11.1 Learning summary report

Self-assessment

I have completed all the pass tasks. I am aiming for a pass in this unit.

Reflection

I have learned throughout this unit that computer programming is categorically different from the user interfaces that we see every day taking precedence over daily functional activities. It grants a greater appreciation for the work that undergirds much of modern software development and programming. It then becomes apparent that the conveniences of user interface design is then understood in its context, that the convenient ergonomics are non-existent without programmers and that computers are essentially unintelligent. The learning module has given me an experience to be able to understand that user interface has to be supported by good programming logic and error handling.
# Tasks for Isaac Low

<table>
<thead>
<tr>
<th>Task</th>
<th>Status</th>
<th>Included</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pass Task 1.1 - Hello World</td>
<td>Complete</td>
<td>YES</td>
</tr>
<tr>
<td>Pass Task 1.2 - Picture Drawing</td>
<td>Discuss</td>
<td>YES</td>
</tr>
<tr>
<td>Pass Task 1.3 - Creating a Procedure</td>
<td>Discuss</td>
<td>YES</td>
</tr>
<tr>
<td>Pass Task 2.1 - Hand Execute Assignment</td>
<td>Discuss</td>
<td>YES</td>
</tr>
<tr>
<td>Pass Task 2.2 - Hello User</td>
<td>Complete</td>
<td>YES</td>
</tr>
<tr>
<td>Pass Task 2.3 - My Drawing Procedure</td>
<td>Complete</td>
<td>YES</td>
</tr>
<tr>
<td>Pass Task 2.4 - My Functions</td>
<td>Discuss</td>
<td>YES</td>
</tr>
<tr>
<td>Pass Task 3.1 - Hand Execution of Control Flow</td>
<td>Ready to Mark</td>
<td>YES</td>
</tr>
<tr>
<td>Pass Task 3.2 - Name Tester</td>
<td>Complete</td>
<td>YES</td>
</tr>
<tr>
<td>Pass Task 3.3 - Circle Moving</td>
<td>Discuss</td>
<td>YES</td>
</tr>
<tr>
<td>Pass Task 4.1 - Using Records and Enumerations</td>
<td>Complete</td>
<td>YES</td>
</tr>
<tr>
<td>Test 1</td>
<td>Working</td>
<td>T1</td>
</tr>
<tr>
<td>Task</td>
<td>Status</td>
<td>Marked</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>-----------------</td>
<td>--------</td>
</tr>
<tr>
<td>Pass Task 5.1 - Hand Execution of Arrays</td>
<td>Ready to Mark</td>
<td>5.1P</td>
</tr>
<tr>
<td>Pass Task 5.2 - Arrays of Records</td>
<td>Discuss</td>
<td>5.2P</td>
</tr>
<tr>
<td>Pass Task 6.1 - Structure Charts</td>
<td>Ready to Mark</td>
<td>6.1P</td>
</tr>
<tr>
<td>Pass Task 7.1 - Programming Principles</td>
<td>Ready to Mark</td>
<td>7.1P</td>
</tr>
<tr>
<td>Pass Task 8.1 - Language Reference Sheet</td>
<td>Complete</td>
<td>8.1P</td>
</tr>
<tr>
<td>Pass Task 8.2 - Circle Moving 2</td>
<td>Ready to Mark</td>
<td>8.2P</td>
</tr>
<tr>
<td>Test 2</td>
<td>Fix and Resubmit</td>
<td>T2</td>
</tr>
<tr>
<td>Pass Task 9.1 - Reading Another Language</td>
<td>Ready to Mark</td>
<td>9.1P</td>
</tr>
</tbody>
</table>
Pass Task 1.1 - Hello World

Submitted By:
Isaac Kar Wai Low

Tutor:
Tanya Linden

August 23, 2015
program HelloWorld;

begin
  Writeln('Hello World !');
end.
Pass Task 1.2 - Picture Drawing

Submitted By: Isaac Kar Wai Low

Tutor: Tanya Linden

August 23, 2015
program PictureDrawing;
uses SwinGame, sgTypes;

begin
  OpenGraphicsWindow('House Drawing', 800, 600);
  ClearScreen(ColorGrey);

  FillEllipse(ColorGreen, 0, 400, 800, 400);
  FillRectangle(ColorTurquoise, 200, 400, 450, 200);
  FillRectangle(ColorYellow, 300, 450, 30, 30);
  FillRectangle(ColorMagenta, 400, 500, 50, 100);
  FillTriangle(ColorRed, 150, 400, 400, 150, 700, 400);
  FillCircle(ColorWhite, 0, 0, 200);

