1. Competency
   After practicing, students will have competencies: knowing, understanding and applying IDE (Integrated Development Environment) Delphi in computer programming.

2. Sub Competencies
   After practicing, students will have sub competencies:
   a. Knowing parts of IDE Delphi.
   b. Understand function and procedure of each part of IDE Delphi.
   c. Applying IDE Delphi to make a simple program.

3. Introduction
   3.1. Knowing IDE Delphi
   IDE Delphi consists of 6 main parts, which are: Menu, Speed Bar, Component Palette, Form Designer, Code Editor and Object Inspector.

   ![Diagram of IDE Delphi parts: Speed Bar, Menu, Component Palette, Object Inspector, Code Editor, Form Designer]
a. Main Menu
Delphi Main Menu have similar function memiliki kegunaan seperti menu pada aplikasi Windows lainnya. From this menu, you can open or save a program, running and searching program, etc. Shortly, everything that connected with IDE Delphi can be called via main menu.

File Edit Search View Project Run Component Database Tools Window Help

b. Speed Bar
Speed Bar or toolbar contain set of button that have same function with mostly item menu.

c. Component Palette
Component Palette contains set of icon that represents components of VCL (Visual Component Library). VCL is component that can be used to build an application.

d. Form Designer
Form Designer is a place to design windows application. The Design is placement of components from Component Palette.
e. Code Editor
Code Editor is place to write program in Pascal Language. Code editor has a Code Inspector, so that we don’t have to write whole instruction. Delphi has write a basic structure of a program. Look at the picture below:
f. Object Inspector

The Object Inspector is the connection between your application’s visual appearance and the code that makes your application run. The Object Inspector enables you to:
- set design-time properties for components you have placed on a form (or for the form itself).
- Create and help you navigate through event handlers.
- Filter visible properties and events.

The Object Inspector has two pages:
- Properties page
- Events page

Object Inspector tabs provide a means to switch between the Property page and the Events page of the Object Inspector. To change pages, click the Properties or Events tab.
You can display and filter properties and events by category. By filtering the properties, you can reduce the number of properties visible in the Object Inspector and focus on those which are primarily of interest at the time. You can also more easily locate related properties by viewing them by category. For example, when localizing your application for other countries, you can display only properties that need to be localized by unchecking all categories except Localizable. See Property and event categories in the Object Inspector.

The Object Inspector also includes expanded inline component references that display the properties and events of a referenced component (or a subcomponent) without having to actually select that component.

You can control the colors in which items appear in the Object Inspector and change options that allow you to display an instance list, a status bar, and show read-only properties. To change these settings for the Object Inspector, choose Tools|Environment Options, and click the Object Inspector tab or right-click the Object Inspector and choose Properties.

4. Tools and Materials
- Personal Computer
- Windows XP Operating System
- Borland Delphi 7

5. Work Safety
a. Before practicing, students should understand the goal and competency of this practice.
b. Read and understand all procedures of this practice carefully.
c. Always keep data from losing causes by extinguished electricity.

6. Procedures

6.1. Creating Simple Program
1. Run Borland Delphi 7 on your computer. Click menu start – All Programs – Borland Delphi 7 – Delphi 7.
2. IDE Delphi will look like this:
3. In the active Form, choose component Button1 on Tab Standard and place it on form just like picture below:

![Button1](image)

4. Set each component’s properties by selecting component first, then choose properties on Object Inspector, with the value below.

<table>
<thead>
<tr>
<th>Komponen</th>
<th>Properties</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Form1</td>
<td>Caption</td>
<td>Praktik Membuat Program Sederhana</td>
</tr>
<tr>
<td>Button1</td>
<td>Caption</td>
<td>Klik Saya !</td>
</tr>
</tbody>
</table>

5. Then the design will look like this:

![Form Design](image)
6. Fill program code of Button1 in two ways: a) double click on Button1, or b) select Button1, select Events on Object Inspector, then double click item OnClick. Write these code:

```pascal
procedure TForm1.Button1Click(Sender: TObject);
begin
  Application.MessageBox('Halo, Saya baru saja diklik', 'Informasi', MB_OK or MB_ICONEXCLAMATION);
end;
```

