Université Mohammed V de Rabat

Ecole Nationale Supérieure d'Informatique et d'Analyse des Systèmes

NOM et PRENOM : .....
CIN : .....
CODE QR: .....
.....
.....
.....

Université Mohammed V de Rabat

Réservé à l'administration

Observations des Correcteurs :	**Note :**	**Code QR:**

Concours d'accès à l'ENSIAS pour les candidats titulaires du DEUG
Epreuve : Informatique
Mercredi 12 Juillet 2023
Horaire : de 11h45 à 12h45

La durée de l'épreuve est de 1 Heure.
Elle comporte III exercices sur 6 pages. Une réponse erronée sera comptabilisée par un 0.
Aucun document n'est autorisé. Le barême est donné seulement à titre indicatif.

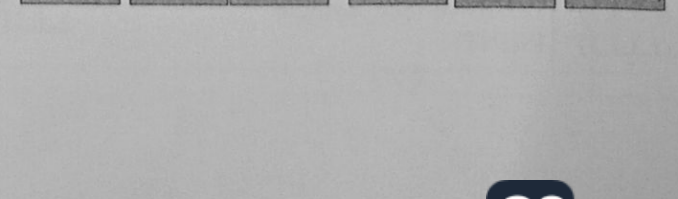
Exercice I — Structure de données, Machine, Code et Calcul (10pts)

Une Pile est une mémoire théoriquement infinie à accès séquentiel organisée selon l'ordre d'arrivée des données stockées (*last-in/first-out* ou *LIFO où la dernière donnée entrée (empilée « Pushed »), est la première donnée sortie (dépilée « Poped ») - comme une pile d'assiettes*). Les machines disposant de mémoire sous forme de pile s'appellent des machines à pile (*Stack Machine*). Les machines à pile ont servi à construire des machines pour des langages tels Smalltalk-80 et Java (JVM - Java Virtual Machine). Les codes à 0-address (stack code) sont des codes très simples pour ces machines à pile. Les opérations d'un code à 0-address supposent un opérande implicite (Une pile représentant la mémoire Stack de la machine abstraite). Un programme se décompose d'une partie définition de variables (ex. **X** 1.000000 **Y** 0.000000) se terminant par le label **begin:** et une partie code commençant après le label **begin:** (début du code) et se terminant par le label **end:** (fin du code). Le Jeu d'instruction d'un code à 0-address se compse de plusieurs instructions : LOAD, PUSH, STORE, POP, ADD, SUB, MULT, SWAP, DUPL, etc. **LOAD** : charge la valeur d'une variable depuis la mémoire statique, puis l'empile vers la mémoire Pile [STATIC→STACK]. **PUSH** : empile une valeur constante vers la mémoire Pile. **STORE** : dépile une valeur de la tête de la mémoire Pile et la stocke dans la variable en paramètre [STACK→STATIC]. Chaque opération binaire **OP** parmi (ADD, SUB, MULT, IDIV (division euclidienne), DDIV (division décimale), etc.) : dépile les deux éléments de Pile et empile leur résultat dans la Pile (Push(Pop() <op> Pop()). Par exemple, **PUSH** 1.0 **PUSH** 3.0 **ADD** donnera une pile avec un seul élement en tête de pile qui est 4.0, **PUSH** 1.0 **PUSH** 3.0 **SUB** donnera une pile avec un seul élement en tête de pile qui est 2.0. **SWAP** inter-change les deux éléments de tête de Pile. SWAP est utile pour les opérations non commutatives. Le label **unlabel :** en début d'instruction est l'adresse d'une instruction (ex. **for:**, **endfor:**). **JMP** <label> : effectue un branchement (Go To) vers le label désigné (ex. JMP endfor : effectue un branchemen vers l'instruction dans l'étiquette est endfor:). **JEQ** <label> : (signifie Jump if Equal) dépile la tête de Pile et la met dans une mémoire temporaire op1, dépile la nouvelle tête de pile et la met dans une mémoire temporaire op2, si op1=op1, l'instruction effectue un branchement (Go To) vers le label désigné. **JNE** <label> : (signifie Jump if Not Equal) dépile la tête de pile et la met dans une mémoire temporaire op1, dépile la nouvelle tête de pile et la met dans une mémoire temporaire op2, si op1 ≠op1, l'instruction effectue un branchement (Go To) vers le label désigné. **JG** <label> : (signifie Jump if Greater) dépile la tête de Pile et la met dans une mémoire temporaire op1, dépile la nouvelle tête de Pile et la met dans une mémoire temporaire op2, si op1>op1, l'instruction effectue un branchement (goto) vers le label désigné. **DUPL** duplique l'élément en tête de Pile. DUPL est utile pour l'exposant (**2) et pour la multiplication (*2). **PRINTI** : dépile la tête de pile (entière/réelle) et l'affiche à l'écran. **PRINTS** <chaîne de caractères>: et affiche la chaîne de caratères à l'écran.

