S3:E2 — Utilisation de `Make` & `Makefile`

LICENCE INFORMATIQUE : PROGRAMMATION NÉCESSAIRE

Stéfane

1 Complétez

Récupérez le ficher `sujet.c` qui est constitué de trois parties distinctes :

1. Une partie *déclaration*
   - des types abstraits (TA)
   - des fonctions et procédures utilisées,

```c
/**********************
      PARTIE I
  TA et déclarations
**********************/
/** @brief
    Les types abstraits modélisant
    un point et un segment du plan
*/
struct point2d {
  double x, y;
};

struct segment2d {
  struct point2d O, E;
};

/** @brief
    Déclarations des focniotns et procédures :
    - newPoint2d crée un point du plan
    - newSegment2d crée un segment du plan
    - viewPoint2d affiche les coordonnées du point P
    - viewSegment2d affiche les coordonnées des points
      origine et extrémité du segment S
*/
struct point2d newPoint2d();
struct segment2d newSegment2d();
void viewPoint2d(struct point2d P);
void viewSegment2d(struct segment2d S);
```

2. Une partie uniquement constituée de la fonction principale `main`

```c
/**********************
      PARTIE II
  La fonction principale
**********************/
int main() {
  struct point2d P = newPoint2d();
  struct segment2d S = newSegment2d();
  viewPoint2d(P);
  viewSegment2d(S);
```

```c
  return 0;
}
```

3. Une partie *définition* des fonctions et procédures utilisées.

```c
/*************************
        PARTIE III
      Définitions des
  fonctions et procédures
*************************/
struct point2d newPoint2d() {
  struct point2d P;
  /** TODO **/
  return P;
}

struct segment2d newSegment2d() {
  struct segment2d S;
  /** TODO **/
  return S;
}

void viewPoint2d(struct point2d P) {
  printf("Le point de coordonnées (%g,%g)\n", P.x, P.y);
}

void viewSegment2d(struct segment2d S) {
  printf("Le segment\n\td'origine : ");
  /** TODO **/
}
```

Complétez les définitions des fonctions.

2 Compilation

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

```
gcc -Wall -std=c11 -o pt_sgt sujet.c
```

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

— `-Wall` *: All Warning ;*

— `-std=c11` *: pour utiliser le standard* `C11` *du langage* `C` *;*

— *Si besoin, vous pouvez ajouter l'option* `-g` *pour utiliser le debugger (GDB).*

Puis vérifiez son fonctionnement en lançant l'exécutable dans le terminal :

```
./pt_sgt
```

3 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 les directives `#include "point2d.h"` et `#include "segment2d.h"`

Puis, créez les fichiers suivants :

— Un fichier `poin2d.h` dans le dossier `include` regroupant la définition du type abstrait (TA) `struct point2d` et les *déclarations* des fonctions correspondantes.

Ce fichier doit **impérativement** commencer par

```
#ifndef _POINT2D_
#define _POINT2D_
```

et terminer par

```
#endif // _POINT2D_
```

Il s'agit de *macros* qui lorsque il y a plusieurs occurrences de

```
#include "point2d.h"
```

ne fait effectivement qu'une seule fois l'inclusion. [1]

— Un fichier `segment2d.h` dans le dossier `include` regroupant la définition du TA `struct segment2d` et les *déclarations* des fonctions correspondantes.

Ce fichier doit **impérativement** commencer par

```
#ifndef _SEGMENT2D_
#define _SEGMENT2D_
```

et terminer par

```
#endif // _SEGMENT2D_
```

— Un fichier `point2d.c` dans le dossier `src` regroupant les *définitions* des fonctions déclarées dans les fichiers `point2d.h`.

1. Vous pouvez facilement le vérifier en mettant 2 `#include "point2d.h"` à la suite dans le fichier `main.c`.

Au début du fichier, ajoutez la directive `#include "point2d.h"` qui permet d'inclure le TA et les déclarations des fonctions.

Indication.

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

— Un fichier `segment2d.c` dans le dossier `src` regroupant les *définitions* des fonctions déclarées dans les fichiers `point2d.h`.

Au début du fichier, ajoutez la directive `#include "segment2d.h"` qui permet d'inclure le TA et les déclarations des fonctions.

Remarque.

Pensez à faire toutes les autres inclusions nécessaires.

Bon fonctionnement.

Faites attention que les fichiers des définitions `point2d.c`, `segment2d.c` et le fichier principal `main.c` soient dans le répertoire `src` (source) et que les fichiers des déclarations `point2d.h` et `segment2d.h` soit dans le répertoire `include`.

Créez un répertoire `obj` où les objets seront rangés lors de la première phase de compilation par `gcc`. (Avant l'édition de lien)

Récupérez sur Arche le `Makefile` correspondant.

```
# dossier des entêtes et TA
IDIR = include
# dossier des objets de la 1ère phase de compilation
ODIR = obj
# dossier des définitions des fonctions
SDIR = src
# dossier des binaires générés
BDIR = bin

# le compilateur
CC = gcc
# les options de compilation
CFLAGS = -g -Wall -std=c99 -I$(IDIR)
# Les librairies utilisées (math)
LFLAGS = -lm

# le programme final
_PROG = pt_sgt2d
# On ajoute include/ à _PROG |==> include/pt_sgt
PROG = $(patsubst %,$(BDIR)/%,$(_PROG))

_DEP = # TODO Les noms des fichiers d'entêtes
DEP = $(patsubst %,$(IDIR)/%,$(_DEP))

_OBJ = # TODO Les noms des fichiers objets
OBJ = $(patsubst %,$(ODIR)/%,$(_OBJ))

# Toutes les étiquettes qui ne sont pas des fichiers sont déclarées ici
.PHONY: run dirs clean delete

```

```
run : $(PROG)
  ./$(PROG)

# S'ils n'existent pas créer les dossiers bin et obj
dirs:
  @mkdir -p $(BDIR)
  @mkdir -p $(ODIR)

# La cible est l'exécutable pt_sgt qui dépent des objets $(OBJ)
#   $@ désigne la cible $(PROG)
#   $^ désigne les dépendances $(OBJ)
$(PROG): $(OBJ)
  $(CC) $(LFLAGS) -o $@ $^

# La cible est un objet (se trouvant dans le répertoire obj)
# Les dépendances sont :
# (+) $(DEP) et
# (+) le fichier source : $(SDIR)/%.c où
#       % désigne le nom de la cible sans le suffixe .o
#     qui se situe dans le répertoire $(ODIR)
$(ODIR)/%.o: $(SDIR)/%.c $(DEP)
  $(CC) $(CFLAGS) -c -o $@ $<

# les cibles de nettoyage
clean :
  rm -rf $(ODIR)

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

Puis dans le terminal lancez la commande `make`. Si le tout est fait correctement le programme s'exécute.

```
(base) PC-UL:sujet$ gcc -g -Wall -std=c11 -o ptseg2d correction.c
(base) PC-UL:sujet$ ./ptseg2d
Donnez l'abcisse du point : 4.5
Donnez l'ordonnée du point : 6.2
Donnez l'abcisse de l'origine du segment : 3.6
Donnez l'ordonnée de l'origine du segment : 78.9
Donnez l'abcisse de l'extrémité du segment : -54.1
Donnez l'ordonnée de l'extrémité du segment : 0.6
Le point de coordonnées (4.5,6.2)
Le segment
        d'origine : Le point de coordonnées (-54.1,0.6)
        d'extrémité : Le point de coordonnées (4.5,6.2)
(base) PC-UL:sujet$
```