TP 9 : Préprocesseurs et macros

Programmation en C (LC4)

Semaine du 24 mars 2008

1 Utilisation du préprocesseur

Exercice 1 Ecrire les macros suivantes :

- Définissant une constante PI
- Calculant le cube d'un nombre
- Calculant la somme de deux nombres
- Calculant le volume d'une sphère ($V = (4\pi R^3)/3$).

Vous utiliserez ces macros dans un main permettant d'afficher la somme des volumes de N sphères, dont le rayon est passé en paramètre du programme.

Vous pourrez utiliser la fonction `double atof(char * chaine)` qui convertit une chaine de caractère en nombre flottant.

► **Correction**

```
#include <stdio.h>
#include<stdlib.h>

#define PI 3.1415927
#define cube(r) r*r*r
#define somme(a,b) a+b
#define volume(r) (4 * PI * cube(r))/3

int main(int argc, char** argv) {
  int i;
  float vol_sphere_i, vol_total=0;
  int N = argc - 1;
  for(i=1;i<=N;i++)
  {
    vol_sphere_i =  volume(atof(argv[i]));
    printf ("le volume de la sphere %d est %f \n", i, vol_sphere_i);
    vol_total = somme(vol_total, vol_sphere_i);
  }
  printf("Le volume total est %f\n",vol_total);
  return 1;
}
```

2 Compilation conditionnelle

Exercice 2 La compilation conditionnelle permet d'orienter la compilation suivant certains critères (compilation sous différentes plateformes par exemple), ici nous allons voir comment remédier aux dépendances cycliques entre modules grâce à la compilation conditionnelle.

- Créez cinq fichiers `tata.h`, `tata.c`, `toto.h`, `toto.c`, et `main.c`.
- Dans `toto.h` sera définit la fonction `somme` qui calcule la somme de deux entiers, cette fonction sera réalisée dans `toto.c`.
- Dans `tata.h` sera définit la fonction `produit` qui calcule le produit de deux entiers, cette fonction sera réalisée dans `tata.c`.
- Le fichier `tata.h` doit dépendre du fichier `toto.h` et réciproquement.
- le fichier `main.c` appelera ces deux fonctions.
- Ecrivez la Makfile qui permet de compiler l'ensemble de ces fichiers. Que se passe t-il ?

– Remédiez à cette erreur en utilisant le compilation conditionnelle.

► **Correction**

toto.h

```
#ifndef TOTO_H
#define TOTO_H
#include"tata.h"

int somme(int x,int y);

#endif
```

toto.c

```
int somme(int x,int y){
  return x + y;
}
```

tata.h

```
#ifndef TATA_H
#define TATA_H
#include"toto.h"

int produit(int x,int y);

#endif
```

tata.c

```
int produit(int x,int y){
  return x*y;}
```

main.c

```
#include"toto.h"
#include"tata.h"

int main(int argc, char ** argv){
  int a=2;
  int b=3;
  int c,d;
  c=somme(a,b);
  d=produit(d,a);
  return 1;
}
```

makefile

```
CC=gcc
EXEC=circulaire
circulaire: tata.o toto.o main.o
	$(CC) -o circulaire tata.o toto.o main.o

tata.o: tata.c toto.h
	$(CC) -o tata.o -c tata.c

toto.o: toto.c tata.h
	$(CC) -o toto.o -c toto.c

main.o: main.c toto.h tata.h
	$(CC) -o main.o -c main.c
```

```
clean:
	rm -rf *.o
```

3 Programmation modulaire

Exercice 3 Le but de cette exercice est de masquer l'implémentation d'un vecteur et de n'utiliser que les fonctions définies dans l'interface.