  RefreshScreen(60);
  Delay(50000);
end.
Pass Task 1.3 - Creating a Procedure

Submitted By: Isaac Kar Wai Low

Tutor: Tanya LINDEN

August 23, 2015
program BikeDrawing;
uses SwinGame, sgTypes;

procedure DrawRedBike();
begin
  DrawTriangle(ColorRed, 10, 20, 25, 5, 40, 20);
  DrawCircle(ColorRed, 10, 20, 10);
  DrawCircle(ColorRed, 40, 20, 10);
  DrawLine(ColorRed, 40, 20, 40, 0);
end;

procedure DrawBlueBike();
begin
  DrawTriangle(ColorBlue, 10, 20, 25, 5, 40, 20);
  DrawCircle(ColorBlue, 10, 20, 10);
  DrawCircle(ColorBlue, 40, 20, 10);
  DrawLine(ColorBlue, 40, 20, 40, 0);
end;

procedure DrawGreenMan();
begin
  DrawCircle(ColorGreen, 30, 20, 10);
  DrawLine(ColorGreen, 30, 30, 30, 70);
  DrawLine(ColorGreen, 30, 40, 60, 20);
  DrawLine(ColorGreen, 30, 40, 0, 20);
  DrawLine(ColorGreen, 30, 70, 40, 90);
  DrawLine(ColorGreen, 30, 70, 20, 90);
end;

procedure DrawLightGreenMan();
begin
  DrawCircle(ColorLightGreen, 30, 20, 10);
  DrawLine(ColorLightGreen, 30, 30, 30, 70);
  DrawLine(ColorLightGreen, 30, 40, 60, 50);
  DrawLine(ColorLightGreen, 30, 40, 0, 50);
  DrawLine(ColorLightGreen, 30, 70, 40, 80);
  DrawLine(ColorLightGreen, 30, 70, 20, 80);
end;

procedure Main();
begin
  OpenGraphicsWindow('BikeDrawing', 200, 200);
  LoadDefaultColors();

  ClearScreen(ColorWhite);
  DrawRedBike();
  RefreshScreen();
  Delay(5000);

  ClearScreen(ColorWhite);
  DrawBlueBike();
  RefreshScreen();
  Delay(5000);

  ClearScreen(ColorBlack);
  DrawGreenMan();
  RefreshScreen();
  Delay(5000);
ClearScreen(ColorBlack);
DrawLightGreenMan();
RefreshScreen();
Delay(5000);

end;

begin
    Main();
end.
Swinburne University of Technology

Introduction to Programming

Doubtfire Submission

Credit Task 1.4 - Concept Map

Submitted By: Isaac Kar Wai Low

Tutor: Tanya Linden

August 10, 2015
Programmers use Source Code to communicate with Computers. Source Code can be transformed using compilers. Transformed code can be directly executable (e.g., MINGW32).

1 unit = a sequence of instructions
Callable units = procedures (set of instructions)
Assembly language
Interpreted/compiled language (e.g., Pascal)
Swinburne University of Technology

Introduction to Programming

Doubtfire Submission

Pass Task 2.1 - Hand Execute Assignment

Submitted By: Isaac Kar Wai Low

Tutor: Tanya Linden

August 21, 2015
Hand Executables

Program 1

\[
\begin{align*}
    a &= 10 \\
    b &= 20 \\
    c &= 30 \\
\text{Width} &= c \text{ is } 30
\end{align*}
\]
Program 2

a = 2

WriteLn = 1 2
pizza = 5
pasta = 10
meals = 0
Written: Pizza: 5, Pasta: 10
Program 4

\[
\begin{align*}
a &= 10 - 20 \\
b &= 20 - 10 \\
c &= 30 - 20 \\
d &= 10 - 20 \\
e &= 50 - 49
\end{align*}
\]

Write: \(20, 10, 20, 20, 49\)
Pass Task 2.2 - Hello User

Submitted By: Isaac Kar Wai Low
Tutor: Tanya LINDE

August 23, 2015
program HelloUser;
uses TerminalUserInput;

procedure Main();
var
  name: String;
  age: Integer;
  year: Integer;
  yearBorn: Integer;
begin
  name := ReadString('Please enter your name: ');
  age := ReadInteger('How old are you this year? : ');
  year := ReadInteger('What year is it? : ');
  yearBorn := year - age;
  WriteLn('Hello ', name, ', you were born in ', yearBorn);
end;

begin
  Main();
end.
Hello Isaac Low, you were born in 1990

Hello Ivan, you were born in 1994

Hello Ivana, you were born in 1996
Pass Task 2.3 - My Drawing Procedure

Submitted By: Isaac Kar Wai Low
Tutor: Tanya Linden

October 23, 2015
program Diamonds;
uses SwinGame, sgTypes;

procedure FillDiamond(clr: Color; x, y: Single; width, height: Integer);
var
  midX, rightX, midY, bottomY: Single;
begin
  midX := x + width / 2;
  rightX := x + width;
  midY := y + height / 2;
  bottomY := y + height;
  //triangles
  FillTriangle(clr, x, midY, midX, y, rightX, midY);
  FillTriangle(clr, x, midY, midX, bottomY, rightX, midY);
end;

procedure DrawDiamond(clr: Color; x, y: Single; width, height: Integer);
var
  midX, rightX, midY, bottomY: Single;
begin
  midX := x + width / 2;
  rightX := x + width;
  midY := y + height / 2;
  bottomY := y + height;
  //lines
  DrawLine(clr, x, midY, midX, y);
  DrawLine(clr, x, midY, midX, y);
  DrawLine(clr, x, midY, midX, bottomY);
  DrawLine(clr, midX, bottomY, x, midY);
end;

