"Rappels" sur le Langage C

Rabie Ben Atitallah

ENSIAME IMS 2ème année 2017 - 2018

Langage C ?

- Langage de Programmation (**LP**) crée dans les années 70
- Le Système d'Exploitation UNIX repose sur le C (≈ 90%)
- Il fait parti des LP les plus utilisés, encore aujourd'hui
- Pourquoi le C ?
  - De nombreux langages plus récents en découlent (C++, Java, C#, …)
  - Langage référence pour la programmation bas niveau (réseaux, système)
  - Apprentissage de la gestion de la mémoire
  - Apprentissage des structures de données dynamiques

  ⇒ permet de **mieux aborder** des notions fondamentales en **POO**

Structure d'un programme

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

#define PI 3.1415

void ma_procedure(float rayon)
{
    float perimetre = 2.0 * PI * rayon ;
    printf("perimetre= %f\n", perimetre);
}

int main()
{
    float rayon;
    printf("Entrer le rayon : ");
    scanf("%f" , &rayon);
    ma_procedure(rayon);
    return 0;
}
```

1

inclusion de librairies

Structure d'un programme

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

#define PI 3.1415

void ma_procedure(float rayon)
{
    float perimetre = 2.0 * PI * rayon ;
    printf("perimetre= %f\n", perimetre);
}
```

2

déf. d'une constante

```
int main()
{
    float rayon;
    printf("Entrer le rayon : ");
    scanf("%f" , &rayon);
    ma_procedure(rayon);
    return 0;
}
```

Structure d'un programme

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

#define PI 3.1415

void ma_procedure(float rayon)
{
   float perimetre = 2.0 * PI * rayon ;
   printf("perimetre= %f\n", perimetre);
}
```

```
int main()
{
   float rayon;
   printf("Entrer le rayon : ");
   scanf("%f" , &rayon);
   ma_procedure(rayon);
   return 0;
}
```

3

déf. d'un sous-programme

Structure d'un programme

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

#define PI 3.1415

void ma_procedure(float rayon)
{
    float perimetre = 2.0 * PI * rayon ;
    printf("perimetre= %f\n", perimetre);
}
```

```c
int main()
{
    float rayon;
    printf("Entrer le rayon : ");
    scanf("%f" , &rayon);
    ma_procedure(rayon);
    return 0;
}
```

4

déf. du progr. principal

Quelques Règles

- Identificateurs : **mot** de la forme
  - Une lettre (+ le souligné _) suivi d'un nombre qque. de lettres et chiffres
  - Pas d'accent en C
  - Distinction majuscules / minuscules
- Un ; doit être mis à la fin des instructions
  - #include et #define sont **des directives de précompilation** ⇒ **pas de ;**
- Commentaires ⇒ **lisibilité, documentation**

  /* ceci est un commentaire en C */

Quelques Règles

- Caractères spéciaux
  - { } ! # , ; ~ . + - * / % < = > ( ) [ ] ' …
- Indentation libre (retraits en début de ligne) ⇒ **lisibilité**

```
int main()
{
    int i;
    printf("Entrer un entier : ");
    …
}
```

Mots Réservés en C

auto	**else**	long	**typedef**
break	enum	register	union
case	extern	**return**	unsigned
char	**float**	short	**void**
const	**for**	signed	volatile
continue	goto	sizeof	**while**
default	**if**	static	…
do	inline	**struct**	
double	**int**	switch	

Types Simples

- Booléen (norme **C99**) : bool ⇒ **#include <stdbool.h>**
- Caractère : char
- Entier : int
- Réel : float
- Remarque
  - Il existe plusieurs types différents pour coder les entiers et les réels, selon la précision nécessaire
  - short, long, + unsigned, signed pour les entiers
  - double pour les réels

Types Simples

- Booléen (norme **C99**) : bool ⇒ **#include <stdbool.h>**
- Caractère : char
- Entier : int
- Réel : float
- Remarque
  - Il existe plusieurs types différents pour coder les entiers et les réels, selon la précision nécessaire
  - short, long, + unsigned, signed pour les entiers
  - double pour les réels

