Projet première partie

1 Calcul matriciel en C/C++

Objectif : créer une librairie dynamique et une librairie statique de calcul matriciel en C et en C++.

1.1 Organisation d'un projet

Afin de faciliter le développement du projet, il est vivement conseillé de créer une arborescence du projet dans le répertoire home pour chaque cas. L'exemple suivant montre l'arborescence du projet de librairie dynamique en C après avoir tout compiler :

```
projetmatricedynamique
+-- include
|   +-- matrice.h
+-- lib
|   +-- libmatrices.so
+-- src
|   +-- Makefile
|   +-- matrice.c
|   +-- matrice.o
+-- test
    +-- main.c
    +-- main.o
    +-- Makefile
    +-- test
```

Le répertoire test contient le programme de test de la librairie.

1.2 Librairies C

1.2.1 Création d'une librairie dynamique en C

Une librairie dynamique est une libraire partagée d'extension .so. Cette librairie est accompagnée d'un fichier de prototypes .h. La compilation se fait avec l'option -fPIC. La création de la librairie dynamique se fait avec l'option -shared.

Fichier Makefile du répertoire src :

```
CC=gcc
CFLAGS= -Wall  -I../include -fPIC -std=gnu99
LDFLAGS=
SRC=$(wildcard *.c)
OBJS= $(SRC:.c=.o)

all: $(OBJS)

%.o: %.c
    $(CC) $(CFLAGS) -c -o $@ $<

lib:
    $(CC) -shared  -o ../lib/libmatrices.so $(OBJS)

clean:
    rm *.o ../lib/libmatrices.so
```

Fichier Makefile du répertoire test :

```
CC=gcc
CFLAGS= -Wall -I../include -std=gnu99
LDFLAGS= -L../lib -lmatrices
EXEC=test
SRC= $(wildcard *.c)
OBJS= $(SRC:.c=.o)

all: $(OBJS)
    $(CC) -o $(EXEC) $^ $(LDFLAGS)

%.o: %.c
    $(CC) $(CFLAGS) -c -o $@ $<

clean:
    rm *.o $(EXEC)
```

Lors de l'exécution du programme de test, il faut préciser au chargeur de librairie du programme de test le chemin pour accéder à la librairie en initialisant la variable d'environnement `LD_LIBRARY_PATH` avec le chemin d'accès à la libraire partagée du projet.

Exemple :

```
export LD_LIBRARY_PATH=../lib;./test
```

1.2.2 Création d'une librairie statique en C

Une libraire statique est une librairie qui contient les fichiers objets d'extension .a et également accompagnée d'un fichier de prototypes .h. La création de la librairie statique se fait avec la commande ar.

Fichier Makefile du répertoire src :

```
CC=gcc
AR=ar
CFLAGS= -Wall -I../include -std=gnu99
LDFLAGS=
SRC=$(wildcard *.c)
OBJS=$(SRC:.c=.o)

all: $(OBJS)

%.o: %.c
    $(CC) $(CFLAGS) -c -o $@ $<

lib:
    $(AR) -cvq ../lib/libmatrices.a $(OBJS)

clean:
    rm *.o ../lib/libmatrices.a
```

Fichier Makefile du répertoire test :

```
CC=gcc
CFLAGS= -Wall -I../include
LDFLAGS= -L../lib -lmatrices
EXEC=test
SRC= $(wildcard *.c)
OBJS= $(SRC:.c=.o)

all: $(OBJS)
    $(CC) -o $(EXEC) $^ $(LDFLAGS)

%.o: %.c
    $(CC) $(CFLAGS) -c -o $@ $<

clean:
    rm *.o $(EXEC)
```

1.2.3 Contenu de la librairie C

La librairie doit contenir au minimum la création de matrice aléatoire, les opérations d'addition, soustraction et multiplication. Les fichiers doivent être suffisamment commentés afin de pouvoir utiliser facilement la librairie.
Pour le trinôme, il est demandé d'ajouter les fonctions de factorisation LU, et d'inversion de matrice.

1.2.4 Programme de test

Le programme doit permettre de tester l'ensemble des fonctions de calcul matriciel implémentées dans la librairie.

1.3 Librairies C++

1.3.1 Création d'une librairie dynamique en C++

Une librairie dynamique est une libraire partagée d'extension .so. Cette librairie est accompagnée d'un fichier de prototypes .h. La compilation se fait avec l'option -fPIC. La création de la librairie dynamique se fait avec l'option -shared.
Fichier Makefile du répertoire src :

```
CC=g++
CFLAGS= -Wall  -I../include -fPIC
LDFLAGS=
SRC=$(wildcard *.cpp)
OBJS= $(SRC:.cpp=.o)

all: $(OBJS)

%.o: %.cpp
    $(CC) $(CFLAGS) -c -o $@ $<

lib:
    $(CC) -shared  -o ../lib/libmatrices.so $(OBJS)

clean:
    rm *.o ../lib/libmatrices.so
```

Fichier Makefile du répertoire test :

```
CC=g++
CFLAGS= -Wall -I../include
LDFLAGS= -L../lib -lmatrices
EXEC=test
SRC= $(wildcard *.cpp)
OBJS= $(SRC:.cpp=.o)

all: $(OBJS)
    $(CC) -o $(EXEC) $^ $(LDFLAGS)

%.o: %.cpp
    $(CC) $(CFLAGS) -c -o $@ $<

clean:
    rm *.o $(EXEC)
```

Lors de l'exécution du programme de test, il faut préciser au chargeur de librairie du programme de test le chemin pour accéder à la librairie en initialisant la variable d'environnement `LD_LIBRARY_PATH` avec le chemin d'accès à la libraire partagée du projet.
Exemple :

```
export LD_LIBRARY_PATH=../lib;./test
```

1.3.2 Création d'une librairie statique en C++

Une libraire statique est une librairie qui contient les fichiers objets d'extension .a et également accompagnée d'un fichier de prototypes .h. La création de la librairie statique se fait avec la commande ar.

