|  |  |  |
| --- | --- | --- |
| **OFPPT** | **مــــــكــــــتــــــب الــــــتــــــكــــــويــــــن الــــــمــــــهــــــنــــــي و إنــــــعــــــاش الــــــشــــــغــــــل** | |
| **Office de la Formation Professionnelle et de la Promotion du Travail** | |
| **Direction Recherche et Ingénierie de la Formation** | | |
| **Examen de passage à la 2ème année**  **Session Juillet 2018** | | |
| **Filière  :** **Techniques de Développement Informatique** | | **Epreuve  : Synthèse** |
| **Niveau  : TS** | | **Variante  : V2** |
| **Durée  : 5 heures** | | **Barème  : / 120 pts** |

|  |
| --- |
| **Eléments de réponse** |

|  |  |
| --- | --- |
| **Partie I : Théorie** | **40 pts** |
|  | |
| **Dossier 1 : L’essentiel en technologies de l’information** | **12 pts** |
|  | |
| 1. **0.5 pt pour chaque conversion** | **6 pts** |
| |  |  |  |  | | --- | --- | --- | --- | | **Binaire** | **Octal** | **Décimal** | **Hexadécimal** | | 111100 | **74** | **60** | **3C** | | **1011001** | 131 | **89** | **59** | | **10000011** | **203** | 131 | **83** | | **100110001** | **461** | **305** | 131 | | |
|  | |
| 1. La fonction f : | **1.5 pts** |
| Simplification : | **1.5 pts** |
| 1. Simplification moyennant le tableau de Karnaugh :  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | |  |  |  |  |  |  | |  | 0 | **1** | 0 | **1** | |  | 0 | **1** | 0 | **1** | | **3 pts** |

|  |  |
| --- | --- |
| **Dossier 2 : Analyse et conception orientée objet** | **16 pts** |
|  | |
| 1. Diagramme de cas d’utilisation : **0.25 pt pour chaque élément (acteur, Cas d’utilisation et association)** | **7.5 pts** |
|  | |
| 1. Diagramme de classes : **0.5 pt pour chaque élément (classe et association)** | **8.5 pts** |
|  | |
| **Dossier 3 : Programmation structurée** | **12 pts** |
|  | |
| Algorithme Gestion des Circonscriptions |  |
| Structure Circonscription  Nom : Chaine  Population : Entier  Region : Chaine  FinStructure | **1 pt** |
| Constante nMax 🡨 20 |  |
| Variables circonscriptions[nMax] : Circonscription | **1 pt** |
| n, i  populationTotal,  populationMin,  indiceCirconscriptionPeuplee : Entier  nomRegion : Chaine  Début |  |
| Répéter  Ecrire("Nombre de circonscriptions ? ")  Lire(n)  TantQue(NON(n > 0 ET n <= nMax)) | **1 pt** |
| Pour i 🡨 0 jusqu’à n - 1  Ecrire("Circonscription ", i + 1, " : \n")  Ecrire("Nom : ")  Lire(circonscriptions[i].Nom)  Ecrire("Population : ")  Lire(circonscriptions[i].Population)  Ecrire("Région : ")  Lire(circonscriptions[i].Region)  FinPour | **2 pts** |
| Ecrire("Région dont vous voulez calculer la population totale : ")  Lire(nomPays)  populationTotale 🡨 0  Pour i 🡨 0 jusqu’à n - 1  Si(nomRegion = circonscriptions[i].Region)  populationTotale += circonscriptions[i].Population  FinSi  FinPour  Ecrire("La population totale est : ", populationTotale) | **3 pts** |
| populationMin 🡨 0  Pour i 🡨 0 jusqu’à n - 1  Si(populationMin > circonscriptions[i].Population)  populationMin 🡨 circonscriptions[i].Population  indiceCirconscriptionPeuplee 🡨 i  FinSi  FinPour  Ecrire("Circonscription la moins peuplée : ", circonscriptions[indiceCirconscriptionPeuplee].Nom)  Fin | **4 pts** |