Types Entier en C

```
int main()
{
   printf(" taille short = %d\n", sizeof( short ));                          2
   printf(" taille unsigned short = %d\n", sizeof( unsigned short ));        2
   printf(" taille int = %d\n", sizeof( int ));                              4
   printf(" taille unsigned int = %d\n", sizeof( unsigned int ));            4
   printf(" taille long int = %d\n", sizeof( long int ));                    4
   printf(" taille unsigned long int = %d\n", sizeof( unsigned long int ));  4
   printf(" taille long long int = %d\n", sizeof( long long int ));          8
   printf(" taille unsigned long long int = %d\n", sizeof( unsigned long long int ));  8
   return 0;
}
```

Types Entier en C

short :	-32768 et 32767
unsigned short :	0 et 65535
int :	-2147483648 et 2147483647
unsigned int :	0 et 4294967295
long int :	-2147483648 et 2147483647
unsigned long int :	0 et 4294967295
long long int :	-9223372036854775808 et 9223372036854775807
unsigned long long int :	0 et 18446744073709551615

Déclaration

- **Constante**
  - #define < identificateur > < valeur >

  #define MAX 100 #define toto "du texte" #define v 'v'

  💣 : le compilateur remplace l'identificateur par tout ce qui suit sur la ligne
- **Variable**
  - < **type** > < liste des identificateurs >

    int a ; int i, j ;
  - < **type** > < identificateur = constante (du type) >

    int i = 0 ; int i, j = 1 ; int i = 0, j = 1, k ;

    char c = 'c' ; char c = '.' ;

    double m = 0.0 ;

Opérateurs …

- … sur les entiers et les caractères
  - +, -, *, /, **%** 13 = **2** * 5 + **3** ⇒ 13 / 5 = **2** et 13 % 5 = **3**
- … sur les réels
  - +, -, *, /
- … sur des opérandes entiers ou booléens
  - && (**et**), || (**ou**), ! (**négation**)
- … de comparaison
  - == (**égalité**), != (**différent**), <, <=, >, >=
- … de manipulation de bits
  - << (**déc. à gauche**), >>, & (**et**), | (**ou inclusif**), ^ (**ou exclusif**), ~ (**complément**)

Opérateurs …

- … d'affectation : = (**ne pas confondre avec** ==)
  - identificateur = expression i = 5 ; j = i + 1 ;
  - Cas particuliers : identificateur **op**= expression
    avec **op** parmi : +, -, *, /, %, <<, >>, &, |, ^ i += 2 ; ⇔ i = i + 2 ;
  - Opérateurs ++ et – pour incrémenter /décrémenter une variable
    - i++ ; ⇔ i = i + 1 ;
    - **Attention** : i++ ; ≠ ++i ;

      i++ ; ⇒ on prend i, on l'utilise, puis on l'incrémente de 1

      ++i; ⇒ on prend i, on l'incrémente de 1, puis on l'utilise
- **Attention :** tous les opérateurs ont un certain niveau de priorité (dans le doute, mettre des ( ))

Lecture & écriture

- Lire (⇔ saisir des données) : **scanf**
  - scanf("chaine de spécification", &variable(s));

  La *chaine de spécification* dépend du type des variables
  - %d (entier) %f (réel) %c (caractère) %s (chaine de caractères) …
  - int i ; char v ; float r ; scanf("%c%d%f", &v, &i, &r);
- Ecrire (⇔ afficher à l'écran) : **printf**
  - printf("chaine à afficher à l'écran");
  - printf("chaine de spécification", variable(s));
  - printf("ceci est une chaine\n");
  - printf("ceci est une chaine avec la valeur d'un entier **%d** un reel **%f**\n", i, r);

