---

1)

```
procedure TForm1.Button2Click(Sender: TObject);
var
   M:set of char;
   ch:char;
   S:string;
   i,k:integer;
begin
   M:=[];
   k:=0;
   for i:=i to ListBox1.Items.Count-1 do
   begin
     S:=S+Listbox1.Items[i];
     for i:=1 to Length(S) do
        if S[i] in M then
           M:=M+[S[i]]
        else
           if M>5 then
           begin
              ShowMessage('Impossible')
           end;
   end;
end;
```

---

2)

```
procedure SetDisplay(R: Real);
var
   S: string[63];
begin
   Str(R: 0: 10, S);
   if S[1] <> '-' then
      Sign := ' '
   else
   begin
      Delete(S, 1, 1);
      Sign := '-';
   end;
   if Length(S) > 15 + 1 + 10 then
      Error
   else
   begin
      while S[Length(S)] = '0' do
         Dec(S[0]);
      if S[Length(S)] = '.' then
         Dec(S[0]);
      Number := S;
   end;
end;
```

---

3)

```
procedure TCalcDisplay.Draw;
var
    Color: Byte;
    I: Integer;
    B: TDrawBuffer;
begin
    Key := UpCase(Key);
    if(Status=csError) and (Key<>'C') then
       Key:=' ';
    Color := GetColor(1);
    I := Size.X - Length(Number) - 2;
    MoveChar(B, ' ', Color, Size.X);
    MoveChar(B[I], Sign, Color, 1);
    MoveStr(B[I + 1], Number, Color);
    WriteBuf(0, 0, Size.X, 1, B);
end;
```

---

4)

```
procedure TCalcDisplay.HandleEvent(var Event: TEvent);
begin
   inherited HandleEvent(Event);
   case Event.What of
      evKeyDown:
        begin
          CalcKey(Event.CharCode);
          ClearEvent(Event);
        end;
      evBroadcast:
        if Event.Command = cmCalcButton then
        begin
          CalcKey(PButton(Event.InfoPtr)^.Title^[1]);
          ClearEvent(Event);
        end;
   end;
end;
```

---

5)

```
constructor TCalculator.Init;
const
    KeyChar: array[0..19] of Char = 'C'#27'%'#241'789/456*123-0.=+';
var
    I: Integer;
    P: PView;
    R: TRect;
begin
    R.Assign(5, 3, 29, 18);
    inherited Init(R, 'Calculator');
    Options := Options or ofFirstClick;
    for I := 0 to 19 do
    begin
       P:=New(PButton, Init(R, KeyChar[I],cmCalcButton, bfNormal + bfBroadcast));
       P^.Options := P^.Options and not ofSelectable;
       Insert(P);
    end;
    R.Assign(3, 2, 21, 3);
    Insert(New(PCalcDisplay, Init(R)));
end;
```

---

6)

```
while Tmp<>nil do
begin
  If  Tmp^.Num = StrToInt(S) then
  begin
     i := i+1;
     Summ := Summ + Tmp^.Mark
  end;
  tmp:=Tmp^.Next
end;
j:= Summ/i;
Edit5.Text := FloatToStr(j)
end;
```

---

7)

```
procedure TForm1.Button3Click(Sender: TObject);
var
    i:extended;
    R:rec;
    kol:integer;
begin
   if (CEdit1.Text='') and (Kedit2.Text='') then
   begin
      ShowMessage('Please enter data');
      exit
   end
   else
     Reset(F);
   i:=StrTofloat(CEdit1.Text);
   kol:=0;
   while not EoF(F) do
   begin
      Read(F,R);
      if R.Price < i then
         kol:=kol+1;
   end;
   KEdit2.Text:=IntToStr(kol)
end;
```

---

8)

```
function StdEditorDialog(Dialog: Integer; Info: Pointer): Word;
var
   R: TRect;
   T: TPoint;
begin
   case Dialog of
        edReplace: StdEditorDialog :=
              Application^.ExecuteDialog(CreateReplaceDialog,Info);
        edReplacePrompt:
        begin
           {Avoid placing the dialog on the same line as the cursor}
           R.Assign(0, 1, 40, 8);
           R.Move((Desktop^.Size.X - R.B.X) div 2, 0);
           if TPoint(Info).Y <= T.Y then
              R.Move(0, Desktop^.Size.Y - R.B.Y - 2);
           StdEditorDialog := MessageBoxRect(R, 'Replace this occurence?',nil,
                                             mfYesNoCancel + mfInformation);
        end;
   end;
end;
```