Exemple de Programme

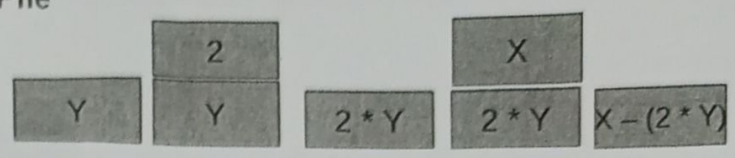
```
LOAD X      // 1. tête de pile = valeur de X
PUSH 2      // 2. tête de pile = 2
LOAD Y      // 3. tête de pile = valeur de Y
MULT        // 4. tête de pile = valeur de 2 * Y
SWAP        // 5. têtes de pile 2*Y & X interverties
SUB         // 6. tête de pile = valeur de X - (2 * Y)
```

Etat de la mémoire Pile de la machine abstraite



```
LOAD Y      // 1. tête de pile = valeur de Y
PUSH 2      // 2. tête de pile = 2
MULT        // 3. tête de pile = valeur de 2 * Y
LOAD X      // 4. tête de pile = valeur de X
SUB         // 5. tête de pile = valeur de X - (2 * Y)
```

Pile



I.1) Quel est l'état de la pile après les instructions 0-address code suivantes (2pts) : **Compléter les pointillés**

Partie 1	Partie 2
X 2.000000 Y 100.000000 Z 0.000000 W 0.000000 begin: PUSH 150.000000 LOAD X SWAP	DDIV DUPL MULT LOAD Y SWAP DUPL end:

Etat de la Pile
.....
.....
.....
.....
.....

I.2) Que calculent et affichent les programmes suivants (*NB. Les résultats sont de très grands nombres que vous ne calculerez pas à la main, seul le* ***nom de la fonction calculée*** *et (ii) le* ***paramètre initial*** *de la fonction importent) :*

Partie 1	Partie 2	Partie 3	Compléter les pointillés
(I.2.1) (2pts) A1 1.000000 B1 0.000000 **begin:** PUSH 1.000000 STORE B1 ENSIAS: PUSH 20.000000	LOAD B1 JG SORTIE LOAD B1 LOAD A1 MULT STORE A1 PUSH 1.000000 LOAD B1	ADD STORE B1 JMP ENSIAS SORTIE: LOAD A1 PRINTI **end:**	**Réponses I.2.1** *nom de la fonction unaire calculée* ..... *paramètre initial* .....
(I.2.2) (2pts) A2 1.000000 B2 1.000000 C2 0.000000 D2 0.000000 **begin:** PUSH 2.000000 STORE D2 RABAT: PUSH 77.000000	LOAD D2 JG EXIT LOAD B2 LOAD A2 ADD STORE C2 LOAD B2 STORE A2 LOAD C2 STORE B2	PUSH 1.000000 LOAD D2 ADD STORE D2 JMP RABAT EXIT: LOAD C2 PRINTI **end:**	**Réponses I.2.2** *nom de la fonction unaire calculée* ..... *paramètre initial* .....

I.3) Compléter les pointillés par les instructions 0-address code manquantes :

Partie 1	Partie 2	
(I.3.1) (2pts) hauteur_triangle 15.000000 base_triangle 10.000000 surface_triangle 0.000000 **begin:** LOAD hauteur_triangle LOAD base_triangle MULT .....*(I.3.1.1)* .....*(I.3.1.2)*	DDIV STORE surface_triangle PRINTS "Surface Trangle dont la Hauteur et la Base sont " LOAD hauteur_triangle PRINTI LOAD base_triangle PRINTI PRINTS " est = " LOAD surface_triangle PRINTI **end:**	*nom de la fonction binaires calculée surface d'un triangle* *paramètre :* hauteur **15cm**, base **10cm** *instruction (I.3.1.1) manquante* ..... *instruction (I.3.1.2) manquante* .....