7. Project is ready to run in three ways: a) Click toolbar Run | Run, or c) push button F9 on keyboard.

8. After program running, click on button 'Klik Saya !', then a message box will showed:

![Message Box](image)

9. Save your project by selecting menu File | Save All. Delphi will show a dialog box and asking you to insert the name of unit in your form. Write ULatihan1.pas and click OK. Delphi will again show dialog box and asking you to insert the name of your project. Fill it with PLatihan1.dpr and click OK.
Semester 2
Understand IDE Delphi and Program Algorithm
Classical hours
4 x 50 mins

Created by: Prohibited to reproduce in part or whole document without permission of the Faculty of Engineering, Yogyakarta State University
Reviewed by:
10. Always try to save unit file begin with letter U________.pas and save project file begin with letter P________.dpr. It is useful to distinguish between unit file with project file.

11. Saved project will creating a lot of files, but 3 main files are:
   - PLatihan1.dpr, contain main program of your application.
   - ULatihan.pas, unit that will handle events on form.
   - ULatihan.dfm, list of components and their properties in the form(s).

7. Discussion
   a. Create a program with 2 buttons. Change property Caption of those buttons become 'Biru' dan 'Merah'. The request are, 'Biru', then form color will change into blue and if we click 'Merah' button, then form color will change into red.
   Running program will look like this picture:

   ![Form with two buttons](image1)

   b. Put three Tbutton components and a Tedit component on a form. When Button1 pressed, text on Edit1 will change into 'Pemrograman'. When Button2 pressed, text on Edit1 will change into 'Komputer'. Last, when Button3 pressed, text on Edit1 will be empty.
   Running program will look like this picture:

   ![Form with edit and buttons](image2)
1. Competency
   After practicing, students will have competencies: know, understand and applying Data Type and application in Programming.

2. Sub Competencies
   After practicing, students will have sub competencies:
   a. Knowing parts of data type.
   b. Understand function and procedure of data type.
   c. Applying data type in programming.

3. Introduction
   3.1. Variable
   We often use variables in computer programming. Variables are simply a place to keep data.

   Syntax of variable declaration on Delphi is:

   ```delphi
   var
       nama_variabel: tipe_variabel;
   ```

   example:
   ```delphi
   var
       i: integer;
   ```

   To declare several variables with different data type, we only have to write keyword `var` once, like this:
   ```delphi
   var
       i: integer;
       j: real;
       k: char;
   ```

   Here another sample of several variables with same data types:
   ```delphi
   var
       i, j, k: integer;
       a, b, c: char;
   ```
Variable name on Delphi must follow these rules:

- Name of variable maximum consist of 63 characters. If variable has more than 63 characters, compiler will abandon the remainder character.
- Name of variable can only contain characters, numbers, underscores, and must not begin with number.
- Keyword of Delphi cannot be a variable name, for example `if`, `else`, `for`, etc.

3.2. Data Type

A type is essentially a name for a kind of data. When you declare a variable you must specify its type, which determines the set of values the variable can hold and the operations that can be performed on it. Every expression returns data of a particular type, as does every function. Most functions and procedures require parameters of specific types. In Delphi, it consists of:

- Integer data types.
- Real data types.
- Boolean data types.
- Character data types.
- String data types.

a. Integer types

An integer type represents a subset of the whole numbers. The generic integer types are Integer and Cardinal; use these whenever possible, since they result in the best performance for the underlying CPU and operating system. The table below gives their ranges and storage formats for the current 32-bit Delphi compiler.

<table>
<thead>
<tr>
<th>Fundamental integer types</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
</tr>
<tr>
<td>Shortint</td>
</tr>
<tr>
<td>Smallint</td>
</tr>
<tr>
<td>Longint</td>
</tr>
<tr>
<td>Byte</td>
</tr>
<tr>
<td>Word</td>
</tr>
</tbody>
</table>
b. Real types

A real type defines a set of numbers that can be represented with floating-point notation. The table below gives the ranges and storage formats for the fundamental real types.