- Définir dans un fichier `type.h` un type vecteur composé de trois rééls, ainsi que les fonctions `creer_vecteur` qui crée un vecteur, et `coord_1`,`coord_2`,`coord_3` qui renvoient respectivement la première, deuxième et troisième coordonnée du vecteur qui leur est passé en paramètre. Ces fonctions seront réalisées dans le fichier `type.c`
- Définir dans un fichier `norme.h` l'en-tête d'une fonction `norme` prenant en argumant un vecteur et renvoyant sa norme. Cette fonction sera réalisée dans le fichier `norme.c`.
- Définir dans un fichier `produits.h` les en-têtes des fonctions `produit_scalaire` et `produit_vectoriel` prenant en argumant deux vecteurs et renvoyant respectivement leur produit saclaire et vectoriel. Ces fonctions seront réalisées dans le fichier `produits.c`.
  - Notons les veteurs $x = (x_1, x_2, x_3$ et $y = (y_1, y_2, y_3$
  - Norme : $\| x \| = \sqrt{x_1^2 + x_2^2 + x_3^2}$
  - Produit scalaire : $x.y = x_1 * y_1 + x_2 * y_2 + x_3 * y_3$
  - Produit scalaire : $x \wedge y = (x_2 * y_3 - x_3 * y_2, x_3 * y_1 - x_1 * y_3, x_1 * y_2 - x_2 * y_1)$
- Ecrire dans un fichier `combinaison.c` un programme qui à partir de deux vecteurs et qui calcule la norme de leur produit vectoriel.
- Ecrire un `Makefile` qui permet de compiler l'ensemble de ces fichiers.

► **Correction**

type.h

```
#ifndef TYPE_H
#define TYPE_H

#include<stdio.h>
#include<stdlib.h>

struct vecteur_s;

typedef struct vecteur_s *vecteur;

vecteur creer_vecteur(float coord_1,float coord_2,float coord_3);
float coord_1(vecteur  x);
float coord_2(vecteur  x);
float coord_3(vecteur  x);

#endif
```

type.c

```
#include "type.h"

struct vecteur_s{
  float x_1;
  float x_2;
  float x_3;
};

vecteur creer_vecteur(float coord_1,float coord_2,float coord_3)
{
  vecteur x = malloc(sizeof(struct vecteur_s));
  x->x_1= coord_1;
  x->x_2 = coord_2;
```

```
  x->x_3 = coord_3;
  return x;
}

float coord_1(vecteur x)
{
  return x->x_1;
}

float coord_2(vecteur x)
{
  return x->x_2;
}

float coord_3(vecteur x)
{
  return x->x_3;
}
```

norme.h

```
#ifndef NORME_H
#define NORME_H

#include<math.h>
#include "type.h"

float norme(vecteur x);

#endif
```

norme.c

```
#include "norme.h"

float norme(vecteur x)
{
  return sqrt(pow(coord_1(x),2) + pow(coord_2(x),2) + pow(coord_3(x),2));
}
```

produits.h

```
#ifndef PRODUITS_H
#define PRODUITS_H

#include "type.h"

float produit_scalaire(vecteur x, vecteur y);
vecteur produit_vectoriel(vecteur x, vecteur y);

#endif
```

produits.c

```
#include "produits.h"

float produit_scalaire(vecteur x, vecteur y)
{
 return coord_1(x)*coord_1(y) + coord_2(x)*coord_2(y) + coord_3(x)*coord_3(y);

}

vecteur produit_vectoriel(vecteur x, vecteur y)
```

```
{
 vecteur z;
 float z_1,z_2,z_3;
 z_1 = coord_2(x) * coord_3(y) - coord_3(x) * coord_2(y);
 z_2 = coord_3(x) * coord_1(y) - coord_1(x) * coord_3(y);
 z_3 = coord_1(x) * coord_2(y) - coord_2(x) * coord_1(y);
 z = creer_vecteur(z_1,z_2,z_3);
 return z;
}
```

combinaison.c

```
#include "norme.h"
#include "produits.h"

int main(int argc, char ** argv)
{
vecteur x = creer_vecteur(2,3,8);
vecteur y = creer_vecteur(4,9,6);
float norme_vectorielle = norme(produit_vectoriel(x,y));

printf("La norme du produit vectoriel de x par y est %f\n",norme_vectorielle);


return 1;
}
```

Makefile

```
CC=gcc
LDFLAGS=-lm
EXEC=combinaison

all:$(EXEC)

combinaison: type.o norme.o produits.o combinaison.o
		$(CC) -o combinaison type.o norme.o produits.o combinaison.o $(LDFLAGS)
type.o: type.c type.h
		$(CC) -o type.o -c type.c
norme.o: norme.c norme.h type.h
		$(CC) -o norme.o -c norme.c
produits.o: produits.c produits.h type.h
		$(CC) -o produits.o -c produits.c
combinaison.o: combinaison.c type.h norme.h produits.h
		$(CC) -o combinaison.o -c combinaison.c

clean:
		rm -rf *.o
```