Fichier Makefile du répertoire src :

```
CC=g++
AR=ar
CFLAGS= -Wall  -I../include
LDFLAGS=
SRC=$(wildcard *.cpp)
OBJS= $(SRC:.cpp=.o)

all: $(OBJS)

%.o: %.cpp
    $(CC) $(CFLAGS) -c -o $@ $<

lib:
    $(AR) -cvq ../lib/libmatrices.a $(OBJS)

clean:
    rm *.o ../lib/libmatrices.so
```

Fichier Makefile du répertoire test :

```
CC=g++
CFLAGS= -Wall -I../include
LDFLAGS= -L../lib -lmatrices
EXEC=test
SRC= $(wildcard *.cpp)
OBJS= $(SRC:.cpp=.o)

all: $(OBJS)
    $(CC) -o $(EXEC) $^ $(LDFLAGS)

%.o: %.cpp
    $(CC) $(CFLAGS) -c -o $@ $<

clean:
    rm *.o $(EXEC)
```

1.3.3 Contenu de la librairie C++

La librairie doit contenir une Classe Matrice, qui permet de créer une matrice aléatoire, d'effectuer les opérations d'addition, de soustraction et de multiplication en incluant la surcharge des opérateurs. Les fichiers doivent être suffisamment commentés afin de pouvoir utiliser facilement la librairie.
Pour le trinôme, il est demandé d'ajouter une classe MatricePlus qui hérite de Matrice et ajoute les méthodes qui permettent d'effectuer la factorisation LU et d'inverser une Matrice.

1.3.4 Programme de test

Le programme doit permettre de tester l'ensemble des méthodes et opérations matricielles implémentées dans la librairie

2 ESP32, SPI, I2C

2.1 Objectif

Implémenter les programmes de gestion de capteurs sur l'ESP32. Donner les performances des programmes implémentés qu niveau de l'empreinte mémoire.

2.2 Travail à réaliser

2.2.1 Communication I2C

Utilisation de la librairie I2C

1. Ecrire le programme qui permet de lire les valeurs du module I2C pmodhygro. Il convient d'utiliser les registres du capteur et non une librairie trouvée sur internet. Ce programme enverra sur la liaison série les valeurs de température et d'humidité.
2. Relever l'empreinte mémoire du programme.

2.2.2 Communication SPI

Utilisation de la librairie SPI.

1. Ecrire le programme qui permet de lire la pression atmosphérique et la température sur le module SPI pmodnav. Il convient d'utiliser les registres du capteur et non une librairie trouvée sur internet. Ce programme enverra ces valeurs sur la liaison série.
2. Relever l'empreinte mémoire de chaque programme.
3. Pour le trinôme, ajouter une fonctionnalité de calcul de l'altitude à partir de la pression et de la température en cherchant sur internet (wikipédia par exemple) une méthode de calcul.

Projet deuxième partie

3 ESP32 et Wifi

3.1 Objectif

Implémenter la communication wifi sur l'esp32 afin de transmettre des données des capteurs à un serveur HTTP.

3.2 Travail à réaliser

1. Ecrire le programme le base qui effectue une connexion cliente sur une borne WIFI et affiche les paramètres réseaux de la connexion.
2. Ecrire le programme client HTTP qui permet d'envoyer des données à un serveur HTTP qui implémente un script CGI fourni.

4 Linux et drivers sur raspberry PI

4.1 Objectif

Implémenter un driver de gestion de gpios sur la raspberry PI. Ce driver permet de commander un port 8 bits (nommé JA, JB, JC) à partir de la commande echo et de lire les valeurs de ce port 8 bits avec la commande cat. Le numéro du port (JA, JB, JC) dépend du numéro mineur. La direction du port est choisie avec les paramètres transmis lors du chargement du driver.

4.2 Travail à réaliser

Il est conseillé de réaliser l'écriture du driver étape par étape.

1. Ecrire le driver qui permet d'accéder au port JC avec la commande echo. La valeur des 8 bits est transmise en base 10 ou 16 avec le préfixe 0x.
2. Ajouter le choix d la direction du port (entrée ou sortie) en utilisant un paramètre lors du chargement du driver.
3. Ajouter le choix du port (JA,JB,JC) en fonction du numéro mineur.

5 Réseaux de neurones

5.1 Objectif

L'objectif est d'implémenter un réseau de neurones monocouche puis avec une couche cachée sur l'ESP32. Les données de ce réseau seront doit des GPIOS soit des données de capteurs. Les résultats pourront être affichés sur des GPIOs ou affichés sur le terminal série.

5.2 Perceptron en C et C++

1. Ecrire une classe C++ Perceptron qui implémente le calcul du résultat d'un perceptron (fichiers Perceptron.cpp et Perceptron.h).
2. Intégrer cette classe dans un projet ESP32 afin de vérifier les apprentissages fournis en cours.
3. Relever l'empreinte mémoire des programmes.

5.3 Perceptron multi-couches en C et C++

1. Ecrire une classe C++ MLP qui implémente, la fonction d'activation est une tangente hyperbolique.
2. Intégrer cette classe dans un projet ESP32 afin de vérifier les apprentissages fournis en cours.
3. Relever l'empreinte mémoire des programmes.