<table>
<thead>
<tr>
<th>Type</th>
<th>Size in Bit</th>
<th>Range/Format</th>
<th>Significant Digit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Real</td>
<td>6 bit</td>
<td>2.9<em>10^-39 .. 1.7</em>10^38</td>
<td>11-12</td>
</tr>
<tr>
<td>Single</td>
<td>4 bit</td>
<td>1.5<em>10^-45 .. 3.4</em>10^38</td>
<td>7-8</td>
</tr>
<tr>
<td>Double</td>
<td>8 bit</td>
<td>5.0<em>10^-324 .. 1.7</em>10^308</td>
<td>15-16</td>
</tr>
<tr>
<td>Extended</td>
<td>10 bit</td>
<td>3.4<em>10^-4932 .. 1.1</em>10^4932</td>
<td>19-20</td>
</tr>
<tr>
<td>Comp</td>
<td>8 bit</td>
<td>-2^63 +1 .. 2^63 -1</td>
<td>19-20</td>
</tr>
<tr>
<td>Currency</td>
<td>8 bit</td>
<td>-922337203685477.5808 .. 922337203685477.5807</td>
<td>19-20</td>
</tr>
</tbody>
</table>

c. Boolean types

The four predefined Boolean types are Boolean, ByteBool, WordBool, and LongBool. Boolean is the preferred type. The others exist to provide compatibility with other languages and operating system libraries.

A Boolean variable occupies one byte of memory, a ByteBool variable also occupies one byte, a WordBool variable occupies two bytes (one word), and a LongBool variable occupies four bytes (two words).

Boolean values are denoted by the predefined constants True and False. The following relationships hold.
d. Character types

The fundamental character types are AnsiChar and WideChar. AnsiChar values are byte-sized (8-bit) characters ordered according to the locale character set which is possibly multibyte. AnsiChar was originally modeled after the ANSI character set (thus its name) but has now been broadened to refer to the current locale character set.

WideChar characters use more than one byte to represent every character. In the current implementations, WideChar is word-sized (16-bit) characters ordered according to the Unicode character set (note that it could be longer in future implementations). The first 256 Unicode characters correspond to the ANSI characters.

Note

On Linux, wchar_t widechar is 32 bits per character. The 16-bit Unicode standard that Delphi WideChars support is a subset of the 32-bit UCS standard supported by Linux and the GNU libraries. Delphi WideChar data must be widened to 32 bits per character before it can be passed to an OS function as wchar_t.

The generic character type is Char, which is equivalent to AnsiChar. Because the implementation of Char is subject to change, it's a good idea to use the standard function SizeOf rather than a hard-coded constant when writing programs that may need to handle characters of different sizes.

A string constant of length 1, such as 'A', can denote a character value. The predefined function Chr returns the character value for any integer in the range of AnsiChar or WideChar; for example, Chr(65) returns the letter A.

Character values, like integers, wrap around when decremented or incremented past the beginning or end of their range (unless range-checking is enabled). For example, after execution of the code

```pascal
var
  Letter: Char;
  I: Integer;
begin
```
Letter := High(Letter);
for I := 1 to 66 do
  Inc(Letter);
end;

Letter has the value A (ASCII 65).

e. String types
A string represents a sequence of characters. Delphi supports the following predefined string types.

<table>
<thead>
<tr>
<th>String types</th>
<th>Maximum length</th>
<th>Memory required</th>
<th>Used for</th>
</tr>
</thead>
<tbody>
<tr>
<td>ShortString</td>
<td>255 characters</td>
<td>2 to 256 bytes</td>
<td>backward compatibility</td>
</tr>
<tr>
<td>AnsiString</td>
<td>~2^31 characters</td>
<td>4 bytes to 2GB</td>
<td>8-bit (ANSI) characters, DBCS ANSI, MBCS ANSI, etc.</td>
</tr>
<tr>
<td>WideString</td>
<td>~2^30 characters</td>
<td>4 bytes to 2GB</td>
<td>Unicode characters; multi-user servers and multi-language applications</td>
</tr>
</tbody>
</table>

AnsiString, sometimes called the long string, is the preferred type for most purposes.