procedure Main();
begin
  OpenGraphicsWindow('Diamonds', 400, 400);
  LoadDefaultColors();
  ClearScreen(ColorWhite);
  FillDiamond(ColorRed, 0, 0, 100, 100);
  FillDiamond(ColorGreen, 350, 350, 50, 50);
  FillDiamond(ColorBlue, 300, 0, 100, 50);
  FillDiamond(ColorGrey, 200, 200, 100, 100);
  FillDiamond(ColorYellow, 0, 300, 50, 100);
  DrawDiamond(ColorRed, 0, 0, 400, 400);
  DrawDiamond(ColorGreen, 30, 30, 250, 250);
  DrawDiamond(ColorMagenta, 100, 100, 0, 0);
  DrawDiamond(ColorBlue, 10, 200, 70, 150);
  DrawDiamond(ColorPink, 200, 10, 200, 70);
  RefreshScreen();
  Delay(5000);
end;

begin
  Main();
end.
end.
Pass Task 2.4 - My Functions

Submitted By: Isaac Kar Wai Low

Tutor: Tanya LINDEN

October 23, 2015
program InvestmentCalc;
uses TerminalUserInput, Math;

function CompoundInterest(principle, years: Integer; rate: Double): Double; //
  ← parameters or global variables
  // function allows for parameters to be passed
  // double stores.
  // double allows for decimals
  begin
    result := (principle * Power(1 + rate, years) - principle); // the function
    ← was drawn from the math
  end;

procedure Main(); // can pass parameters or global variables
  // procedure
  var
    principle, years: Integer; rate, BankA, BankB: Double;
    // local variables
  begin
    BankA := CompoundInterest(1000, 3, 0.035);
    BankB := CompoundInterest(1000, 3, 0.045);
    WriteLn ('Bank A $', BankA:4:2);
    WriteLn ('Bank B $', BankB:4:2);
    years := ReadInteger('Please enter the number of years for investment: ');
    principle := ReadInteger('Please enter a principle: $');
    rate := ReadDouble('Please enter a rate(%): ');
    WriteLn('You have accrued ', CompoundInterest(principle, years,
        ← rate/100):4:2, '$ for a ', years, ' -year investment.');
    // decimal places reinforced
  end;

begin
  Main();
end.
<table>
<thead>
<tr>
<th>Investment Details</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bank A $108.72</td>
<td>( \text{Accrued} = 160703819.66 ) for a 30-year investment</td>
</tr>
<tr>
<td>Bank B $141.17</td>
<td>( \text{Accrued} = 22118.78 ) for a 10-year investment</td>
</tr>
<tr>
<td>Bank A $108.72</td>
<td>( \text{Accrued} = 795.96 ) for a 6-year investment</td>
</tr>
</tbody>
</table>
**Concept Map 2.5**

**Concpts**
- Data type
  - e.g., Boolean, String (seq of chrs), Character, Integer, Double, etc.
- Value
  - e.g., irreducible, e.g., `Single`, `Double`
  - These are usually Variables

**Artefacts**
- Constant
  - unchanging data type value
- Variable
  - Values can vary/manipulated

**ACTION**
- Assign them!

- e.g.,
  - Procedure: `(x, y: Single, ...)`
  - Function: `(x, y: Single, ...)`

**Var**
- `midx := x - y` etc.

- Local variable

- In Pascal
  - Code uses built-in assembly modes

- Level of technology: type of variable used
  - Global/local
Pass Task 3.1 - Hand Execution of Control Flow

Submitted By: Isaac Kar Wai Low
Tutor: Tanya LINDEN

November 5, 2015
program GameMain;
uses SwinGame, sgTypes;

function buttonClicked(button_X, button_Y: Single; width, length: Integer): Boolean;
begin
  var mouse_X, mouse_Y, buttonWidth, buttonLength: Single;
  begin
    mouse_X := MouseX();
    mouse_Y := MouseY();
    buttonWidth := button_X + width;
    buttonLength := button_Y + length;
    result := false;
    if MouseClicked(LeftButton) and (mouse_X >= button_X) and (mouse_X <= buttonWidth) and (mouse_Y >= button_Y) and (mouse_Y <= buttonLength) then
      begin
        result := true;
      end;
  end;
end;

Procedure Main();
begin
  var clr: Color;
  OpenGraphicsWindow('Test Program for Button Click Code', 800, 600);
  ShowSwinGameSplashScreen();
  clr := ColorWhite;
  repeat
    ProcessEvents();
    ClearScreen(clr);
    DrawFrameRate(0,0);
    FillRectangle(ColorGrey, 50, 50, 100, 30);
    DrawText('Click Me', ColorBlack, 'arial.ttf', 14, 55, 55);
    If ButtonClicked(50, 50, 100, 30) then
      begin
        clr := RandomRGBColor(255);
      end;
  end;
  until WindowCloseRequested();
begin
  Main();
end.
Pass Task 3.2 - Name Tester

