


```
72:     Label10: TLabel;
73:     Label11: TLabel;
74:     Label12: TLabel;
75:     Label13: TLabel;
76:     Label14: TLabel;
77:     Label15: TLabel;
78:     Label16: TLabel;
79:     Label17: TLabel;
80:     Label18: TLabel;
81:     Label19: TLabel;
82:     GroupBox6: TGroupBox;
83:     Label20: TLabel;
84:     Label21: TLabel;
85:     Label22: TLabel;
86:     Label23: TLabel;
87:     Label24: TLabel;
88:     Label25: TLabel;
89:     Label26: TLabel;
90:     Label27: TLabel;
91:     Label28: TLabel;
92:     Label29: TLabel;
93:     Label30: TLabel;
94:     Label31: TLabel;
95:     Label32: TLabel;
96:     Label33: TLabel;
97:     Label34: TLabel;
98:     GroupBox7: TGroupBox;
99:     MKlartext: TMemo;
100:    GroupBox8: TGroupBox;
101:    MGeheimtext: TMemo;
102:    BtEncrypt: TButton;
103:    BtDecrypt: TButton;
104:    OpenDialog1: TOpenDialog;
105:    procedure Info1Click(Sender: TObject);
106:    procedure Exit1Click(Sender: TObject);
107:    procedure BtGenerierePrimzahlenClick(Sender: TObject);
108:    procedure FormCreate(Sender: TObject);
109:    procedure BtGeneriereSchluesselClick(Sender: TObject);
110:    procedure BtRandomPrimzahlenClick(Sender: TObject);
111:    procedure BtEncryptClick(Sender: TObject);
112:    procedure BtDecryptClick(Sender: TObject);
113:    procedure FormClose(Sender: TObject; var Action: TCloseAction);
114:    procedure MenuPrimzahllisteladenClick(Sender: TObject);
115:    procedure GenerierePrimzahl(a,b:integer);//Prozedur zur Primzahlgenerierung.
116:
117:    private
118:        Primzahlen: TStringlist; // speichert alle generierten Primzahlen.
119:        tempX: String; // dient zur Zwischenspeicherung des Ver- und Entschluesselungsprodukt.
120:        Klartext, Geheimtext: array of integer; // speichert die normalen und verschlüsselten Zahl
enwerte des Klartes.
121:        laenge: integer; // speichert die Arraylaenge (length(Klartext)), für jedes Zeichen ein El
ement des Arrays.
122:
123:        { Private-Deklarationen }
124:    public
125:        { Public-Deklarationen }
126:    end;
127:
128: var
129:     Form1: TForm1;
130:
131: implementation
132:
133: {$R *.dfm}
134:
135: //=====Variablenklassen=====
136:
137: Type
138:     TDrei = Record
139:         d, x, y: Integer;
140:     end;
```

```
141:
142: TAusgabe = Record // Variablenklasse für RSA Schlüssel.
143:   n, e, d, phi: Integer;
144: end;
145:
146: //=====Functions=====
147:
148: Procedure TForm1.GenerierePrimzahl(a,b:integer); // Primzahlgenerator.
149: var
150:   zahlen: array of Boolean; //Primzahl oder nicht !?
151:   i, j, k, PrimeCount: integer;
152: begin
153:   MPrimzahlen.Text:='';
154:   Primzahlen.Clear;
155:   if (b < 0) then
156:   begin
157:     exit;
158:   end;
159:
160:   SetLength(zahlen, b + 1);
161:
162:   for i := a to b do
163:     zahlen[i] := True;
164:   for i := a to b do
165:   begin
166:     if zahlen[i] then
167:     begin
168:       for j := 2 to round(sqrt(b)) do // Wurzel b effizienter da weniger schleifendurchläufe
169:       begin
170:         k := j + j;
171:         while k <= b do
172:         begin
173:           zahlen[k] := False;
174:           k := k + j;
175:         end;
176:       end;
177:     end;
178:   end;
179:
180:   PrimeCount := 0; // Array gibt an welche Zahl (i) eine Primzahl war
181:   for i := a to b do
182:   begin
183:     if zahlen[i] then
184:     begin
185:       MPrimzahlen.Text := MPrimzahlen.Text +IntToStr(i) + ' ';
186:       Primzahlen.Add(inttostr(i));
187:       inc(PrimeCount);
188:     end;
189:   end;
190:   MPrimzahlen.Lines.Add(inttostr(PrimeCount)+' Primzahlen gefunden. ');
191:   MPrimzahlen.Lines.Add('Bitte wählen Sie zwei Primzahlen als "p" und "q". ');
192: end;
193:
194: Function Drei(d, x, y: integer): TDrei; // Weißt drei Werte der Variablenklasse
195: begin // TDrei zu, damit diese im ganzen
196:   result.d := d; // Programm leicht Abrufbar sind.