String types can be mixed in assignments and expressions; the compiler automatically performs required conversions. But strings passed by reference to a function or procedure (as var and out parameters) must be of the appropriate type. Strings can be explicitly cast to a different string type (see Typecasts).

The reserved word string functions like a generic type identifier. For example,

    var S: string;

creates a variable S that holds a string. In the default {$H+} state, the compiler interprets string (when it appears without a bracketed number after it) as AnsiString. Use the {$H-} directive to turn string into ShortString.

The standard function Length returns the number of characters in a string. The SetLength procedure adjusts the length of a string. Comparison of strings is defined by the ordering of the characters in corresponding positions. Between strings of unequal length, each character in the longer string without a corresponding character in the shorter string takes on a
greater-than value. For example, "AB" is greater than "A"; that is, 'AB' > 'A' returns True. Zero-length strings hold the lowest values.

You can index a string variable just as you would an array. If $S$ is a string variable and $i$ an integer expression, $S[i]$ represents the $i$th character—or, strictly speaking, the $i$th byte—in $S$. For a ShortString or AnsiString, $S[i]$ is of type AnsiChar; for a WideString, $S[i]$ is of type WideChar. For single-byte (Western) locales, $MyString[2] := 'A$; assigns the value $A$ to the second character of $MyString$. The following code uses the standard AnsiUpperCase function to convert $MyString$ to uppercase.

```pascal
var
I: Integer;
begin
I := Length(MyString);
while I > 0 do
begin
MyString[I] := AnsiUpperCase(MyString[I]);
I := I - 1;
end;
end;
```

Be careful indexing strings in this way, since overwriting the end of a string can cause access violations. Also, avoid passing long-string indexes as var parameters, because this results in inefficient code.

You can assign the value of a string constant—or any other expression that returns a string—to a variable. The length of the string changes dynamically when the assignment is made. Examples:

- $MyString := 'Hello world!$;
- $MyString := 'Hello ' + 'world$;
- $MyString := MyString + '!$;
- $MyString := '\n$;          { space }
- $MyString := ''$;           { empty string }

### 3.3. Constants

Several different language constructions are referred to as "constants". There are numeric constants (also called numerals) like 17, and string constants (also called character strings or string literals) like 'Hello world!'. Every
enumerated type defines constants that represent the values of that type. There are predefined constants like True, False, and nil. Finally, there are constants that, like variables, are created individually by declaration.

Declared constants are either true constants or typed constants. These two kinds of constant are superficially similar, but they are governed by different rules and used for different purposes.

Pendeklarasian konstanta dilakukan dengan sintaks berikut:

```plaintext
const
nameof_konstanta = valueof_konstanta;
```

every example:

```plaintext
const
    Maksimum   = 10;
    LogNatural = 2.718281828;
    Bahasa     = 'Object Pascal';
```

We could determine the type of constant. It called typed constanta.

every example:

```plaintext
const
    Maksimum   : integer = 10;
    LogNatural : double = 2.718281828;
    Bahasa     : ShortString = 'Object Pascal';
```

4. Tools and Materials
   - Personal Computer
   - Windows XP Operating System
   - Borland Delphi 7

5. Work Safety
   a. Before practicing, students should understand the goal and competency of this practice.
   b. Read and understand all procedures of this practice carefully.
   c. Always keep data from losing causes by extinguished electricity.