Submitted By: Isaac Kar Wai Low

Tutor: Tanya LINDEN

October 29, 2015
program NameTester;
uses TerminalUserInput;

procedure OutputSillyName(name: String);
var i: Integer;
begin
  i := 0;
  Write(name, ' is a '); // Write output w/o going to new line
  while i < 100 do
  begin
    WriteLn('silly ');
    i += 1;
  end; // loop statement repeats till i reaches 100
  WriteLn('name.');
end;

procedure Main();
var name: String;
begin
  name := ReadString('Please enter your name: '); // nested if-then in pascal
  if name = 'Isaac' then
     begin
      WriteLn('Your name is the best in the world!');
     end // nested statements do not require end colons
  else
     if name = 'Joe' then
        begin
          WriteLn('Looks like you are just another average ''Joe''!');
        end
     else
        if name = 'G.I. Joe' then
           begin
             WriteLn('Your must be very big.');
           end
        else
           begin
             OutputSillyName(name);
           end;
  end;
begin Main();
end.
<table>
<thead>
<tr>
<th>File 2 of 2</th>
<th>Screenshot</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>MINGW32\c\Users\Isaac\Desktop\CODE\Pass Tasks\3.2\3.2P-resources</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Isaaclow\ISAAC</td>
<td>c\Users\Isaac\Desktop\CODE\Pass Tasks\3.2\3.2P-resources</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>fpc -S2 Nametester.pas</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Free Pascal Compiler version 2.6.4 [2014/03/06] for i386</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Copyright (c) 1993-2014 by Florian Klaempfl and others</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Target OS: Win32 for i386</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Compiling Nametester.pas</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Linking Nametester.exe</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>47 lines compiled, 2.5 sec, 60816 bytes code, 12844 bytes data</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Isaaclow\ISAAC</td>
<td>c\Users\Isaac\Desktop\CODE\Pass Tasks\3.2\3.2P-resources</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Please enter your name: Isaac</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Your name is the best in the world!</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Isaaclow\ISAAC</td>
<td>c\Users\Isaac\Desktop\CODE\Pass Tasks\3.2\3.2P-resources</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Please enter your name: Joe</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Looks like you are just another average 'Joe'!</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Isaaclow\ISAAC</td>
<td>c\Users\Isaac\Desktop\CODE\Pass Tasks\3.2\3.2P-resources</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Please enter your name: G.I. Joe</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Your must be very big.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Isaaclow\ISAAC</td>
<td>c\Users\Isaac\Desktop\CODE\Pass Tasks\3.2\3.2P-resources</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>$</td>
<td></td>
</tr>
</tbody>
</table>
Pass Task 3.3 - Circle Moving

Submitted By: Isaac Kar Wai Low

Tutor: Tanya Linden

September 7, 2015
program CharacterMoving;
uses SwinGame, sgTypes;

procedure Main();
var
  x, y: Single;
const
  CIRCLE_RADIUS = 100;

begin
  OpenGraphicsWindow('CharacterMoving', 800, 600);
  Delay(5000);
  x := 400;
  y := 300;
  repeat
    // game loop
    ProcessEvents(); // event updated every time screen is refreshed
    if KeyDown(VK_LEFT) and (x - CIRCLE_RADIUS > 0) then begin
      x -= 1;
    end;
    if KeyDown(VK_RIGHT) and (x + CIRCLE_RADIUS < ScreenWidth()) then begin
      x += 1;
    end;
    if KeyDown(VK_UP) and (y - CIRCLE_RADIUS > 0) then begin
      y -= 1;
    end;
    if KeyDown(VK_DOWN) and (y + CIRCLE_RADIUS < ScreenHeight()) then begin
      y += 1;
      ClearScreen(ColorWhite);
      FillCircle(ColorGreen, x, y, 150);
      RefreshScreen(60); // 60 frames per second
    end;
  until WindowCloseRequested();

begin
  Main();
end.
Pass Task 4.1 - Using Records and Enumerations

Submitted By: Isaac Kar Wai Low

Tutor: Tanya LINDEN

September 7, 2015
program HighScoresLog;
uses TerminalUserInput;

type
  Difficulty = (Normal, Hard, Insane);
GameStoreData = record
  name: String;
  id: Integer;
  score: Integer;
  diff: Difficulty;
end;

function ReadDifficulty(prompt: String): Difficulty;
var
  option: Difficulty;
  val: Integer;
begin
  WriteLn('1 - Normal');
  WriteLn('2 - Hard');
  WriteLn('3 - Insane');
  val := ReadInteger(prompt);
  while ((val < 1) or (val > 3)) do
  begin
    WriteLn('Please enter a value between 1 and 3. ');
    val := ReadInteger(prompt);
  end;
  result := Difficulty(val - 1);
end;

function ReadGameScoreData(prompt: String): GameStoreData;
begin
  WriteLn(prompt);
  result.name := ReadString('Enter contact’s name: ');
  result.id := ReadInteger('Enter game id: ');
  result.score := ReadInteger('Please enter score ');
  result.diff := ReadDifficulty('Please enter difficulty -- ');
  while (result.score < 0) do
  begin
    result.score := ReadInteger('Invalid value. Please reenter score...');
  end;
end;

procedure PrintScoreData(const MyGameData: GameStoreData);
var
  name: String;
begin
  case MyGameData.diff of
    Normal: name := 'Normal';
Hard: name := 'Hard';
Insane: name := 'Insane';