Structures de Contrôle

- **Séquence :** début … fin { … }
- **Alternative :**
  - si (cond) alors … if (cond) … (pas de **then**)
  - si (cond) alors … sinon … if (cond) … else …
- **Tant que :**
  - tant que (cond) faire … while (cond) … (pas de **do**)
- **Faire … tant que :** (pas de répéter … jusqu'à en C) :
  - répéter … tant que (cond) ; do … while (cond) **;**
- **Pour :**
  - pour i de bi **à** bs pas **p** faire … for (i=bi; i **<=** bs; i**+=** **p**) …
  - pour i de bs **BAS** bi pas **p** faire … for (i=bs; i **>=** bi; i**-=** **p**) …

Sous-programmes

- La **déclaration** se fait **avant** le programme principal
- Il **ne peut pas être imbriqué** dans un autre sous-programme
- Procédure
  - < procédure > identificateur(< liste des paramètres en entrée et/ou sortie >)
  - **< void >** identificateur (< liste des paramètres >)
- Fonction
  - < fonction > identificateur (< liste des paramètres en entrée >) : < **type** >
  - < **type** > identificateur (< liste des paramètres >)
- Paramètres en entrée **par défaut**. En sortie ⇒ **pointeur**

Sous-programmes : exemples d'en-tête

- void compte_a_rebours (int debut)
- void affiche_nombres_pairs (int debut, int fin)
- void affiche_nombres_fibonacci (int limite)

- int est_premier (int nombre)
- float calcul_moyenne (int nnotes)
- int factorielle (int n)

Sous-programmes : exemples

```
/* affiche les valeurs de début à 0 à l'écran */
void compte_a_rebours (int debut)
{
   int i = debut ;
   while (i >= 0) {
      printf ("%d\n", i);
      i = i - 1 ;
   }
}
```

Sous-programmes : exemples

```
/* affiche les valeurs de début à 0 à l'écran */
void compte_a_rebours (int debut)
{
   int i = debut ;
   while (i >= 0) {
      printf ("%d\n", i);
      i-- ;
   }
}
```

Sous-programmes : exemples

```
/* affiche les valeurs de début à 0 à l'écran */
void compte_a_rebours (int debut)
{
    while (debut >= 0)
        printf ("%d\n", debut--);
}
```

Remarque : valeur de ***debut*** après la procédure ?

```
int main() {     compte_a_rebours(10);    return 0 ;      }
```

Sous-programmes : exemples

```
/* affiche les nombres pairs dans [debut ; fin] */
void affiche_nombres_pairs (int debut, int fin)
{
    int i = debut;
    do {
        if (i % 2 == 0) printf("%d est pair\n", i) ;
        i++;
    } while (i <= fin) ;
}
```

Sous-programmes : exemples

```
/* affiche les nombres pairs dans [debut ; fin] */
void affiche_nombres_pairs (int debut, int fin)
{
   int i;
   if(debut % 2 == 0) i = debut ;
   else i = debut + 1;
   for(i; i <= fin; i+=2) printf("%d est pair\n", i) ;
}
```

Sous-programmes : exemples

```
/* affiche les nombres de Fibonacci <= limite  (on suppose que limite est > 0) */
void affiche_nombres_fibonacci (int limite)
{
    int a = 0, b = 1, c = 0;
    do {
        printf("%d ", c) ;
        c = a + b ;
        a = b ;
        b = c ;
    } while (c <= limite) ;
}
```

Sous-programmes : exemples

```
/* retourne 1 si nombre est premier, 0 sinon */
int est_premier (int nombre)
{
   int i = 2;
   for(i=2; i < nombre; i++)
      if(nombre % i == 0) return 0;
   return 1;
}
```

Sous-programmes : exemples

```
/* retourne 1 si nombre est premier, 0 sinon */
int est_premier (int nombre)
{
   int i ;
   if (nombre % 2 == 0) return 0;
   for(i=3; i <= sqrt(nombre); i+=2)  /* avec #include <math.h> */
      if(nombre % i == 0) return 0;
   return 1;
}
```