6. Procedures
6.1. Praktik Penggunaan Tipe Bilangan Bulat

1. Run Borland Delphi 7 on your computer.
2. In active form, select components and place on form like picture below:

3. Set the properties of each components as follow:

<table>
<thead>
<tr>
<th>Component</th>
<th>Properties</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Form1</td>
<td>Caption</td>
<td>Tipe Bilangan Bulat</td>
</tr>
<tr>
<td>Label1</td>
<td>Caption</td>
<td>x =</td>
</tr>
<tr>
<td></td>
<td>Font - Size</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>Font - Style - fsBold</td>
<td>True</td>
</tr>
<tr>
<td>Label2</td>
<td>Caption</td>
<td>y =</td>
</tr>
<tr>
<td></td>
<td>Font - Size</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>Font - Style - fsBold</td>
<td>True</td>
</tr>
<tr>
<td>Label3</td>
<td>Caption</td>
<td>x + y =</td>
</tr>
<tr>
<td></td>
<td>Font - Size</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>Font - Style - fsBold</td>
<td>True</td>
</tr>
<tr>
<td>Label4</td>
<td>Caption</td>
<td>x * y =</td>
</tr>
<tr>
<td></td>
<td>Font - Size</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>Font - Style - fsBold</td>
<td>True</td>
</tr>
</tbody>
</table>
4. The result is like picture below:

![Image of a programming interface showing input fields for x, y, and a button labeled Hitung.]

5. Double Click on Button1, then write code program as follow:

```pascal
procedure TForm1.Button1Click(Sender: TObject);
var
  x, y, z: integer;
begin
  x := StrToInt(Edit1.Text);
  y := StrToInt(Edit2.Text);
  z := x + y;
  Edit3.Text := IntToStr(z);
  z := x * y;
  Edit4.Text := IntToStr(z);
end;
```

6. Run project by clicking toolbar or press F9 on your keyboard.
7. After program running, entry the number 3 and 4 sequentially on Edit1 and Edit2, then click button Hitung. We can see that on Edit3 will show result 3 + 4 = 7 and on Edit4 will show result 3 * 4 = 12 like picture below:

![Screenshot of program with inputs 3 and 4 and outputs 7 and 12]

8. Try to entry another number on Edit1 dan Edit2 just like step 7.

9. Save your project as TipeData1.

6.2. Applying Real Type on Programming
1. Run Borland Delphi 7 on your computer.

2. In active form, select components and place on form like picture below:

![Screenshot of Delphi form with label, edit, button, and text fields]
3. Set the properties of each components as follow:

<table>
<thead>
<tr>
<th>Component</th>
<th>Properties</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Form1</td>
<td>Caption</td>
<td>Tipe Bilangan Real</td>
</tr>
<tr>
<td>Label1</td>
<td>Caption</td>
<td>Mencari Luas dan Keliling Lingkaran</td>
</tr>
<tr>
<td></td>
<td>Font - Size</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>Font - Style - fsBold</td>
<td>True</td>
</tr>
<tr>
<td>Label2</td>
<td>Caption</td>
<td>Masukkan nilai jari-jari</td>
</tr>
<tr>
<td>Label3</td>
<td>Caption</td>
<td>Luas lingkaran =</td>
</tr>
<tr>
<td>Label4</td>
<td>Caption</td>
<td>Keliling lingkaran =</td>
</tr>
<tr>
<td>Edit1</td>
<td>Text</td>
<td>&lt;empty&gt;</td>
</tr>
<tr>
<td>Edit2</td>
<td>Text</td>
<td>&lt;empty&gt;</td>
</tr>
<tr>
<td>Edit3</td>
<td>Text</td>
<td>&lt;empty&gt;</td>
</tr>
<tr>
<td>Button1</td>
<td>Caption</td>
<td>Hitung</td>
</tr>
</tbody>
</table>

4. The result is like picture below:

5. Double Click on Button1, then write code program as follow:

```pascal
procedure TForm1.Button1Click(Sender: TObject);
var
    jari_jari: real;
    luas_lingkaran: real;
    keliling_lingkaran: real;
const
    phi = 3.14;
begin
    // Code goes here
end;
```
jari_jari := StrToFloat(Edit1.Text);
luas_lingkaran := 2 * phi * jari_jari;
Edit2.Text := FloatToStr(luas_lingkaran);
kelliling_lingkaran := phi * jari_jari * jari_jari;
Edit3.Text := FloatToStr(kelliling_lingkaran);
end;