---

9)

```
function TIndicator.GetPalette: PPalette;
const
   P: string[Length(CIndicator)] = CIndicator;
begin
   R.A.X := (I mod 4) * 5 + 2;
   R.A.Y := (I div 4) * 2 + 4;
   case Operator of
          '+', '-': R := Operand * R / 100;
          '*', '/': R := R / 100;
   end;
   GetPalette := @P;
end;
```

```
asm     LES     DI,Buf
        MOV     CX,Count
        XOR     DX,DX
        MOV     AL,0DH
        CLD
@@1:    JCXZ    @@2
        REPNE   SCASB
        JNE     @@2
end;
```

---

10)

```
procedure TIndicator.SetValue(ALocation: TPoint; AModified: Boolean);
begin
   case Operator of
            '+': SetDisplay(Operand + R);
            '-': SetDisplay(Operand - R);
            '*': SetDisplay(Operand * R);
            '/': if R = 0 then Error else SetDisplay(Operand / R);
   end;
   if (Longint(Location) <> Longint(ALocation)) or (Modified <> AModified) then
   begin
      Location := ALocation;
      Modified := AModified;
      DrawView;
   end;
end;
```

---

11)

```
procedure TIndicator.Draw;
var
   Color: Byte;
   Frame: Char;
   L: array[0..1] of Longint;
   S: String[15];
   B: TDrawBuffer;
begin
   if State and sfDragging = 0 then
   begin
      Color := GetColor(1);
      Frame := #205;
   end
   else
      MoveChar(B, Frame, Color, Size.X);
   if Modified then
      WordRec(B[0]).Lo := 15;
   FormatStr(S, ' %d:%d ', L);
   MoveStr(B[8 - Pos(':', S)], S, Color);
   WriteBuf(0, 0, Size.X, 1, B);
end;
```

---

12)

```
procedure TForm1.BitBtn2Click(Sender: TObject);
var
   S:string;
   i,n:integer;
   code:byte;
begin
    S:=ListBox1.Items[0];
    n:=Length(S);
    for i:=1 to n do
    begin
       if S[i] in ['A'..'Z'] then
       begin
          code:=ord(S[i]);
          Inc(code,2);
          S[i]:=Chr(code)
       end
   end;
end;
```

---

13)

```
constructor TEditor.Load(var S: TStream);
begin
   inherited Load(S);
   GetPeerViewPtr(S, HScrollBar);
   GetPeerViewPtr(S, VScrollBar);
   GetPeerViewPtr(S, Indicator);
   S.Read(BufSize, SizeOf(Word));
   S.Read(CanUndo, SizeOf(Boolean));
   InitBuffer;
   if Buffer <> nil then
      IsValid := True
   else
   begin
      EditorDialog(edOutOfMemory, nil); BufSize := 0;
   end;
   SetBufLen(0);
end;
```

---

14)

```
procedure TEditor.DrawLines(Y, Count: Integer; LinePtr: Word);
var
   Color: Word;
   B: array[0..MaxLineLength - 1] of Word;
begin
   Color := GetColor($0201);
   while Count > 0 do
   begin
      FormatLine(B, LinePtr, Delta.X + Size.X, color);
      WriteBuf(0, Y, Size.X, 1, B[Delta.X]);
      LinePtr := NextLine(LinePtr);
      Inc(Y);
      Dec(Count);
   end;
end;
```

---

15)

```
procedure TEditor.Find;
var  FindRec: TFindDialogRec;
begin
   with FindRec do
   begin
      Find := FindStr;
      Options := EditorFlags;
      if EditorDialog(edFind, @FindRec) <> cmCancel then
      begin
         FindStr := Find;
         EditorFlags := Options and not efDoReplace;
         DoSearchReplace;
      end;
   end;
end;
```

---

16)

```
function TEditor.LineMove(P: Word; Count: Integer): Word;
var
   Pos: Integer;
   I: Word;
begin
   I := P;
   P := LineStart(P);
   Pos := CharPos(P, I);
   while Count <> 0 do
   begin
      I := P;
      if Count < 0 then
      begin
         P := PrevLine(P);
         Inc(Count);
      end
      else
      begin
         P := NextLine(P);
         Dec(Count);
      end;
   end;
   if P <> I then
      P := CharPtr(P, Pos);
   LineMove := P;
end;
```

---

17)

```
procedure TEditor.NewLine;
const
   CrLf: array[1..2] of Char = #13#10;
var
   I, P: Word;
begin
   P := LineStart(CurPtr);
   I := P;
   while (I < CurPtr) and ((Buffer^[I] = ' ') or (Buffer^[I] = #9)) do
      Inc(I);
   InsertText(@CrLf, 2, False);
   if AutoIndent then
      InsertText(@Buffer^[P], I - P, False);
end;
```