WriteLn(MyGameData.name, ':', MyGameData.score, '(', MyGameData.id, ' -- ',
        ← name, ')');
if MyGameData.score < 1000 then
begin
    WriteLn('Newb')
end;
if MyGameData.score > 100000 then
begin
    WriteLn('Godlike')
end;

procedure Main();
var
    val: GameStoreData;
begin
    val := ReadGameScoreData('Enter Score Data -');
    PrintScoreData(val);
end;
begin
    Main();
end.
$ ./HighScoresLog
- Enter Score Data -
Enter contact's name: IL
Enter game id: 290871
Please enter score -1
1 - Normal
2 - Hard
3 - Insane
Please enter difficulty -- 5
Please enter a value between 1 and 3.
Please enter difficulty -- 2
Invalid value. Please reenter score... 10
IL:10(290871 -- Hard)

Newb

Isaaclow@ISAAC /c/Users/Isaaclow/Documents/CODE/4.1/4.1P-resources
$ ./HighScoresLog
- Enter Score Data -
Enter contact's name: IL
Enter game id: 290872
Please enter score 500000
1 - Normal
2 - Hard
3 - Insane
Please enter difficulty -- 3
IL:500000(290872 -- Insane)
Godlike

Isaaclow@ISAAC /c/Users/Isaaclow/Documents/CODE/4.1/4.1P-resources
$
Pass Task 5.1 - Hand Execution of Arrays

Submitted By: Isaac Kar Wai Low

Tutor: Tanya Linden

November 6, 2015
<table>
<thead>
<tr>
<th>data</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>[-2, 3, 8, 7]</td>
<td>0, 6, 4, 7, 16</td>
</tr>
<tr>
<td>[2, 6, 1, 3]</td>
<td>0, 2, 8, 7, 10</td>
</tr>
</tbody>
</table>
Function Name: CheckArrValue

<table>
<thead>
<tr>
<th>data</th>
<th>val</th>
<th>result</th>
</tr>
</thead>
<tbody>
<tr>
<td>2, 6, 3, 7</td>
<td>3</td>
<td>True</td>
</tr>
<tr>
<td>E, 1, 8, 2, 4, 9</td>
<td>6</td>
<td>False</td>
</tr>
</tbody>
</table>
Pass Task 5.2 - Arrays of Records

Submitted By: Isaac Kar Wai Low
Tutor: Tanya LINDEN

October 29, 2015
program HighScoresLog;
uses TerminalUserInput;

type
  Difficulty = (Normal, Hard, Insane); // coming out for the second test
GameStoreData = record
  name: String;
  id: Integer;
  score: Integer;
  diff: Difficulty;
end;
// dynamic array of player details stored in the GSD record
GameArray = array of GameStoreData;

function ReadDifficulty(prompt: String): Difficulty;
var
  val: Integer;
begin
  WriteLn('1 - Normal');
  WriteLn('2 - Hard');
  WriteLn('3 - Insane');
  val := ReadInteger(prompt);
  while ((val < 1) or (val > 3)) do
  begin
    WriteLn('Please enter a value between 1 and 3. ');
    val := ReadInteger(prompt);
  end;
  result := Difficulty(val - 1);
end;

function ReadGameScoreData(prompt: String): GameStoreData;
begin
  WriteLn(prompt);
  result.name := ReadString('Enter player''s name: ');
  result.id := ReadInteger('Enter game id: ');
  result.score := ReadInteger('Please enter score ');
  result.diff := ReadDifficulty('Please enter difficulty -- ');
  while (result.score < 0) do
  begin
    result.score := ReadInteger('Invalid value. Please reenter score...');
  end;
end;

procedure PrintScoreData(const MyGameData: GameStoreData);
var
  name: String;
begin
  case MyGameData.diff of
    Normal: name := 'Normal';
    // other cases
end;

Hard: name := 'Hard';
Insane: name := 'Insane';

WriteLn(MyGameData.name, ':', MyGameData.score, '(', MyGameData.id, ' -- ', name, ')');
if MyGameData.score < 1000 then
begin
WriteLn('Newb')
end;
if MyGameData.score > 100000 then
begin
WriteLn('Godlike')
end;
end;

procedure PopulatePlayers(var parray: GameArray);
var
i: Integer;
begin
for i := 0 to High(parray) do
begin
parray[i] := ReadGameScoreData('- Enter Player Score Data -');
end;
end;

procedure PrintPlayers(var players: GameArray);
var
i: Integer;
begin
for i := 0 to High(players) do
begin
WriteLn('Player Details:');
WriteLn('Name: ', players[i].name);
WriteLn('ID: ', players[i].id);
WriteLn('Score: ', players[i].score);
WriteLn('Difficulty: ', players[i].diff);
end;
end;

procedure AddPlayers(var players: GameArray);
begin
SetLength(players, Length(players) + 1);
players[High(players)] := ReadGameScoreData('Enter new player details: ');
end;