6. Run project by clicking toolbar or press F9 on your keyboard.

7. After program running, entry the number 5 on Edit1, click button Hitung, then Edit2 will show circle vast as 31.4 and Edit3 will show circle around equal to 78.5 like shown below:

8. Try to entry another number on Edit1 dan Edit2 just like step 7.

9. Save your project as TipeData2.

6.3. Applying Boolean Type on Programming

1. Run Borland Delphi 7 on your computer.

2. In active form, select components and place on form like picture below:
3. Set the properties of each components as follow:

<table>
<thead>
<tr>
<th>Komponen</th>
<th>Properties</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Form1</td>
<td>Caption</td>
<td>Tipe Data Boolean</td>
</tr>
<tr>
<td>Edit1</td>
<td>Text</td>
<td>&lt;empty&gt;</td>
</tr>
<tr>
<td>Button1</td>
<td>Caption</td>
<td>True vs False</td>
</tr>
</tbody>
</table>

4. The result is like picture below:
5. Double Click on Button1, then write code program as follow:

```pascal
procedure TForm1.Button1Click(Sender: TObject);
begin
  if (gantian = False) then
    begin
      gantian := True;
      Edit1.Text := 'Nilai Boolean = True';
    end
  else
    begin
      gantian := False;
      Edit1.Text := 'Nilai Boolean = False';
    end;
end;
```

6. Add variable `gantian` on global variable declaration as follow:

```pascal
var
  Form1: TForm1;
  gantian: boolean;
```

7. Run project by clicking toolbar or press F9 on your keyboard.

8. After program running, press the button once, then edit1 will show message 'Nilai Boolean = True', press again the button, edit1 will show message 'Nilai Boolean = False', etc. this will happen everytime you press the button.
9. Save your project as TipeData3.

7. Discussion
   a. Create a program to count resistance of two resistor installed both serial and parallel with configuration just like picture below:

   ![Program window](image)

   Resistor 1 = 10000 ohm
   Resistor 2 = 10000 ohm

   RI seri R2 = 20000 ohm
   R1 paralel R2 = 5000 ohm

   b. Create a program to perform temperatur conversion from Celcius to Farenheit with formula \( F = \frac{9}{5} \times C + 32 \), and from Fahrenheit to Celcius with formula \( C = \frac{5}{9} \times (F - 32) \).
1. **Competency**
   After practicing, students will have competencies: know, understand and applying branch structure in Programming.

2. **Sub Competencies**
   After practicing, students will have sub competencies:
   a. Knowing concept of branch structure.
   b. Understand function and procedure of branch structure.
   c. Creating computer program with branch structure.

3. **Introduction**
   Conditional or branch structure used to turn the flow of a process. It is very useful, especially for complex program. Basically, there are 3 commands in conditional structure, they are:
   a. IF – THEN
   b. IF – THEN – ELSE
   c. CASE – OF

3.1. **IF – THEN**
   “IF command” used to select a single condition. If the condition true, then the statement in its block will be processed. If not, program will process nothing. The syntax of an “IF...THEN” statement is

   ```
   if expression then statement
   ```

   Where expression returns a Boolean value. If expression is true, then statement is executed; otherwise it is not. For example,

   ```
   if J <> 0 then Result := I/J;
   ```

   **Flowchart** of IF – THEN is:

   ![Flowchart of IF – THEN](image-url)
3.2. Perintah IF – THEN – ELSE

The syntax of an if...then...else statement is

```
If expression then statement1 else statement2
```

Where expression returns a Boolean value. If expression is true, then statement1 is executed; otherwise statement2 is executed. For example,

```plaintext
if J = 0 then Exit
else
  Result: = I/J;
```

The then and else clauses contain one statement each, but it can be a structured statement. For example,

```plaintext
if J <> 0 then
begin
  Result := I/J;
  Count := Count + 1;
end
else if Count = Last then
  Done := True
else
  Exit;
```