(I.3.2) (2pts) rayon_cercle 0.000000 surface_cercle 0.000000 pi 3.141593 begin: PUSH 100.000000 STORE rayon_cercle LOAD pi LOAD rayon_cercle DUPL	.....*(I.3.2.1)* .....*(I.3.2.2)* STORE surface_cercle PRINTS "Surface Cercle de Rayon (" LOAD rayon_cercle PRINTI PRINTS ") = " LOAD surface_cercle PRINTI end:	***nom de la fonction binaires calculée*** *surface d'un cercle* ***paramètre :*** rayon **100cm** ***instruction (I.3.2.1) manquante*** ..... ***instruction (I.3.2.2) manquante*** .....

Exercice II — Algorithmique (11pts)

Écrivez les lettres **A, B, C ou D** devant chaque proposition ou remplir les pointillées à l'endroit désigné

Concept/Question/Algorithme	**Compléter les pointillés**
(II.**1**) Compléter l'algorithme de calcul pour la valeur 120 de la fonction Factorielle écrit dans le langage de programmation pédagogique Hortensias par l'instruction correspondante (Hortensias n'est pas sensible à la casse, REM désigne un commentaire sur 1 ligne et = l'affectation) (1pt) : n int 10; facto int 1; i int; **BEGIN** **FOR** i = 1 **TO** 120 **DO** facto = **REM Compléter cette instruction dans la Réponse II.1.1** **ENDFOR** **PRINT** facto; **END**	**Réponse II.1.1** facto = ...
(II.**2**) Compléter l'algorithme de recherche du zéro d'une fonction croissante par méthode dichotomique écrit dans le langage de programmation pédagogique Hortensias par les trois instructions correspondantes (/= désigne l'opérateur de comparaison booléenne ≠) (3pt): REM programme recherchant la solution f(x) = 0 pour f(x) = x**2 - 2.x - 100 sctictement croissante entre REM [minx, maxx]avec f(minx) * f(maxx) < 0 minx double -20.0; maxx double 20.0; min double; max double; i double ; **BEGIN** min = minx; max = maxx; i = (min + max) / 2; **WHILE** (((i * i) - (2 * i) - 100) /= 0) **DO** **IF** (((i * i) - (2 * i) - 100) * ((max * max) - (2 * max) - 100) < 0) **THEN** **REM Proposer cette instruction dans la Réponse II.2.1** **ELSE** **REM Proposer cette instruction dans la Réponse II.2.2** **ENDIF** **REM Proposer cette instruction dans la Réponse II.2.3** **ENDWHILE** PRINT I ; PRINT (i * i) - (2 * i) - 100; **END**	**Réponse II.2.1** ..... **Réponse II.2.2** ..... **Réponse II.2.3** .....
(II.**3**) Compléter l'algorithme de calcul pour la valeur 1000 de la fonction Fibonacci écrit dans le langage de programmation pédagogique Hortensias par les trois instructions correspondantes (3pts) : REM fibonacci REM grand pere Fibo(i=0) = 1 gp int 1; REM pere Fibo(i=1) = 1 p int 1; REM petit fils pf int 0; i int; **BEGIN** REM calcul de Fibo(i=1000) **FOR** i = 2 **TO** 1000 **DO** pf = **REM Compléter cette instruction dans la Réponse II.3.1** gp = **REM Compléter cette instruction dans la Réponse II.3.2** p = **REM Compléter cette instruction dans la Réponse II.3.3** **ENDFOR** **PRINT** pf; **END**	**Réponse II.3.1** pf = ... **Réponse II.3.2** gp = ... **Réponse II.3.3** p = ...

