Les sous-programmes

Un sous-programme est une unité de conception et de programmation. Dans un processus de développement d'un algorithme (processus de raffinage).

Conception

Une action représentant le programme est décomposée en actions plus élémentaires. Chacune de ses actions peut être représentée par un sous-programme, càd un nom et une liste de paramètres qui expriment les entrées et les sorties.

Une action peut être utilisée dans des différents endroits dans un programme. Le sous-programme va nous permettre de réutiliser cette action à travers son nom et une liste de paramètres dits effectifs.

Exemple

Ecrire un programme qui calcule pour M>=N>=0 la combinaison C_M^N.

comprendre

La définition des combinaisons est la suivante : = M!/(N!*(M-N)!)

spécification

/* M>=N>=0*/ combinaison(M,N,c) /* c= M!/(N!*(M-N)!) */

modèle de solution

on peut commencer par calculer M!
puis N! et ensuite (M-N)!
après ces calcules nous pouvons effectuer la multiplication et la divisision

Développement

1er niveau

```
/* M>=N>=0 */
fm = fact(M) ;
fn = fact(N) ;
```

```
fmn = fact(M-N)
c = fm/(fn*fmn) ;
/* c= M!/(N!*(M-N)!) */
```

2ème niveau il suffit de développer le sous-programme fact :

```
int fact(int N) { if (N==0) return 1 ; else return N*fact(N-1) ; }
```

Programmation

Deux façons de programmer un sous-programme. Ces deux façons dépendent de son utilisation. Si l'utilisation du sous-programme, dans le processus de raffinage, retourne une valeur cette action sera programmer comme une fonction sinon s'il est utilisée comme une instruction alors dans ce cas il sera programmer comme une procédure.

Un sous-programme est composé d'un profil qui représente sa partie visible ou son interface et par un corps qui représente sa partie implémentation.

Fonction : Définition

Est un sous programme qui décrit :
le type de la valeur de retour
un identificateur
une liste de paramètres d'entrées

type_retour identificateur (paramètres formels en in) {**corps ;}**

Est utilisée comme une expression

x = identificateur(paramètres effectifs) ;

le type de x et type_de_retour doivent être les mêmes

Quand on appel une fonction on doit préciser les valeurs et les variables sur les quelles la fonction va effectuer son exécution. On les appelle paramètres effectifs qui correspondent en nombre, en type et en ordre aux paramètres formels.

Procédures

Est un sous programme qui décrit :
void comme type de retour
un identificateur
une liste de paramètres d'entrées, de sorties et d'entrée/sortie

interface

void identificateur (liste de paramètres formels **e, s, e/s**
) {**corps ;**}

utilisation

identificateur(liste de paramètres effectifs) ;

Quand on appel une procédure on doit préciser les valeurs, en entrée, et les variables, en entrée, sortie ou entrée/sortie, sur les quelles la procédure va effectuer son exécution. On les appelle paramètres effectifs qui correspondent en nombre, en type et en ordre aux paramètres formels.

Exemple

Ecrire un programme qui effecue l'échange entre deux variables de type char

Solution 1 :
void xch (char x, char y) { char t ; t = x ; x = y ; y = t ; }

Faire un appel à xch dans un programme sur deux variables (c et d comme paramètres effectifs) pour vérifier l'échange :

```
{char c, d ;
c ='a' ; d ='b' ; xch(c,d) ;
}
```

A l'appel de xch trois cases (x, y et t) sont réservées dans la pile d'exécution :

3 étapes :

1-

x contient la valeur de c

y contient la valeur de d

t contient ?

2-

On exécute le corps de xch sur x, y et t

3-

On retourne à l'instruction qui suit l'appel de xch(c,d)

on trouve que c et d sont inchangées.

Solution 2

En langage C :

int x ;

signifie une case réservée qui s'appelle x et qui peut mémoriser des entiers :

x = 5 ;

&x

signifie l'adresse de la variable x

int * px ;

signifie une case réservée qui s'appelle px et qui peut mémoriser des adresses de variables de type int.

Exemple :

px = &x ;

comment accéder x à travers px : *px

Nous utilisons cette indirection pour accéder à une variable pour pouvoir modifier les paramètres effectifs dans le cas où le paramètre est une sortie ou une entrée : sortie.

void xch (char *x, char *y) { char t ; t = *x ; *x = *y ; *y = t ; }

le faire exécuter sur un exemple :

char c, d ;

c =’a’ ; d =’b’ ; xch(&c,&d) ;

A l’appel de xch trois cases (x, y et t) sont réservées dans la pile d’exécution :

3 étapes :

1-

x contient la valeur &c : l’adresse de c

y contient la valeur &d : l’adresse de d

t contient ?

2-

On exécute le corps de xch sur *x, *y, qui représentent effectivement c et d, et t

3-

On retourne à l’instruction qui suit l’appel de xch(&c,&d)

on trouve que c et d sont échangées.