Sous-programmes : exemples

```
/* saisit nnotes, puis calcule et retourne la moyenne */
float calcul_moyenne (int nnotes)
{
   int i = 1;
   float m = 0., note;
   while (i <= nnotes) {
        printf("Entrer la note %d : ", i);  scanf("%f", &note);
        m+= note;
        i++;
   }
   return m/(float)nnotes ;          /* conversion explicite */
}
```

Sous-programmes : exemples

```
/* calcule et retourne n! */
int factorielle (int n)
{
    int f = 1;
    while (n > 0) {
        f*= n;
        n--;
    }
    return f;
}
```

Sous-programmes : exemples

```c
#include <stdio.h>

int main()
{
    int i, j;
    compte_a_rebours(i);                    /* 💣 */
    scanf("%d", &i);
    affiche_nombres_pairs(i, 50);
    affiche_nombres_fibonacci(i/2);         /* arrondi à l'inférieur */
    printf("%d premier ? %d\n", i, est_premier(i));
    printf("moyenne= %f\n", calcul_moyenne(i));
    factorielle(i);                         /* 💣 */
}
```

Tableaux

- Pour définir **son propre type** : **typedef**
- Tableau à 1 dimension
  - CONST MAX = 50 ; — #define MAX 50
  - TYPE Tab = tableau [MAX] de Réel; — typedef float Tab [MAX] ;
- Accès à 1 élément du tableau ⇒ **opérateur [ ] (1er indice 0)**
  - VAR t : Tab; — Tab t ;
  - écrire (t[0], t[i]); — printf("%f\t%f\n", t[0], t[i]);
- Tableau à 2 dimensions

CONST MAX2 = 20 ; — #define MAX2 20

TYPE Tab2 = tableau [MAX][MAX2] de Caractère; — typedef charTab2 [MAX][MAX2] ;

Tableaux : exemple

- Tableau contenant au plus 10 entiers ⇒ type **Tab**

```
#define MAX 10
typedef int Tab[MAX] ;
void saisie(int n, Tab t)
{
    int i;
    for(i=0; i<n; i++) {
        printf("Case %d : ", i);
        scanf("%d", &t[i]);
    }
}
```

```
void affiche(int n, Tab t)
{
    int i = 0;
    while(i < n) {
        printf("t[%d] %d\n", i, t[i]);
        i++;
    }
}
```

Tableaux : exemple

- Tableau contenant au plus 10 entiers ⇒ type **Tab**

```
#define MAX 10
typedef int Tab[MAX] ;
int pos_max(int n, Tab t)
{
    int i, p = 0;
    for(i=1; i<n; i++)
        if(t[i] > t[p])
            p = i;
    return p;
}
```

```
int est_present(int n, int v, Tab t)
{
    int i = 0;
    while(i < n) {
        if(t[i] == v) return 1;
        i++;
    }
    return 0;
}
```

Tableaux : exemple

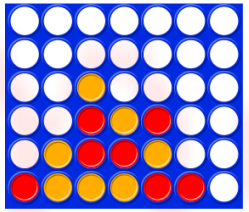
- Tableau contenant au plus 10 entiers ⇒ type Tab

```
#define MAX 10
typedef int Tab[MAX];

void saisie(int n, Tab t) { ...}
void affiche(int n, Tab t) { ...}
int pos_max(int n, Tab t) { ... }
int est_present(int n, int v, Tab t) { ... }
```

```
int main()
{
    int nb;                          Tab mon_tab;
    printf("Combien ? "); scanf("%d", &nb);
    saisie(nb, mon_tab);
    affiche(nb, mon_tab);
    printf("max= %d en position %d\n",
           t[pos_max(nb, mon_tab)],
           pos_max(nb, mon_tab));     /* 💣 */
    return 0;
}
```