(II.**4**) Que Calcule l'algorithme suivant, écrit dans le langage de programmation pédagogique Hortensias ? (== désigne l'opérateur de comparaison booléenne =) (1pt)

```
s1 string "Bonjour, " ; s2 string "hello world";
I int 0; X int; Y int; Z int 0; W bool; A double 1.0; B double 1 ; C double 1.0; N int 5473; VI int 0; VD Double ; D int;
BEGIN
        PRINT s1;          PRINT "Algorithme Mystère appliqué à" ; PRINT N;
        FOR I = 2 TO 74 DO
                X = ((74 - I) + 2);
                Y = N - ((N / X) * X) ;
                IF (Y == 0) THEN   Z = Z + 1 ;   D = X ;        ENDIF
        ENDFOR
        IF (Z == 0) THEN
          PRINT "True :-) " ;
          FOR I = 1 TO 180 DO
                C = C * N;
                A= 2 * A* I * 0.5;
                B = B + (1 / (A/ C));
          ENDFOR
          PRINT "The image is = " ; PRINT B;
        ELSE PRINT "False :-(" ; PRINT "here =" ; PRINT D; PRINT "there =" ; PRINT Z;
        ENDIF
END
```

Réponse II.4.1. concise

.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....

(II.**5**) Compléter par les instructions correspondantes, l'algorithme de calcul, pour la valeur 15, de la fonction Exponentielle par puissance du nombre d'Euler et approximée par la méthode de développement limité, écrit dans le langage de programmation pédagogique Hortensias (3pts) :

```
X int 15 ;          I int 1;          FACTO double 1.0;          EXPO_X double 0.0 ;
PUISSANCE double 1.0;                 NOMBRE_EULER double 2.71828182845904523536;
TAUX_ERREUR DOUBLE;
BEGIN
FOR I = 1 TO 180 DO
        PUISSANCE = REM Compléter cette instruction dans la Réponse II.5.1
        FACTO = REM Compléter cette instruction dans la Réponse II.5.2
        EXPO_X = EXPO_X + (PUISSANCE / FACTO);
ENDFOR
PRINT EXPO_X;
REM Calcul de l'exponentielle par puissance du nombre d'Euler
PUISSANCE = 1.0;
FOR I = 1 TO 15 DO
        REM Compléter cette instruction dans la Réponse II.5.3
ENDFOR
PRINT PUISSANCE;
TAUX_ERREUR = (EXPO_X - PUISSANCE) / PUISSANCE ;
PRINT TAUX_ERREUR;
END
```

Réponse II.5.1

PUISSANCE =

.....

Réponse II.5.2

FACTO =

.....

Réponse II.5.3

Exercice III — Structure de données et Programmation en Langage (10pts)

Remplir les pointillées à l'endroit désigné

Concept/Question/Script	Compléter les pointillés
(III.1) Soit la fonction C reverse qui inverse une chaîne de caractères, compléter par les instructions correspondantes en C cette fonction et la fonction myitoa qui transforme un nombre entier en chaînes de caractères (2pts): `#include <string.h>` `#include <stdlib.h>` `void reverse(char s[]){` `  int i, j;` `  char c;` `  for (i = 0, j = strlen(s)-1; i < j; i++, j--) {` `    c = s[i];` `    s[i] = s[j];` `    /* Proposer cette instruction dans la Réponse III.1.1 */` `  }` `}` `char * myitoa(int n){` `  int i = 0;  char s[100];  char * result;` `  do {` `    s[i++] = /* Compléter cette instruction dans la Réponse III.1.2 */` `  } while ((n /= 10) > 0);` `  s[i] = '\0';` `  reverse(s);` `  result = (char *) malloc(strlen(s) + 1);` `  strcpy(result, s);` `  return result;` `}`	Réponse III.1.1 ..... Réponse III.1.2 s[i++] =.....
(III.2) Soit la fonction C substring qui retourne une sous-chaîne d'une chaîne de caractère en entrée qui commence à partir d'une position donnée (>=1) et d'une taille donnée (>=1), compléter par l'instruction correspondante : (1pt) `char *substring(char *string, int position, int length)` `{` `  char *pointer;` `  int c;` `  pointer = malloc(length+1);` `  for (c = 0 ; c < length ; c++){` `    /* Proposer cette instruction dans la Réponse III.2.1 */` `    *(pointer+c) = *(string+position-1);` `    string++;` `  }` `  *(pointer+c) = '\0';` `  return pointer;}`	Réponse III.2.1 .....
(III.3) Que fait le script shell unix suivant (en toronto cshell, $< signifie lecture au clavier, @ précède toute variable numérique en modification, $ précède toute variable en lecture, / désigne une division Euclidienne et % désigne l'opérateur de modulo (congruence)) : `#!/bin/tcsh` `@ other = 2` `touch complex.log` `@ other--` `echo -n "entrez un"` `@ here =$<` `@ other++` `@ there = 157 * $other` `while ($here > 0)` `@ real = (2 * $there * $here) / 100` `@ imaginary = (2 * $there * $here) % 100` `echo "Resultat de "$here " => " $real"."$imaginary` `echo  "Resultat de "$here " => "$real"."$imaginary >> complex.log` `echo -n "entrez un"` `@ here =$<` `continue` **end** (1pt)	Réponse III.3.1 concise ....