Notice that there is never a semicolon between the then clause and the word else. You can place a semicolon after an entire if statement to separate it from the next statement in its block, but the then and else clauses require nothing more than a space or carriage return between them. Placing a semicolon immediately before else (in an if statement) is a common programming error. Flowchart of IF – THEN – ELSE command is:
another syntax of IF – THEN – ELSE command is:

```
if (kondisi percabangan) then
begin
    ......; {aksi-1}
end
else
begin
    ......; {aksi-2}
end;
```

### 3.3. CASE – OF Command

CASE – OF command generally used to condition with many branches. The case statement may provide a readable alternative to deeply nested if conditionals. A case statement has the form

```
Case selectorExpression of
    caseList1: statement1;
    ...
    caseListn: statementn;
end
```

Where selector Expression is any expression of an ordinal type (string types are invalid) and each case List is one of the following:
A numeral, declared constant, or other expression that the compiler can evaluate without executing your program. It must be of an ordinal type compatible with selectorExpression. Thus 7, True, 4 + 5 * 3, 'A', and Integer('A') can all be used as caseLists, but variables and most function calls cannot. (A few built-in functions like Hi and Lo can occur in a caseList. See Constant expressions.)

A sub range having the form First. Last, where First and Last both satisfy the criterion above and First is less than or equal to Last.

A list having the form item1, ..., itemn, where each item satisfies one of the criteria above.

Each value represented by a caseList must be unique in the case statement; subranges and lists cannot overlap. A case statement can have a final else clause:

case selectorExpression of
    caseList1: statement1;
    ...
    caseListn: statementn;
    else
        statements;
end

where statements is a semicolon-delimited sequence of statements. When a case statement is executed, at most one of statement1 ... statementn is executed. Whichever caseList has a value equal to that of selectorExpression determines the statement to be used. If none of the caseLists has the same value as selectorExpression, then the statements in the else clause (if there is one) are executed.

The case statement

case I of
    1..5: Caption := 'Low';
    6..9: Caption := 'High';
    0, 10..99: Caption := 'Out of range';
    else
        Caption := "";
end;
is equivalent to the nested conditional

```plaintext
if I in [1..5] then
    Caption := 'Low'
else if I in [6..10] then
    Caption := 'High'
else if (I = 0) or (I in [10..99]) then
    Caption := 'Out of range'
else
    Caption := ''

flowchart of CASE - OF command is:
```

![Flowchart diagram]

Created by: Prohibited to reproduce in part or whole document without permission of the Faculty of Engineering, Yogyakarta State University
Reviewed by:
Bentuk sintaks dari perintah CASE – OF adalah sebagai berikut:

```
case (parameter) of
   nilai_1: .....; {aksi-1}
   nilai_2: .....; {aksi-2}
   nilai_3: .....; {aksi-3}
else
   .....; {aksi-4}
end;
```

4. Tools and Materials
- Personal Computer
- Windows XP Operating System
- Borland Delphi 7

5. Work Safety
   a. Before practicing, students should understand the goal and competency of this practice.
   b. Read and understand all procedures of this practice carefully.
   c. Always keep data from losing causes by extinguished electricity.

6. Procedures

6.1. IF – THEN Practice
1. Run Borland Delphi 7 on your computer.
2. In active form, select components and place on form like picture below:

   ![Image of components](image-url)

3. Set the properties of each components as follow:
4. The result is like picture below:

5. Double Click on Button1, then write code program as follow:

```pascal
procedure TForm1.Button1Click(Sender: TObject);
var
  nilai: integer;
begin
  nilai := StrToInt(Edit1.Text);
  if (nilai >= 70) then
  begin
    MessageDlg('Nilai Anda dinyatakan LULUS', mtInformation, [mbOk], 0);
  end;
end;
```

6. Run project by clicking toolbar or press F9 on your keyboard.

7. After program running, entry the number 75 into Edit1. Then press button Aksi. A popup message will show:
8. Entry value 50 into Edit1. Then press button Aksi. What would happen?