---

18)

```
procedure TEditor.SetState(AState: Word; Enable: Boolean);
begin
   inherited SetState(AState, Enable);
   case AState of
     sfActive:
        begin
          if HScrollBar <> nil then HScrollBar^.SetState(sfVisible, Enable);
          if VScrollBar <> nil then VScrollBar^.SetState(sfVisible, Enable);
          if Indicator <> nil then Indicator^.SetState(sfVisible, Enable);
          UpdateCommands;
        end;
     sfExposed:
        if Enable then Unlock;
   end;
end;
```

---

19)

```
procedure TForm1.Button2Click(Sender: TObject);
begin
   if  Form1.Button2.Caption='Cancel' Then
   begin
      Form1.Button2.Caption:='Previous';
      Form1.Button1.Caption:='Next'
   end;
   if FilePos(F)>0 then
   begin
      Seek(F,FilePos(F)-1);
      ReadRec
   end;
end;
```

---

20)

```
procedure TEditor.DoSearchReplace;
var
   I: Word;
   C: TPoint;
begin
   repeat
      if not Search(FindStr, EditorFlags) then
      begin
         I := cmYes;
         if EditorFlags and efPromptOnReplace <> 0 then
            if I = cmYes then
            begin
               InsertText(@ReplaceStr[1], Length(ReplaceStr), False);
               TrackCursor(False);
            end;
      end;
   until (I = cmCancel)or (EditorFlags and efReplaceAll = 0);
end;
```

---

21)

```
procedure TEditor.DeleteRange(StartPtr, EndPtr: Word; DelSelect: Boolean);
begin
   if HasSelection and DelSelect then
      DeleteSelect
   else
   begin
      CursorVisible:=(CurPos.Y >= Delta.Y) and (CurPos.Y<Delta.Y+Size.Y);
      SetSelect(CurPtr, EndPtr, True);
      DeleteSelect;
      SetSelect(StartPtr, CurPtr, False);
      DeleteSelect;
   end;
end;
```

---

22)

```
procedure TEditor.ConvertEvent(var Event: TEvent);
var
   ShiftState: Byte absolute $40:$17;
   Key: Word;
begin
   if Event.What = evKeyDown then
   begin
      if KeyState <> 0 then
         Key := ScanKeyMap(KeyMap[KeyState], Key);
      KeyState := 0;
      if Key <> 0 then
         if Hi(Key) = $FF then
         begin
            KeyState := Lo(Key);
            ClearEvent(Event);
         end
      else
      begin
         Event.What := evCommand;
         Event.Command := Key;
      end;
   end;
end;
```

---

23)

```
function TEditor.CharPtr(P: Word; Target: Integer): Word;
var
   Pos: Integer;
begin
   Pos := 0;
   while (Pos < Target) and (P< BufLen) and (BufChar(P) <> #13) do
   begin
      if BufChar(P) = #9 then Pos := Pos + 7;
      Inc(Pos);
      Inc(P);
   end;
   if Pos > Target then Dec(P);
   CharPtr := P;
end;
```

---

24)

```
procedure TEditor.DoUpdate;
begin
   if UpdateFlags <> 0 then
   begin
      SetCursor(CurPos.X - Delta.X, CurPos.Y - Delta.Y);
      if UpdateFlags and ufView <> 0 then
         DrawView
      else
      if UpdateFlags and ufLine <> 0 then
         DrawLines(CurPos.Y - Delta.Y, 1, LineStart(CurPtr));
      if HScrollBar <> nil then
         HScrollBar^.SetParams(Delta.X, 0, Limit.X - Size.X, Size.X div 2, 1);
      if Indicator <> nil then
         Indicator^.SetValue(CurPos, Modified);
      if State and sfActive <> 0 then
         UpdateCommands;
      UpdateFlags := 0;
   end;
end;
```

---

25)

```
procedure TEditor.ScrollTo(X, Y: Integer);
begin
   CheckFirst;
   if Length(Number) = 1 then Number := '0'
   else Dec(Number[0]);
end;

asm
        MOV     AX,X
        CMP     AX,Y
        JLE     @@1
        MOV     AX,Y
@@1:
   X := Max(0, Min(X, Limit.X - Size.X));
   Y := Max(0, Min(Y, Limit.Y - Size.Y));
   if (X <> Delta.X) or (Y <> Delta.Y) then
   begin
      Delta.X := X;
      Delta.Y := Y;
      Update(ufView);
   end;
end;
```