# Utilisation de Make & Makefile

Licence Informatique : Programmation Nécessaire

Stéfane

## Table des Matières



## 1 Complétez

*Récupérez et complétez le fichier* `sujet.c` *qui est constitué de trois parties distinctes:*

*La partie déclaration*

```c
/**********************
 *      PARTIE I      *
 * TA et déclarations *
 **********************/

#define NBCOMPMAX 20

/** @brief le type abstrait modélisant
 * les vecteurs réels d'au plus NBCOLMPMAX composantes
 *
 * @param nbComp : nombre effectif de composantes
 * @param values : le vecteur contenant les coefficients réels
 */
typedef struct vector {
  int nbComp;
  double values[NBCOMPMAX];
} vector_t;

/** @brief Déclarations des fonctions et procédures PUBLIQUES:
    - scanVector crée un nouveau vecteur
    - printVector affiche le vecteur
    - prodScal calcule le produit scalaire
    - prodInt calcule le produit interne
*/
vector_t scanVector();
void printVector (vector_t V);
double prodScal(vector_t V1, vector_t V2);
vector_t prodInt(vector_t V, double lambda);
```

*La partie principale (`main`)*

```c
/*************************
 *      PARTIE II        *
 * La fonction principale *
 *************************/
int main() {
  vector_t V = scanVector();
  printVector(V);

  double lambda;
  printf("lambda ? ");
  scanf(" %lf", &lambda);

  vector_t V2 = prodInt(V,lambda);
  printVector(V2);
  printf(" prod scal = %lf\n", prodScal(V,V2));
  return 0;
}
```

*La partie définition*

```c
/*****************************
 *        PARTIE III         *
 *      Définitions des      *
 *  fonctions et procédures  *
 *****************************/
vector_t scanVector () {
  vector_t V;

  do {
    printf ("Donnez le nombre de composantes du vecteur  ( %d) : ", NBCOMPMAX);
    scanf ("%d", &V.nbComp);
  } while (V.nbComp < 1 || V.nbComp > NBCOMPMAX);

  for(int c = 0; c < V.nbComp; c += 1) {
    printf(" V[%d] ? ", c);
    scanf(" %lf", &(V.values[c]));
  }

  return V;
}

void printVector (vector_t M) {
  printf ("[");
  for (int c = 0; c < M.nbComp; c += 1) {
    printf (" %2.2lf", M.values[c]);
  }
  printf (" ]\n ");
}

double prodScal(vector_t V1, vector_t V2){
  double p = 0;
  // À COMPLÉTER
  return p;
}
vector_t prodInt(vector_t V, double lambda){
  vector_t lV;
  // À COMPLÉTER
  return lV;
}
```

# 2 Compilez

*Une fois le fichier* `sujet.c` *complété, compilez le dans un terminal avec la commande:*

```
gcc -o vector sujet.c -Wall -lm
```

*Explication. Le compilateur est* `gcc` *avec les options :*

- `-Wall` *: afficher les alertes ;*
- *Si besoin, vous pouvez ajouter l'option* `-lm` *pour utiliser la librairie mathématique.*

*Puis vérifiez son fonctionnement en lançant l'exécutable dans le terminal:*[1]

```
./vector
```

[1] *À la fin du sujet vous trouverez un exemple d'exécution.*

# 3 Les fichiers séparés

Créez trois sous-dossiers : `src`, `obj` et `include`.

Créez un nouveau fichier `main.c` dans le dossier `src`. Déplacer la fonction `main` dans ce nouveau fichier et ajouter en début de ce fichier la directive `#include "vector.h"`.

Puis, créez les fichiers suivants:

- Un fichier `vetor.h` dans le dossier `include` regroupant la définition du type abstrait (TA) `vector_t` et les déclarations des fonctions associées.

  Ce fichier doit *impérativement* commencer par

  ```
  #pragma once
  ```

**Remarque.**

Il s'agit d'une macro qui lorsque il y a plusieurs occurrences de

```
#include "vector.h"
```

ne fait effectivement qu'une seule fois l'inclusion.[2]

- Un fichier `vector.c` dans le dossier `src` regroupant les définitions des fonctions déclarées dans le fichier `vetor.h`.

  Au début du fichier, ajoutez la directive `#include "vector.h"` qui permet d'utiliser le TA `vector_t`.

**Indication.**

De cette manière les fonctions peuvent être définies et utilisées en tenant compte d'aucun ordre établi.

**Remarque.**

Pensez à faire toutes les autres inclusions nécessaires.

**Bon fonctionnement.**

Faites attention que le fichier des définitions `vector.c` et le fichier principal `main.c` soient dans le répertoire `src` (source) tandis que le fichier des déclarations `vector.h` soit dans le répertoire `include`.

Récupérez sur Arche le `Makefile` correspondant.

[2] Vous pouvez facilement le vérifier en mettant 2 `#include "pair.h"` à la suite dans le fichier `main.c`.

## Le fichier Makefile

```
IDIR = include
ODIR = obj
SDIR = src
BDIR = bin

CC = gcc
CFLAGS = -Wall -I$(IDIR) -c
LDFLAGS = -lm

PROG=$(BDIR)/mat

_DEPS = # À COMPLÉTER : liste des fichiers .h séparés par des espaces
DEPS = $(patsubst %,$(IDIR)/%,$(_DEPS))

_OBJS= # À COMPLÉTER : les listes des fichiers .o ayant chacun un correspondant .c
OBJS = $(patsubst %,$(ODIR)/%,$(_OBJS))

.PHONY : run all dirs clean delete

run : all
  ./$(PROG)

all : dirs $(OBJS)
  $(CC) -o $(PROG) $(OBJS) $(LDFLAGS)

$(ODIR)/%.o : $(SDIR)/%.c $(DEPS)
  $(CC) -o $@ $< $(CFLAGS)

dirs :
  @mkdir -p $(ODIR) $(BDIR)

clean :
  rm -rf $(ODIR) core

delete : clean
  rm -rf $(BDIR)
```

Dans le terminal lancez la commande `make`. Si tout est correcte le programme s'exécute.

```
Scripts$ ./vectorz
Donnez le nombre de composantes du vecteur (≤ 20) : 3
 V[0] ? 8
 V[1] ? 7
 V[2] ? 6
[ 8.00 7.00 6.00 ]
 lambda ? -2
[ -16.00 -14.00 -12.00 ]
  prod scal = -298.000000
Scripts$
```