197:   result.x := x;
198:   result.y := y;
199: end;
200:
201: Function PrimTest(Value: integer): Boolean;
202: var i, t: integer;
203: begin
204:   t := round(sqrt(Value));
205:   for i := 2 to t do // Primzahltest: Teilt den zu testenden Wert
206:     if Value mod i = 0 then // (Value) durch alle Zahlen von 2 bis
207:     begin // Wurzel aus Value, da nur diese Zahlen als
208:       result := false; // mögliche Teiler in Frage kommen.
209:       exit;
210:     end;
211:   result := true;
```

```

212: end;
213:
214: Function XhochYmodZ(x, y, z: integer): integer; //RSA-Algorithmus.
215: var i, e: integer;
216: begin
217:   e := 1;
218:   for i := 1 to y do // (x^y) mod z = e.
219:     e := (e*x) mod z;
220:   result := e; // e wird als Ergebnis der Funktion zurückgegeben.
221: end;
222:
223: Function Euklid(a, b: integer): integer;
224: var i: integer;
225: begin
226:   if a < b then // Euklidscher Algorithmus.
227:     begin
228:       i := a;
229:       a := b;
230:       b := i;
231:     end;
232:   if b = 0 then
233:     result := 0
234:   else
235:     begin
236:       i := a mod b;
237:       if i = 0 then
238:         result := b
239:       else
240:         result := Euklid(b, i);
241:     end;
242:   end;
243:
244: Function ErweiterterEuklid(a, b:integer): TDrei;
245: var t: integer;
246:     temp: TDrei;
247: begin // Erweiterter Euklidscher Algorithmus.
248:   if a < b then
249:     begin
250:       t := a;
251:       a := b;
252:       b := t;
253:     end;
254:   if b = 0 then
255:     result := Drei(a, 1, 0)
256:   else
257:     begin
258:       temp := ErweiterterEuklid(b, a mod b);
259:       result := Drei(temp.d, temp.y, temp.x - (a div b)*temp.y);
260:     end;
261:   end;
262:
263: Function Schluesselgenerierung(p, q: integer): TAusgabe;
264: var e, phi: integer;
265:     b: boolean; // Generiert durch die Eulersche phi-Funktion,
266:     temp: TDrei; // den Satz des Euklid und den Erweiterten
267: begin // Satz des Euklid die Schlüsselpaare n, e, d
268:   result.n := p*q; // und gibt diese in der Variablen Klasse
269:   phi := (p-1)*(q-1); // TAusgabe aus.
270:   result.phi := phi;
271:   e := -1;
272:   b := true;
273:   while b do
274:     begin
275:       e := random(phi-3) + 2;
276:       if Euklid(phi, e) = 1 then
277:         b := false;
278:     end;
279:     result.e := e;
280:     temp := ErweiterterEuklid(e, phi);
281:     if temp.y < 0 then
282:       result.d := phi + temp.y

```

```
283:     else
284:         result.d := temp.y;
285: end;
286:
287: //=====
288: //=====
289: //-----Hauptprogramm-----
290: //=====
291: procedure TForm1.BtGenerierePrimzahlenClick(Sender: TObject); // Generiert Primzahlen.
292: begin
293:     GenerierePrimzahl(strtoint(EPrimzahlenvon.Text), strtoint(EPrimzahlenbis .Text));
294: end; // zwischen " und ".
295:
296: procedure TForm1.BtRandomPrimzahlenClick(Sender: TObject);
297: begin
298:     if Primzahlen.Count>0 then // Testet angegebene ob Primzahlen
299:     begin // echte Primzahlen sind.
300:         EPrimzahlp.Text := Primzahlen.Strings[random(Primzahlen.Count)];
301:         EPrimzahlq.Text := Primzahlen.Strings[random(Primzahlen.Count)];
302:     end;
303: end;
304:
305: procedure TForm1.BtEncryptClick(Sender: TObject); // Verschlüsseln.
306: var i, j, m: integer;
307: begin
308:     if MKlartext.Text = '' then
309:     begin
310:         Beep;
311:         exit;
312:     end
313:     else
314:     begin
315:         MGeheimtext.Lines.Add('');
316:         BtDecrypt.Enabled := true;
317:         laenge := length(MKlartext.Text);
318:         SetLength(Klartext, laenge);
319:         SetLength(Geheimtext, laenge);
320:         tempX := '';
321:         for i := 0 to laenge-1 do // Erklärung in Dokumentation.
