost: 98020
Games won : 2772
Max. money: 1000
-------------------------------------
Testcase : Bet on lucky number 12:
Games lost: 23435
Games won : 641
Max. money: 1000
-------------------------------------
Testcase : Bet on lucky number 13:
Games lost: 19165
Games won : 519
Max. money: 1000
-------------------------------------
Testcase : Bet on lucky number 14:
Games lost: 4990
Games won : 114
Max. money: 1000
-------------------------------------
Testcase : Bet on lucky number 15:
Games lost: 83390
Games won : 2354
Max. money: 1000
-------------------------------------
Testcase : Bet on lucky number 16:
Games lost: 49790
Games won : 1394
Max. money: 1000
```

```
--------------------------------------
Testcase : Bet on lucky number 17:
Games lost: 100435
Games won : 2841
Max. money: 1000
--------------------------------------
Testcase : Bet on lucky number 18:
Games lost: 25115
Games won : 689
Max. money: 1000
--------------------------------------
Testcase : Bet on lucky number 19:
Games lost: 7230
Games won : 178
Max. money: 1000
--------------------------------------
Testcase : Bet on lucky number 20:
Games lost: 140965
Games won : 3999
Max. money: 1000
--------------------------------------
Testcase : Bet on lucky number 21:
Games lost: 86750
Games won : 2450
Max. money: 1000
--------------------------------------
Testcase : Bet on lucky number 22:
Games lost: 16820
Games won : 452
Max. money: 1000
--------------------------------------
Testcase : Bet on lucky number 23:
Games lost: 54235
Games won : 1521
Max. money: 1000
--------------------------------------
Testcase : Bet on lucky number 24:
Games lost: 12970
Games won : 342
Max. money: 1000
--------------------------------------
Testcase : Bet on lucky number 25:
Games lost: 23680
Games won : 648
Max. money: 1000
--------------------------------------
Testcase : Bet on lucky number 26:
Games lost: 11605
Games won : 303
```

```
Max. money: 1000
-------------------------------------
Testcase : Bet on lucky number 27:
Games lost: 29210
Games won : 806
Max. money: 1000
-------------------------------------
Testcase : Bet on lucky number 28:
Games lost: 32815
Games won : 909
Max. money: 1000
-------------------------------------
Testcase : Bet on lucky number 29:
Games lost: 36525
Games won : 1015
Max. money: 1000
-------------------------------------
Testcase : Bet on lucky number 30:
Games lost: 47165
Games won : 1319
Max. money: 1000
-------------------------------------
Testcase : Bet on lucky number 31:
Games lost: 74150
Games won : 2090
Max. money: 1000
-------------------------------------
Testcase : Bet on lucky number 32:
Games lost: 16505
Games won : 443
Max. money: 1000
-------------------------------------
Testcase : Bet on lucky number 33:
Games lost: 151115
Games won : 4289
Max. money: 1000
-------------------------------------
Testcase : Bet on lucky number 34:
Games lost: 19585
Games won : 531
Max. money: 1000
-------------------------------------
Testcase : Bet on lucky number 35:
Games lost: 16785
Games won : 451
Max. money: 1000
-------------------------------------
Testcase : Bet on lucky number 36:
Games lost: 27915
```

```
Games won : 769
Max. money: 1000
---------------------------------------
```

3.4 Analyse Unterschied zwischen BenefitForEvenNr und BenefitForLuckyNr

Wie aus Punkt 3.3 Testfälle zu entnehmen ist, verteilt sich die Wahrscheinlichkeitsrechnung anders. Wenn wir alle Zahlen von 0 bis 36 insgesamt 1000 Mal durchprobieren (z.B. 0 wird 1000 Mal ausprobiert), dann gibt es beim Roulette immer 37 verschiedene Möglichkeiten, welche Zahl jetzt das Ergebnis ist, somit ist die Wahrscheinlichkeit, dass unsere LuckyNr auch das Ergebnis des Rouletts ist, ungefähr bei 2,6 Prozent (1/37). Wenn wir aus den Testfällen für die lucky Numbers berechnen indem wir gamesWon / (gamesWon + gamesLost) rechnen, dann kommen wir immer auf eine Gewinnwahrscheinlichkeit zwischen 2 und 3 Prozent hin. Es spielt hier auch keine Rolle, ob wir jedes Mal eine andere Lucky Number wählen oder die gleiche für die 1000 versuche.

Wenn wir nun auf nur gerade Zahlen setzen, dann haben wir einen Anstieg in der Wahrscheinlichkeit. Es gibt insgesamt 37 verschiedene Möglichkeiten, wir gewinnen etwas, wenn die Zahl gerade und nicht null ist. Somit haben wir für 18 verschiedene Zahlen die Möglichkeit, zu gewinnen, wir haben eine Wahrscheinlichkeit von 18/37, sprich eine Wahrscheinlichkeit von 48,6 Prozent. Wenn wir nun 1000 Spiele auf eine gerade Zahl setzen, dann haben wir dort auch ungefähr eine Gewinnwahrscheinlichkeit von 18/37, in unserem Testfall hatten wir sogar eine Gewinnwahrscheinlichkeit von 48,8 Prozent (hier wurde gewonnene Spiele durch insgesamt gespielte Spiele gerechnet).