Tableaux : exemple



- Simulation puissance 4 : tableau 6 * 7 ⇒ type Grille

```
#define NL 6
#define NC 7
typedef int Grille[NL][NC] ;

void initialise(int l, int c, Grille G)
{
    int i, j;
    for(i = 0; i < n; i++)
            for(j = 0; j < c; j++)
                G[i][j] = 0;
}
```

```
void affiche(int l, int c, Grille G)
{
    int i, j;
    for(i = 0; i < n; i++) {
            for(j = 0; j < c; j++)
                printf("%3d", G[i][j]);
            printf("\n");
    }
}
void ajoute_pion(int i, int j, int joueur, Grille G)
{
     G[i][j] = joueur;
}
```

Structures

- Pour définir **son propre type** : **typedef**
- Structure (ou enregistrement)

TYPE < **identifiant** > = enregistrement
    < champs1 > : < type1 > ;
    < champs2 > : < type2 >;
fin;

```
typedef struct {
        < type1 > champs1;
        < type2 > champs1;
} < identifiant > ;
```

- Accès à 1 élément d'une structure ⇒ **opérateur .**

Structures : exemples

- Structure pour représenter un nombre complexe :

```
typedef struct {
    float pReelle;
    float pImagin;
} Complexe ;
Complexe saisie()
{
    Complexe c;
    printf("Entrer les parties reelle imagin. :");
    scanf("%f%f", &c.pReelle, &c.pImagin);
    return c;
}
```

```
void affiche(Complexe c)
{
    printf("%f + i%f\n", c.pReelle, c.pImagin);
}
int egaux(Complexe c1, Complexe c2)
{
    return (c1.pReelle == c2.pReelle &&
            c1.pImagin == c2.pImagin);
}
```

Structures : exemples

- Structure pour représenter un nombre rationnel :

```
typedef struct {
   float num;
   float deno;
} Rationnel;
Rationnel somme(Rationnel r1, Rationnel r2)
{
   Rationnel r3;
   r3.num = r1.num * r2.deno + r2.num * r1.deno ;
   r3.deno = r1.deno * r2.deno;
   return r3;
}
```

```
Rationnel produit(Rationnel r1, Rationnel r2)
{
   Rationnel p;
   p.num = r1.num * r2.num;
   p.deno = r1.deno * r2.deno;
   return p;
}
```

Structures : exemples

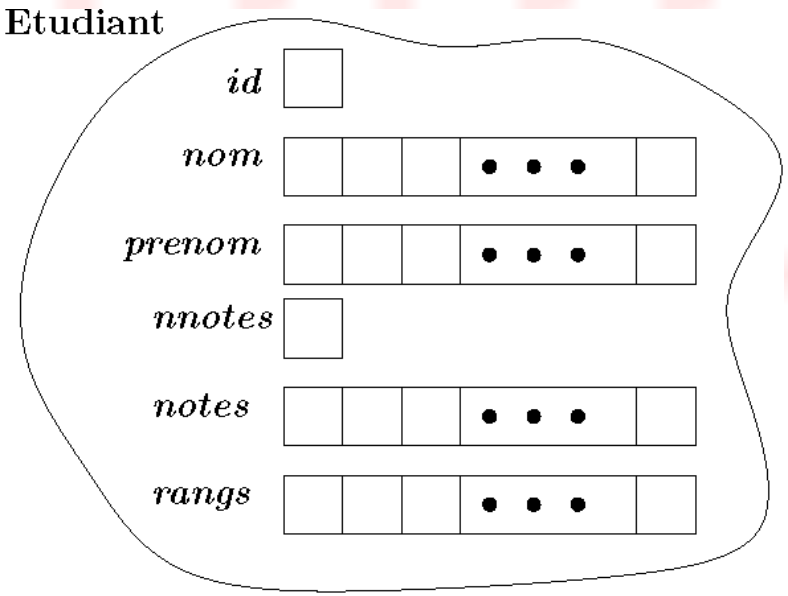
- Structure pour représenter un **Etudiant** : numéro d'étudiant, nom, prénom, notes dans les modules + moyenne, rang dans les modules + au semestre

```
#define MAXC 25
#define MAXN 15
typedef char Chaine [MAXC] ;
typedef int TRang[MAXN] ;
typedef double TNotes[MAXN] ;
```

```
typedef struct {
        long int id ;
        Chaine nom, prenom ;
        TNotes notes ;        int nnotes ;
        TRang rangs ;
} Etudiant;
```