function AverageUserScore(const scores: GameArray; name: string): Double;
var
i: Integer;
numScores: Integer;
begin
result := 0;
numscores := 0;
end;
for i := 0 to High(scores) do
    begin
        if scores[i].name = name then
            begin
                result += scores[i].score;
                numScores += 1;
                if numScores > 0 then
                    begin
                        result /= numScores;
                    end;
                end;
            end;
end;

function HighestUserScore(const scores: Gamearray; name: string): Integer;
var
    i: Integer;
    Highest: Integer;
begin
    Highest := 0;
    for i := 0 to High(scores) do
        begin
            if scores[i].name = name then
                begin
                    if Highest < scores[i].score then
                        begin
                            Highest := scores[i].score;
                        end;
                end;
        end;
    result := Highest;
end;

procedure Main();
var
    Playerdatabase: GameArray;
    numPlayers: Integer;
    option: Integer;
begin
    numPlayers := ReadInteger('Enter initial number of contacts: ');
    SetLength(Playerdatabase, numPlayers);
    PopulatePlayers(Playerdatabase);
    while (numPlayers < 0) do
    begin
        numPlayers := ReadInteger('Please enter a number more than 0...');
    end;
    Repeat
        WriteLn('Menu');
        WriteLn('1: Add another score');
        WriteLn('2: Print all score');
        WriteLn('3: Print average user score for a user');
        WriteLn('4: Print highest score for a user');
        WriteLn('5: Quit');
        option := ReadInteger('option -- > ');
    end;
while ((option < 1) or (option > 5)) do
begin
option := ReadInteger('Please enter a value between 1 and 5 to select option from menu...');
end;
case option of
1: AddPlayers(Playerdatabase);
2: PrintPlayers(Playerdatabase);
3: WriteLn('Average User Score: ', AverageUserScore(Playerdatabase, ReadString('Which player name: '))):4:2);
4: Writeln('Highest User Score: ', HighestUserScore(Playerdatabase, ReadString('Which player name: ')));
end;
until option = 5;
end;

// the loop function goes back to menu until option is 5
begin
Main();
end.
Enter player's name: Derp
Enter game ID: 304
Please enter score 1000
Option 1: Normal
Option 2: Insane
Please enter difficulty -- 2
Enter player score data:
Enter player's name: Derp
Enter game ID: 304
Please enter score 1000
Option 1: Normal
Option 2: Insane
Please enter difficulty -- 1
Menu:
1: Add another score
2: Print all score
3: Print average user score for a user
4: Print highest score for a user
5: Quit
Option 3: Average user score: Which player name: Derp
Total: 1000
Menu:
1: Add another score
2: Print all score
3: Print average user score for a user
4: Print highest score for a user
5: Quit
Ready
Pass Task 6.1 - Structure Charts

Submitted By: Isaac Kar Wai Low
Tutor: Tanya Linden

November 15, 2015
6.1 Structure Chart

Main()

PrintPlayers()

SetLength()

ReadGameStoreData

AddPlayers()

PopulatePlayers()

Average(UserScore())

Highest(UserScore())

String(User input)

String(User)

Playerdatabase

Playerdatabase

Playerdatabase

numPlayers (from user)

GameAway

Playerdatabase

File 1 of 1 Program structure chart
Pass Task 7.1 - Programming Principles

Submitted By: Isaac Kar Wai LOW
Tutor: Tanya LINDEM

November 15, 2015
7.1 Programming principles

To issue commands to a program we need to define the language that we are using. Most programming languages may use the same grammar but not the same syntax.

For example:

Programming in C++:

```c++
int main()
{
    const int CIRCLE_RADIUS = 100;
    int x, y;
}
```

Programming in Pascal:

```pascal
procedure Main();
const
CIRCLE_RADIUS = 100;
var
x, y: Single;
begin
    // insert procedure
end.
```

These languages usually require compiling in order to become executable. The main language that is going to be explained in this document is Pascal code.

Variables and constants

In programming, variables are values that are defined sequentially within a procedure or function, and they are subject to change depending on what the programmer assigns to it. A constant is basically a value that does not change. Variables and constants are usually part of instructions. The interpreter goes through an instruction which is ended with a semicolon. As for the C++ example excerpt above, the interpreter runs through the script as such:

- Define a constant with integer type called CIRCLE_RADIUS and assign it the value of 100. Instruction ends.
- Define variables with integer type x and y. Instruction ends.

Variables in Pascal are usually declared before the “begin” phase of the function or procedure being declared.

Variable types

a) Integers and floats

Integers simply represents natural numbers, i.e. numbers with no decimals. Should you want to use a number with decimals, use floats.

```pascal
var
x, y: Integer;
```
b) Strings  
Strings usually represent a lines of text or a long sentence of multiple characters. Text are usually enclosed in double quotes or single quotes when assigned.
e.g.

```pascal
var
  x, y: String;
```

c) Booleans  
Booleans are usually outcome based and may represent the end result of a function, that is a value of true/false, yes/no.

d) Arrays  
Usually declared thus:

```pascal
var
  x: an array of [0..3] Integer; // can also be other types like String
```

Arrs store multiple variables of the same type.