9. Try to input another value (0 - 100) into Edit1, then press button Aksi. What would happen?

10. Save your project as Percabangan1.

6.2. Praktik IF – THEN – ELSE

1. Create a new project in Delphi.

2. In active form, select components and place on form like picture below:
3. Set the properties of each components as follow:

<table>
<thead>
<tr>
<th>Component</th>
<th>Properties</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Form1</td>
<td>Caption</td>
<td>Struktur Percabangan IF-THEN-ELSE</td>
</tr>
<tr>
<td>Edit1</td>
<td>Text</td>
<td>(dikosongi)</td>
</tr>
<tr>
<td>Button1</td>
<td>Caption</td>
<td>Aksi</td>
</tr>
<tr>
<td>Label1</td>
<td>Caption</td>
<td>Hasil</td>
</tr>
<tr>
<td></td>
<td>Font - Size</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Font - Style - fsBold</td>
<td>True</td>
</tr>
</tbody>
</table>

4. The result is like picture below:

5. Double Click on Button1, then write code program as follow:

```pascal
procedure TForm1.Button1Click(Sender: TObject);
var
    nilai: integer;
begin
    nilai := StrToInt(Edit1.Text);
    if (nilai >= 70) then
        begin
            Label1.Caption := 'Hasil : Nilai Anda dinyatakan LULUS';
        end;
    else
        begin
            Label1.Caption := 'Hasil : Nilai Anda dinyatakan TIDAK LULUS';
        end;
end;
```

Created by : Prohibited to reproduce in part or whole document without permission of the Faculty of Engineering, Yogyakarta State University

Reviewed by :
6. Run project by clicking toolbar or press F9 on your keyboard.

7. Input value 75 into Edit1. Press button Aksi. The result will be shown like below:

![Image showing the result of inputting 75]

8. If value input is less than 70, then the result will be shown like below:

![Image showing the result of inputting 0]

9. Try to input another value (0 - 100) into Edit1, then press button Aksi. What would happen?

10. Save your project as Percabangan2.
6.3. Practicing CASE – OF

1. Create a new project in Delphi.

2. In active form, select components and place on form like picture below:

3. Set the properties of each components as follow:

<table>
<thead>
<tr>
<th>Component</th>
<th>Properties</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Form1</td>
<td>Caption</td>
<td>Struktur Percabangan CASE-OF</td>
</tr>
<tr>
<td>Edit1</td>
<td>Text</td>
<td>(dikosongi)</td>
</tr>
<tr>
<td>Button1</td>
<td>Caption</td>
<td>Aksi</td>
</tr>
<tr>
<td>Label1</td>
<td>Caption</td>
<td>Hasil Konversi ke Huruf =</td>
</tr>
<tr>
<td></td>
<td>Font - Size</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Font - Style - fsBold</td>
<td>True</td>
</tr>
</tbody>
</table>

4. The result is like picture below:
5. Double Click on Button1, then write code program as follow:

```pascal
procedure TForm1.Button1Click(Sender: TObject);
var
    nilai: integer;
begin
    nilai := StrToInt(Edit1.Text);
    case nilai of
        0..40: Label1.Caption := 'Hasil Konversi ke Huruf = D';
        41..65: Label1.Caption := 'Hasil Konversi ke Huruf = C';
        66..75: Label1.Caption := 'Hasil Konversi ke Huruf = B';
        76..100: Label1.Caption := 'Hasil Konversi ke Huruf = A';
        else
            Label1.Caption := 'Hasil Konversi ke Huruf = ';
    end;
end;
```

6. Run project by clicking toolbar or press F9 on your keyboard.

7. Input value 75 into Edit1. Press button Aksi. The result will be shown like below:
8. Try to input another value (0 - 100) into Edit1, then press button Aksi. What would happen?

9. Save your project as Percabangan3.

7. Discussion
a. Create an algorithm in a form of flowchart and the program to determine the biggest number between two integer!

b. Create an algorithm in a form of flowchart and the program to determine the smallest number between 3 integer input!

c. Create an algorithm in a form of flowchart and the program to determine whether the input is a positive integer or negative integer!