Consolidation

Questions

1) Un étudiant pose la question suivante : Pourquoi avoir le concept constant dans un langage de Programmation puisqu'on peut déclarer une variable et ne pas changer son contenu tout au long D'un programme ? Proposer lui une explication.

On utilise le concept constant dans un langage de programmation dans le but de fixer cette valeur et de ne pas déclarer une autre fois pour moins utilisation pour la mémoire et le temps d'exécution.

2) Un étudiant implémente une application en utilisant le langage de programmation C. Il l'exécute sur son PC équipé du Windows 10. Il trouve un problème lorsqu'il lance l'exécutable de son application sur le Macintosh du son ami. Expliquer le problème et donner une solution.

Le système d'exploitation Windows est diffèrent au système d'exploitation Macintosh

Par exemple en Windows on utilise \ or en Macintosh on utilise / .Alors à l'exécution on va rencontrer des problèmes.

La solution :

3) Pourquoi on utilise un tableau pour sauvegarder des éléments du même type et non pas réserver plusieurs variables?

On utilise ces types tableaux pour enregistrer les adresses des variables en cas de les appelle.

4) Pourquoi en langage C le passage d'un tableau comme paramètre dans l'invocation d'une fonction est considérée toujours comme passage par adresse ?

Le passage d'un tableau comme paramètre d'une fonction est impossible en tant que valeur : la recopie du tableau prendrait trop de temps et de place. On passe donc à la fonction l'adresse du tableau, ce qui permettra à la fonction d'effectuer des lectures et des écritures DIRECTEMENT DANS LE TABLEAU.

5) Donner l'interprétation des déclarations suivantes :

int a, *p, T[5], **q, *F[10], ***r, **L[20];
a=entier

p=pointeur

T=tableau de 5 entiers

q=adresse

F=tableau de 10 adresses entiers

r=pointeur

L=tableau de 20 adresses entiers

6) - Soient les déclarations suivantes :

```
struct Point
{
float x;
float y;
};
struct Personne
{
char nom[20];
char prenom[20];
unsigned age;
};
int *a,**b;
float *f;
Double ***d;
struct Point *p1, **q;
struct Personne *ps;
```

Comparer:

sizeof(int*)= allocation dynamique d'entiers

sizeof(int)= allocation dynamique d'adresses entiers**

sizeof(float*)= allocation dynamique de reels

sizeof(double*)= allocation dynamique d'adresses de reels**

sizeof(struct Point *)= allocation dynamique de structure Point

sizeof(struct Point **)= allocation dynamique d'adresses de structure Point

sizeof(struct Personne*)= allocation dynamique de structure Personne

7) Soit le programme en C suivant :

```
#include<stdio.h>
struct Point
{
float x;
float y;
};
void main()
{
struct Point *p1,*p2;
scanf("%f",&p1.x); scanf("%f",&p1.y);
scanf("%f",&p2.y); scanf("%f",&p2.y);
}
```

a. Corriger le programme pour qu'il permette la saisie de deux points tout en gardant la partie déclarative.

```
scanf("%f",&p1.x); scanf("%f",&p1.y);
scanf("%f",&p2.x); scanf("%f",&p2.y);
```

b. Donner la signification de cette instruction *p2=*p1;

la valeur de p1 est égale à la valeur p2.

c. Modifier le programme pour qu'il soit capable de lire N points en les sauvegardant dans un tableau P tout en réservant l'espace mémoire nécessaire.

```
int N=??;
int P[??];
for( int i=0 ; i<N ; i++)
    scanf("%f",&P[i].x); scanf("%f",&P[i].y);
```

8) Discuter la validité des définitions suivantes du type composé struct element :

struct element

{

int valeur;

struct element p;

};

→Cette formule est valide. Elle est correcte.

struct element

{

int valeur;

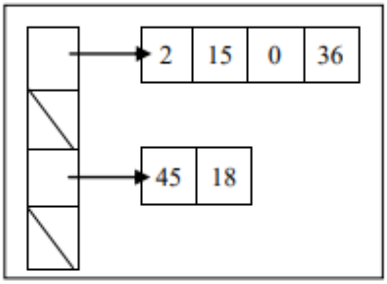
struct element * p;

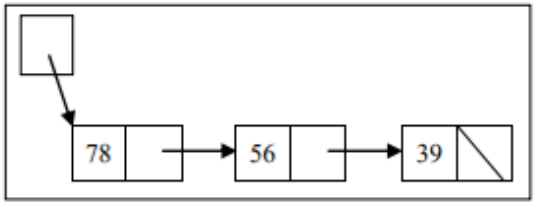
}Element, *element ;

```
typedef struct StackElement
    {
        int value;
        struct StackElement *next;
    }StackElement, *Stack;
```

→Cette formule n'est pas valide car il y'a manque d'une autre declaration.