Écrivez les lettres **A, B ou C** devant chaque proposition :

Concept/Question/Script	Compléter par A, B, C
(III.4) #include <stdio.h> #include <stdlib.h> #include <string.h> typedef enum { False=0 , True=1} **boolean** ; typedef void* **ptrgenerique**; typedef boolean **Comparateur** (ptrgenerique e1, ptrgenerique e2) ; typedef Comparateur* **ptrComparateur**; boolean **fnc1** (int *e1, int *e2) { if (*e1 < *e2) return True; else return False; } boolean **fnc2** (char *e1, char * e2) { if (strcmp(e1,e2) < 0) return True; else return False; } boolean **fnc3** (float *e1, float *e2) { if (*e1 < *e2) return True; else return False;} boolean **generique**(ptrgenerique e1, ptrgenerique e2, ptrComparateur C){ return (*C) (e1,e2); } int main(){ int * e1 = (int*) malloc(sizeof(int)); int * e2 = (int*) malloc(sizeof(int)); (*e1) = 4; (*e2) = 5; boolean Resultat = **generique**((ptrgenerique) e1, (ptrgenerique)e2, (ptrComparateur)&fnc1); printf("%s\n",Resultat==True?"true":"false"); //... } **L'instruction printf affichera false. (A) Vrai, (B) Faux.** (1pt)	.....
(III.5) Resultat = **generique**((ptrgenerique)"abcd", (ptrgenerique)"aabyclo", (ptrComparateur)&fnc2); printf("%s\n",Resultat==True?"true":"false"); **L'instruction printf affichera false. (A) Vrai, (B) Faux.** (1pt)	.....
(III.6) float * e21 = (float*) malloc(sizeof(float)); float * e22 = (float*) malloc(sizeof(float)); (*e21) = 100.6; (*e22) = 100.8; Resultat = **generique**((ptrgenerique)e21, (ptrgenerique)e22, (ptrComparateur)&fnc3); printf("%s\n",Resultat==True?"true":"false");**L'instruction printf affichera false. (A) Vrai, (B) Faux.** (1pt)	.....
(III.7) Soit le type boolean défini plus haut et soit la fonction C échanger qui interchange le contenu entier de deux emplacements mémoire. On définit f1 comme : void **echanger**(int *a, int *b){ int temp = *b; *b = *a; *a = temp; } void **f1**(int * t, int taille){ int i = 0, j, mini, n=taille; while (i < n){ mini = i; j = i+1; while (j < n) { if (t[mini] > t[j]) mini = j; j++; } Echanger(&t[i], &t[mini]); i++; } } **La fonction f1 est une fonction C de (A) tri à bulles, (B) tri par selection, (C) de tri par insertion** (1pt)	.....
(III.8) Soit le type boolean défini plus haut. On définit f2 comme : void **f2**(int * t, int taille){ int interne, externe, l = taille; for (externe = 1; externe < l; externe++) { int e = t[externe]; interne = externe; while ((interne > 0) && (t[interne - 1] >= e)) { t[interne] = t[interne - 1]; interne--; } t[interne] = e; } } **La fonction f2 est une fonction C de (A) tri à bulles, (B) tri par selection, (C) de tri par insertion** (1pt)	.....
(II.9) Soit le type boolean défini plus haut, la fonction echanger définie plus haut. On définit f3 comme : void **f3**(int * t, int taille){ int n=taille; boolean modification = False; do{ modification = False; for(int i=0; i<n-1; i++){ if (t[i] > t[i+1]){ **echanger**(&(t[i]), &(t[i+1])); modification = True; }} n--; }while(modification == True); } **La fonction f3 est une fonction C de (A) tri à bulles, (B) tri par selection, (C) de tri par insertion** (1pt)	.....