### Declaring functions and procedures

Declaring functions and procedures in Pascal is fairly explicit as their labels suggest:

```pascal
function CompoundInterest(principle, years: Integer; rate: Double): Double;
```

The components in the bracket are called parameters, or what is known as global variables. These are variables declared that would have an effect within the whole function. The type declaration at the end of the function is the return value of the function.

In Pascal, the function must be assigned a return value. E.g.

```pascal
begin
  result := (principle * Power(1 + rate, years) - principle);
end;
```

The declaration of a procedure remains with the same principles only it does not need to return a value.

```pascal
procedure main ();
```

### Conditions

(script excerpt below is referenced as Figure 1)

```pascal
function HighestUserScore(const scores: Gamearray; name: string): Integer;
var
  i: Integer;
  Highest: Integer;
begin
  Highest := 0;
  for i := 0 to High(scores) do
    begin
      if scores[i].name = name then
        begin
          if Highest < scores[i].score then
            begin
```
The instruction if is usually preceded by then, and an opening statement. This is a conditional statement that constrains an outcome to certain conditions.

As evident above, there is also the allocation of a nested if statement, which is basically a conditional statement within another conditional statement. A nested if is usually used when there are conditional statements that are mutually dependent.

**Loops**

a) **For Loops**

For loops in Pascal usually demand the assignments of multiple arrays to conditional statements. Take for example:

```pascal
procedure AddPlayers(var players: array of Integer);
begin
  SetLength(players, Length(players) + 1);
  players[High(players)] := ReadInteger('Enter new player number: ');
end;
```

Here is a procedure that is declared named PopulatePlayers with a parameter representing an indefinite array of integers. Usually, programmers would use i as the variable type integer. When beginning the procedure,

- a variable is declared within the procedure of an array type of indefinite integers
- SetLength helps add 1 player integer to the existing length of players
- High(Player) references an integer with the highest number, which is the newly added number in this case (Length(players) + 1)

If a programmer wants to input player numbers for an indefinite amount of arrays, a different type of loop will become useful namely,

b) **Repeat loops, while loops and case statements**

‘Repeat’ loops carry out similar functions, only that the loop terminates upon a conditional statement indicated by an ‘until’ statement. A while loop is like an ‘if’ statement that repeats itself. A case gives a variable an array of callable statements, usually integers, which programmers can use to create a menu of options. The case option 1 calls for the AddPlayers function which runs through the array

For example:
procedure Main(); // this is only an excerpt of the procedure called Main
var
  option: Integer;
begin
  Repeat //repeat loops
    WriteLn('Menu');
    WriteLn("1: Add another player number");
    WriteLn("2: End");
    option := ReadInteger('option -- > '); // while loop
    while ((option < 1) or (option > 2)) do
      begin
        option := ReadInteger('Please enter a value between 1 and 2 to select option from menu...');
      end;
    case option of
      1: AddPlayers();
    end; // note that ending statement does not have ‘begin’ in this case
  until option = 2; // loop ends here when ‘2’ is selected but allows addition of multiple players within the repeat loop
end;
Swinburne University of Technology

Introduction to Programming

Doubtfire Submission

Pass Task 8.1 - Language Reference Sheet

Submitted By: Isaac Kar Wai LOW

Tutor: Tanya LINDEN

October 30, 2015

Produced by Doubtfire³
## C reference sheet

<table>
<thead>
<tr>
<th>Program structure/ function</th>
<th>C preprocessor</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>type fnc(type1,..)</code></td>
<td>function declarations</td>
</tr>
<tr>
<td><code>type name</code></td>
<td>external variable declarations</td>
</tr>
<tr>
<td><code>main() { ... }</code></td>
<td>main routine</td>
</tr>
<tr>
<td><code>type fnc(arg1, ..) {   }</code></td>
<td>function definition</td>
</tr>
<tr>
<td><code>/* */</code></td>
<td>comments</td>
</tr>
<tr>
<td><code>main(int argc, char *argv[])</code></td>
<td>main with args</td>
</tr>
<tr>
<td><code>exit (arg)</code></td>
<td>terminate execution</td>
</tr>
</tbody>
</table>

### Data types/declarations

<table>
<thead>
<tr>
<th>Character</th>
<th>Integer</th>
<th>Real number</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>char</code></td>
<td><code>int</code></td>
<td><code>float, double</code></td>
</tr>
</tbody>
</table>

### Arrays and Pointers

<table>
<thead>
<tr>
<th>Character</th>
<th>Integer</th>
<th>Real number</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>type pointer</code></td>
<td><code>type</code></td>
<td><code>type (*pf)();</code></td>
</tr>
</tbody>
</table>

### Operators

#### Increment/ decrement
`++`, `--`  
Go to `goto label;`

#### Relational comparisons
`>`, `>=`, `<`, `<=`  
Return value `return value`  
Return `return expr`  
Flow Constructions