9) Donner le programme C permettant d'implémenter ces représentations de la mémoire ci-dessous : la flèche signifie un pointeur pointant sur une variable.





10)

```
#include<stdlib.h>
void main()
{
int A[5]={2,8,9,7,4};
free(A);
}
```

free(A) est elle un instruction correcte? Expliquer.

free(A) n’est pas une instructyion correcte car on utilise seulement en malloc.

(La fonction « free » sert à restituer l'espace que l'on avait alloué avec "malloc". Nous vous conseillons d'utiliser cette fonction autant de fois que la fonction "malloc" pour libérer la place allouée dynamiquement.)

12) Ecrire une fonction en C permettant de retourner le maximum et le minimum à partir d'un Tableau T de N entiers.

```c
void MinMax(int V[],int N,int *min,int *max)
{
    int i;
    min=V[0];
    max=V[0];
    for(i=0;i<N;i++)
    {
        if(V[i]<*min)
        *min=T[i];
        if(V[i]>*max)
        *max=T[i];
    }
}
```

13) Ecrire une fonction en C permettant de supprimer toutes les occurrences d'un entier x à partir d'un tableau T de N entiers.

```c
void Supprimmer_les_occurrnces(int T[],int N)
{
    int A,*P1,*P2
    for (P1=P2=A; P1<A+N; P1++)
     {
      *P2 = *P1;
      if (*P2 != X)
           P2++;
     }
}
```

Exercice 1

Écrire une fonction en C permettant de remplir un tableau T de N entiers distincts.

```c
#include<stdio.h>


void Remplir(int T[],int N)
{
    int i;
    for(i=0;i<N;i++)
    {
        printf("donner l'elementdu tableau V[%d]\n",i);
        scanf("%d",&T[i]);
    }
}
void main()
{
    int max,min;
    int T[10];
    Remplir(T,10);
}
```

Exercice 2

Écrire une fonction en C permettant de remplir un tableau T de N entiers distincts et ordonnés dans l'ordre croissant.

```c
#include<stdio.h>
 void Remplir(int T[],int N)
{
    int i,j;
    i=0;
    j=1;
    While((i<j) && (T[i]<=T[j]))
    {
        for(i=0;i<N;i++)
        {
             printf("donner l'elementdu tableau V[%d]\n",i);
             scanf("%d",&T[i]);
        }
        j++;
    }
}
```

```c
void main()
{
    int max,min;
    int T[10];
    Remplir(T,10);
}
```

Exercice 3

Ecrire une fonction en C permettant de vérifier si un tableau T de N entiers est trié ou non dans l'ordre croissant.

```c
#include<stdio.h>

 void Remplir(int T[],int N)
{
    int i;
    for(i=0;i<N;i++)
    {
        printf("donner l'elementdu tableau V[%d]\n",i);
        scanf("%d",&T[i]);
    }
}
void Verif(int T[],int N)
{
    int i,j;
    int verif=0;
    for(i=0;i<N-1;i++)
     {
         for(j=1;j<N;j++)
            {
                if(T[i]<=T[j])
                    verif=1;
                 else
                    verif=0;
            }
     }
    if(verif=0)
     printf("tableau non trié \n");
    else if(verif=1)
     printf("tableau trié \n");
}
void main()
```

```
{
    int max,min;
    int T[10];
    Remplir(T,10);
    Verif(T,10);
}
```

Exercice 4

Ecrire une fonction en C permettant de changer la disposition de N entiers dans un Tableau T pour maximiser la différence entre la somme des valeurs se trouvant dans le premier moitié du tableau et la somme des éléments du deuxième moitié :

$$\max\left(\sum_{i=0}^{i=n/2} T[i] - \sum_{i=\frac{n}{2}+1}^{n-1} T[i]\right)$$

```
#include<stdio.h>
int MAX(int T[10],int N)
{
    int i;
    int som1=0,som2=0;
    for(i=0;i<N/2;i++)
         som1+=T[i];
    for(j=N/2;j<N;j++)
         som2+=T[j];
    MAX=som1-som2;
    return(MAX);
}
```

Cordialement Arous Achraf (LSI 1 A1-1).