|  |  |
| --- | --- |
| **Partie II : Pratique** | **80 pts** |
|  | |
| **Dossier 1 : Programmation structurée** | **16 pts** |
|  | |
| 1. f(387) 🡪 1 | **2 pts** |
| 1. f(38) 🡪 2 | **2 pts** |
| 1. f(3) 🡪 3 | **2 pts** |
| |  | | --- | |  | | |
| |  | | --- | |  | | |
| **Dossier 2 : Programmation événementielle et orientée objet** | **64 pts** |
|  | |
| class Région  {  //1. 1.5 pts  private string \_Nom;  public string Nom  {  get { return \_Nom; }  set  {  //1.a 3 pts  if (value.Length < 4)  throw new Exception("Le nom doit contenir au moins 4 caractères");  foreach (char caractère in value)  {  if (char.IsDigit(caractère))  throw new Exception("Le nom ne peut pas contenir des chiffres");  }  \_Nom = value;  }  }  //1.(suite) 1.5 pts  private List<Circonscription> \_Circonscriptions = new List<Circonscription>();  public List<Circonscription> Circonscriptions  {  get { return \_Circonscriptions; }  set { \_Circonscriptions = value; }  }  //1.(suite) 1.5 pts  private Circonscription \_CirconscriptionPrincipale;  public Circonscription CirconscriptionPrincipale  {  get { return \_CirconscriptionPrincipale; }  set  {  //1.b 3 pts  Ajouter(value);  //La méthode vérifie l'existence de la circonscription  \_CirconscriptionPrincipale = value;  }  }  //2.a 1.5 pts  public Région(string nom)  {  Nom = nom;  }  //2.b 2 pts  public Région(string nom, Circonscription circonscriptionPrincipale)  {  Nom = nom;  CirconscriptionPrincipale = circonscriptionPrincipale;  }  //3 2 pts  public int NombreCirconscriptions  {  get  {  return Circonscriptions.Count;  }  }  //4 3 pts  public int Rechercher(String nomCirconscription)  {  foreach (Circonscription circonscription in Circonscriptions)  if (circonscription.Nom == nomCirconscription)  return Circonscriptions.IndexOf(circonscription);  return -1;  }  //5 3 pts  public void Ajouter(Circonscription circonscription)  {  if (Rechercher(circonscription.Nom) == -1)  Circonscriptions.Add(circonscription);  }  //6 4 pts  public void Supprimer(String nomCirconscription)  {  int indiceCirconscription = Rechercher(nomCirconscription);  if (indiceCirconscription != -1)  {  Circonscriptions.RemoveAt(indiceCirconscription);  if (CirconscriptionPrincipale.Nom == nomCirconscription)  CirconscriptionPrincipale = null;  }  }  //7 5 pts  public List<Circonscription> TrierParPopulation(string ordre)  {  List<Circonscription> circonscriptions = new List<Circonscription>();  circonscriptions.AddRange(Circonscriptions);  if (ordre.ToLower() == "croissant")  circonscriptions.Sort(new PopulationCroissantComparer());  else if (ordre.ToLower() == "décroissant")  circonscriptions.Sort(new PopulationDécroissantComparer());  return circonscriptions;  }  //8 3 pts  public int PopulationTotale()  {  int populationTotale = 0;  foreach (Circonscription circonscription in Circonscriptions)  populationTotale += circonscription.Population;  return populationTotale;  }  }  //7(suite) 5 pts  class PopulationCroissantComparer : IComparer<Circonscription>  {  public int Compare(Circonscription x, Circonscription y)  {  return x.Population.CompareTo(y.Population);  }  }  //7(suite) 5 pts  class PopulationDécroissantComparer : IComparer<Circonscription>  {  public int Compare(Circonscription x, Circonscription y)  {  return -x.Population.CompareTo(y.Population);  }  } | |
|  | |
|  | |
| public partial class Gestion\_des\_Régions : Form  {  //9.a 1 pt  List<Région> LesRégions = new List<Région>();  public Gestion\_des\_Régions()  {  InitializeComponent();  }  //9.b 1 pt  private void Gestion\_des\_Régions\_Load(object sender, EventArgs e)  {  //9.b.i 6 pts  Région région1 = new Région("Casablanca-Settat");  région1.Ajouter(new Circonscription("Préfecture de Mohammédia", 322286));  région1.Ajouter(new Circonscription("Préfecture de Casablanca", 2949805));  région1.Ajouter(new Circonscription("Province de Nouaceur", 236119));  région1.Ajouter(new Circonscription("Province de Médiouna", 122851));  région1.Ajouter(new Circonscription("Marrakech", 928850));  région1.CirconscriptionPrincipale = région1.Circonscriptions[1];  LesRégions.Add(région1);  Région région2 = new Région("Marrakech-Safi");  LesRégions.Add(région2);  Région région3 = new Région("Laâyoune-Sakia El Hamra");  région3.CirconscriptionPrincipale =  new Circonscription("Province de Laâyoune", 210023);  région3.Ajouter(new Circonscription("Province de Boujdour", 46129));  région3.Ajouter(new Circonscription("Province de Tarfaya", 10410));  LesRégions.Add(région3);  //9.b.ii 3 pts  comboBox\_Régions.DataSource = LesRégions;  comboBox\_Régions.DisplayMember = "Nom";  }  //9.c 4 pts  private void comboBox\_Région\_SelectedIndexChanged(object sender, EventArgs e)  {  Région Région = comboBox\_Régions.SelectedItem as Région;  listBox\_Circonscriptions.Items.Clear();  if (Région.NombreCirconscriptions != 0)  {  foreach (Circonscription Circonscription in Région.Circonscriptions)  {  //9.c.i 2 pts  if (Région.CirconscriptionPrincipale != null &&  Circonscription.Nom == Région.CirconscriptionPrincipale.Nom)  listBox\_Circonscriptions.Items.Add("--> " + Circonscription);  else  listBox\_Circonscriptions.Items.Add(Circonscription);  }  }  //9.c.ii 2 pts  else  listBox\_Circonscriptions.Items.Add("Aucune circonscription !!!");  }  //9.d 4 pts  private void button\_Trier\_Click(object sender, EventArgs e)  {  Région Région = comboBox\_Régions.SelectedItem as Région;  if (Région.NombreCirconscriptions != 0)  {  string ordre = "croissant";  List<Circonscription> Circonscriptions;  if (radioButton\_Décroissant.Checked) ordre = "décroissant";  Circonscriptions = Région.TrierParPopulation(ordre);  listBox\_Circonscriptions.Items.Clear();  foreach (Circonscription Circonscription in Circonscriptions)  {  if (Région.CirconscriptionPrincipale != null &&  Circonscription.Nom == Région.CirconscriptionPrincipale.Nom)  listBox\_Circonscriptions.Items.Add("--> " + Circonscription);  else  listBox\_Circonscriptions.Items.Add(Circonscription);  }  }  }  //9.e 4 pts  private void button\_OrdreInitial\_Click(object sender, EventArgs e)  {  comboBox\_Région\_SelectedIndexChanged(comboBox\_Régions, new EventArgs());  }  //9.f 3 pts  void Gestion\_des\_Régions\_FormClosing(object sender, FormClosingEventArgs e)  {  if (MessageBox.Show("Voulez-vous vraiment fermer l'application ?",  "Attention",  MessageBoxButtons.YesNo,  MessageBoxIcon.Question) == DialogResult.No)  e.Cancel = true;  }  } | |