322:         begin
323:             m := ord(MKlartext.Text[i+1]);
324:             j := XhochYmodZ(m, strtoint(ESchluessele.Text), strtoint(ESchluesseln.Text));
325:             Geheimtext[i] := j;
326:             tempX := tempX + inttostr(j);
327:         end;
328:         MGeheimtext.Lines.Add('Verschlüsselte Nachricht: '+tempX);
329:     end;
330: end;
331:
332: procedure TForm1.BtDecryptClick(Sender: TObject); // Entschlüsseln.
333: var i, j, m: integer;
334: begin
335:     if MGeheimtext.Text = '' then
336:     begin
337:         Beep;
338:         exit;
339:     end
340:     else
341:     begin
342:         MKlartext.Lines.Add('');
343:         tempX := '';
344:         for I := 0 to laenge-1 do // Erklärung in Dokumentation.
345:         begin
346:             m := Geheimtext[i];
347:             j := XhochYmodZ(m, strtoint(ESchluessele2.Text), strtoint(ESchluesseln2.Text));
348:             tempX := tempX + chr(j);
349:         end;
350:         MKlartext.Lines.Add('Entschlüsselte Nachricht: '+tempX);
351:     end;
352: end;
353:
```

```
354: procedure TForm1.BtGeneriereSchluesselClick(Sender: TObject); // Generiert Schlüssel.
355: var schluessel:TAusgabe;
356: begin
357:   if PrimTest(strtoint(EPrimzahlp.Text)) and PrimTest(strtoint(EPrimzahlq.Text)) then
358:   begin
359:     schluessel := Schluesselgenerierung(strtoint(EPrimzahlp.Text), strtoint(EPrimzahlq.Text));
360:     ESchluesslele.Text := inttostr(schluessel.e); // Schlüssel werden ausgegeben.
361:     ESchluesseln.Text := inttostr(schluessel.n);
362:     ESchluesseld.Text := inttostr(schluessel.d);
363:     ESchluesseln2.Text := inttostr(schluessel.n);
364:   end;
365:   Label12.Caption := EPrimzahlp.Text;
366:   Label13.Caption := EPrimzahlq.Text;
367:   Label14.Caption := inttostr(schluessel.d);
368:   Label15.Caption := inttostr(schluessel.n);
369:   Label16.Caption := inttostr(schluessel.phi);
370: end;
371:
372:
373:
374: //=====Diverses=====
375: procedure TForm1.Exit1Click(Sender: TObject);
376: begin
377:   Form1.close;
378: end;
379:
380: procedure TForm1.FormClose(Sender: TObject; var Action: TCloseAction);
381: begin
382:   Primzahlen.Free; // Gibt den Speicher, der der Stringlist zugewiesen wurde
383: end; // wieder frei.
384:
385: procedure TForm1.FormCreate(Sender: TObject); // Programmstart.
386: begin
387:   Form1.Height := 685; // Initialisierung des Fensters/Programms.
388:   Form1.Width := 959;
389:   MPrimzahlen.Clear;
390:   MPrimzahlen.Lines.Add('Am besten verwendet man den Primzahlenbereich von 1 bis 100. Bei zu g
roßen Zahlen scheitert die Ent- bzw. Verschlüsselung !');
391:   MKlartext.Clear;
392:   MGeheimtext.Clear;
393:   Primzahlen := TStringlist.Create;
394:   tempX := '';
395: end;
396:
397: procedure TForm1.InfolClick(Sender: TObject); // Infofenster.
398: begin
399:   ShowMessage('Programm zur Verdeutlichung des RSA_Algorithmus' + #13#10 +
400:   ' © Benedikt Welp' + #13#10 +
401:   'erstellt mit BORLAND® DELPHI® 2005 PERSONAL ');
402: end;
403:
404: procedure TForm1.MenuPrimzahllisteladenClick(Sender: TObject);
405: begin
406:   if OpenFileDialog1.Execute then // Primzahlliste laden.
407:   begin
408:     Primzahlen.Clear;
409:     Primzahlen.LoadFromFile(OpenFileDialog1.FileName);
410:     MPrimzahlen.Lines.Add('Primzahlen erfolgreich aus Datei geladen. ('+
411:     OpenFileDialog1.FileName+')');
412:   end;
413: end;
414:
415: end.
```