Cahier de TD/TP

Génie Logiciel

Correction partielle

Qualité et maintenance du code

1°/ Analyse du code

Soit le code source suivant (Langage C):

```
#include<stdio.h>
#define TM 100
int main(void){
float x,v,c[TM];
int i,n;
float x0,s;
scanf("%d",&n);
for(i=0;i<=n;i++){scanf("%f",&c[i]);}
scanf("%f",&x);
v= c[0];x0=x;s=v;
for(i=1;i<=n;i++){s=s+c[i]*x0;x0=x0*x;}
v=s;printf("%f",v);
return(0);}
```

Analysez le code, que fait-il ?

$i = 0 \qquad s = c[0] + c[1] * x$

$i = 1 \qquad s = c[0] + c[1] * x + c[2] * x^2$

$i = 1 \qquad s = c[0] + c[1] * x + c[2] * x^2 + c[3] * x^3$

$i = n \qquad s = \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots + c[n] * x^n$

$= c[0] + \sum_{i=1}^{n} c[i] * x^i$

$= \sum_{i=0}^{n} c[i] * x^i$

Quelles sont les améliorations que vous pouvez apporter à ce code ?

1. Supprimer des variables inutiles (ex: s)
2. Réutilisation: créer une fonction polynomial
3. IHM *(Interface Homme Machine)* (Ex: printf avant scanf)
4. Gestion des exceptions
5. Commentaires
6. Optimisation → espace / fonctions C existances / dimensionnement des variables
7. Indentation
8. Nommage des variables / fonctions

A savoir: une fonction recursive prends plus de mémoire qu'une fonction normale

2°/ Recherche d'erreurs

Le code source suivant correspond à un programme qui utilise la fonction *compare (a, b)* permettant la comparaison de deux chaînes de caractères a et b.
La fonction retourne une valeur négative, nulle ou positive selon que « a » est lexico-graphiquement inférieure, égale ou supérieure à b.

```
#include<stdio.h>
#define TM 30

int compare(char a[TM],char b[TM]){
      int i = 0;
      while((a[i]==b[i]) && ((a[i]!='\0') && (b[i]!='\0'))){
            i++;
      }

      if (a[i]==b[i]) return(0);
      else if (a[i]=='\0' || b[i]=='\0\0') return(-2);
      else if (a[i]>b[i]) return(1);

      return(-1);
}

int main(void){
      char a[TM],b[TM];
      int comp;
      printf("Saisissez un mot a et un mot b:");
      scanf("%s %s", &a a, &b b);
      comp = compare(a,b);

      switch (comp)
      {
            case 0: printf("%s et %s sont égaux! \n", a, b); break;
            case 1: printf("%s est lexico-graphiquement inferieure a
            %s\n",a,b); break;
            case (-1): printf("%s est lexico-graphiquement supérieure
            a %s\n",b,a a, b); break;
            default: printf("Il y a inclusion!\n");
      }
}
```

Analysez et corrigez les erreurs qui se trouvent dans le code. Quelles sont les améliorations que vous pouvez apporter à ce code ?

Correction en rouge.

3°/ Optimisation

Un client aimera disposer d'un programme qui lui permet de calculer le nombre de combinaisons possible de *p* objets parmi *n*. La fonction est donnée par la formule suivante :

$$C_n^p = \frac{n!}{p!\,(n-p)!}$$

Le client propose le test suivant (méthode main) pour vérifier le programme :

```
int main(void){
        int n,p;
        double c;
        printf("Calculons C(n,p)\n");
        printf("Entrez la valeur de n ");
        scanf("%d",&n);
        printf("Entrez la valeur de p ");
        scanf("%d",&p);
        c=factoriel(n)/(factoriel(p)*factoriel(n-p));
        printf(" C(%d,%d) vaut %lf", n,p,c);
        printf("\n");
        return(0);
}
```

Proposez une solution pour ce client.
Peut-on proposer une solution optimisée ?
Est-ce qu'on peut avoir une solution sans utiliser l'opération de multiplication* ?

$$* : C_n^p = C_{n-1}^{p-1} + C_{n-1}^p$$