#### Logical and
`&&`  
If `if statement`  
If `(expr1) statement1`
| Logical or       | while statement       | else if (expr2) statement2
|-----------------|-----------------------|---------------------------
| logical or      | while statement       | else statement3           |
| assignment operators | for statement  | while (expr) statement   |
| expression evaluation separator | do statement  | for (expr1; expr2; expr3) statement |
| separator       | do statement          | do statement              |
| separator       | switch statement      | switch (expr) {           |
| expression evaluation separator | separator  | case const1: statement1, |
| separator       | do statement          |   break;                  |
| expression evaluation separator | separator  | case const2: statement2, |
| separator       | do statement          |   break;                  |
| expression evaluation separator | separator  | default: statement       |

**ANSI Standard Libraries**

<table>
<thead>
<tr>
<th>Character Class Tests &lt;ctype.h&gt;</th>
<th>String Operations &lt;string.h&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>alphanumeric?</td>
<td>isalnum(c)</td>
</tr>
<tr>
<td>alphabetic?</td>
<td>isalpha(c)</td>
</tr>
<tr>
<td>control character?</td>
<td>iscntrl(c)</td>
</tr>
<tr>
<td>decimal digit?</td>
<td>isdigit(c)</td>
</tr>
<tr>
<td>lower case letter?</td>
<td>islower(c)</td>
</tr>
<tr>
<td>upper case letter?</td>
<td>isupper(c)</td>
</tr>
<tr>
<td>convert to lower case?</td>
<td>tolower(c)</td>
</tr>
<tr>
<td>convert to upper case?</td>
<td>toupper(c)</td>
</tr>
<tr>
<td>hexadecimal digit?</td>
<td>isxdigit(c)</td>
</tr>
<tr>
<td>length of s</td>
<td>strlen(s)</td>
</tr>
<tr>
<td>copy ct to s</td>
<td>strcpy(s, ct)</td>
</tr>
<tr>
<td>concatenate ct to s</td>
<td>strcat(s, ct)</td>
</tr>
<tr>
<td>compare cs to ct</td>
<td>strcmp(cs, ct)</td>
</tr>
<tr>
<td>copy cs to ct</td>
<td>strncmp(cs, ct)</td>
</tr>
</tbody>
</table>
Pass Task 8.2 - Circle Moving 2

Submitted By: Isaac Kar Wai Low

Tutor: Tanya Linden

November 6, 2015
#include <stdio.h>
#include "SwinGame.h"

int main() /* declares a procedure */
{
    const int CIRCLE_RADIUS = 100;
    int x, y;
    open_graphics_window("Circle Moving2 - C/C++", 800, 600);
    x = 400;
    y = 300;
    do
    {
        // do-while syntax loop is do {...} while(); return ...;
        // a replacement for pascal’s repeat-until syntax loop
        process_events();
        if (key_down(VK_LEFT) && (x - CIRCLE_RADIUS)>0)
        {
            x -= 1;
        }
        if (key_down(VK_RIGHT) && (x + CIRCLE_RADIUS)<screen_width())
        {
            x += 1;
        }
        if (key_down(VK_UP) && (y - CIRCLE_RADIUS)>0)
        {
            y -= 1;
        }
        if (key_down(VK_DOWN) && (y + CIRCLE_RADIUS) < screen_height())
        {
            y += 1;
        }
        clear_screen(COLOR_WHITE);
        fill_circle(COLOR_GREEN, x, y, CIRCLE_RADIUS);
        refresh_screen(60); // 60 frames per second
    } while (!window_close_requested());
    delay(5000);
    return 0;
}
Pass Task 9.1 - Reading Another Language

Submitted By: Isaac Kar Wai Low

Tutor: Tanya Linden

November 15, 2015
Function Name: int neg_num

<table>
<thead>
<tr>
<th>data</th>
<th>S2</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>[4, -1, -3, 2, 7]</td>
<td>5</td>
<td>0, 1, 2</td>
</tr>
<tr>
<td>[2, 6, -1, 3]</td>
<td>4</td>
<td>0, 1</td>
</tr>
</tbody>
</table>

Section 3.
3.1. Update
Write an SQL Update statement that sets the max_pallets of warehouse 54 to 20,
Function Name: bool check_value

<table>
<thead>
<tr>
<th>data</th>
<th>≤</th>
<th>Val</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>[2, 6, -3, 3, 7]</td>
<td>≤</td>
<td>-3</td>
<td>false, true, false</td>
</tr>
<tr>
<td>[-1, 8, 2, -4, 9]</td>
<td>≤</td>
<td>6</td>
<td>false</td>
</tr>
</tbody>
</table>
Pass Task 11.1 - Learning Summary Report

Submitted By: Isaac Kar Wai Low
Tutor: Tanya Linden

November 15, 2015
11.1 Learning summary report

Self-assessment

I have completed all the pass tasks. I am aiming for a pass in this unit.

Reflection

I have learned throughout this unit that computer programming is categorically different from the user interfaces that we see every day taking precedence over daily functional activities. It grants a greater appreciation for the work that undergirds much of modern software development and programming. It then becomes apparent that the conveniences of user interface design is then understood in its context, that the convenient ergonomics are non-existent without programmers and that computers are essentially unintelligent. The learning module has given me an experience to be able to understand that user interface has to be supported by good programming logic and error handling.