Structures : exemples

```
typedef struct {
        long int id ;
        Chaine nom, prenom ;
        TNotes notes ;        int nnotes ;
        TRang rangs ;
 } Etudiant;
```



Etudiant E ; …

Accès au prénom de l'étudiant ⇒ E.prenom (**%s**)

Accès à la moyenne de la ième matière de l'étudiant ⇒ E.notes[i] (**%f ou %lf**)

Accès au rang au semestre de l'étudiant ⇒ E.rang[0]

Structures : exemples

- **Groupe TD** ⇒ tableau d'Etudiant + nombre d'étudiants

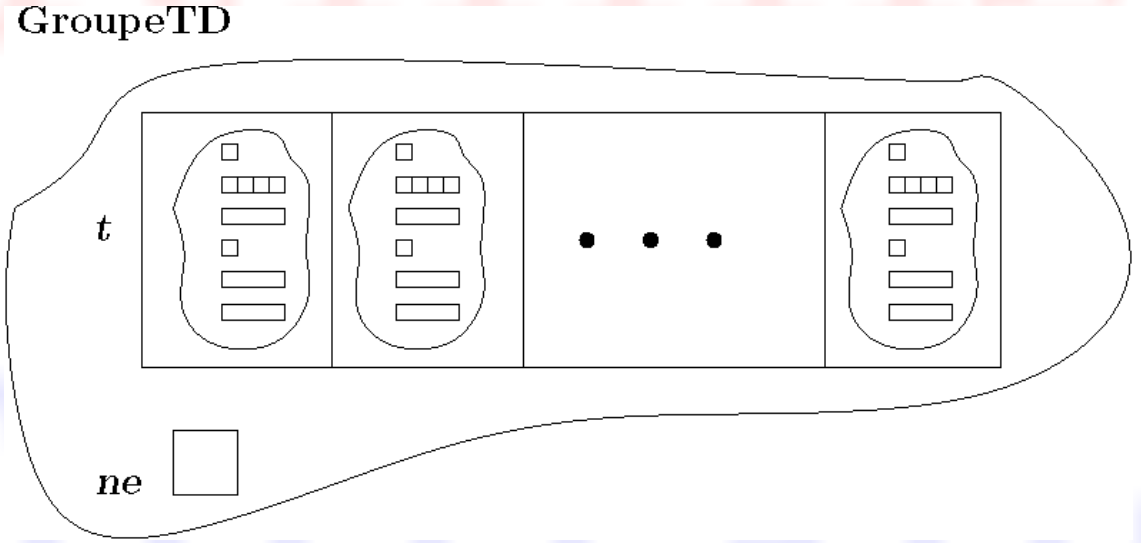
```
#define MAXC 25
#define MAXN 15
#define MAXE 40
typedef char Chaine [MAXC] ;
typedef int TRang[MAXN] ;
typedef double TNotes[MAXN] ;
typedef Etudiant TEtudiant[MAXE];
typedef struct {    TEtudiant t;  int ne;  } GroupeTD;
```

```
typedef struct {
        long int id ;
        Chaine nom, prenom ;
        TNotes notes ;        int nnotes ;
        TRang rangs ;
} Etudiant;
```

Structures : exemples

```
typedef struct {
        long int id ;
        Chaine nom, prenom ;
        TNotes notes ;
        TRang rangs ;
} Etudiant;
```

```
#define MAXE 40
typedef Etudiant TEtudiant[MAXE];
typedef struct { TEtudiant t; int ne; } GroupeTD;
```



GroupeTD G ; …

Accès au nombre d'étudiants du groupe

Accès à la moyenne de la ième matière de l'étudiant j ⇒ G.t[j].notes[i]

Accès au rang au semestre de l'étudiant j ⇒ G.t[j].rang[0]

Structures : exemples

- **Promo** ⇒ tableau de Groupes + nombre de groupes

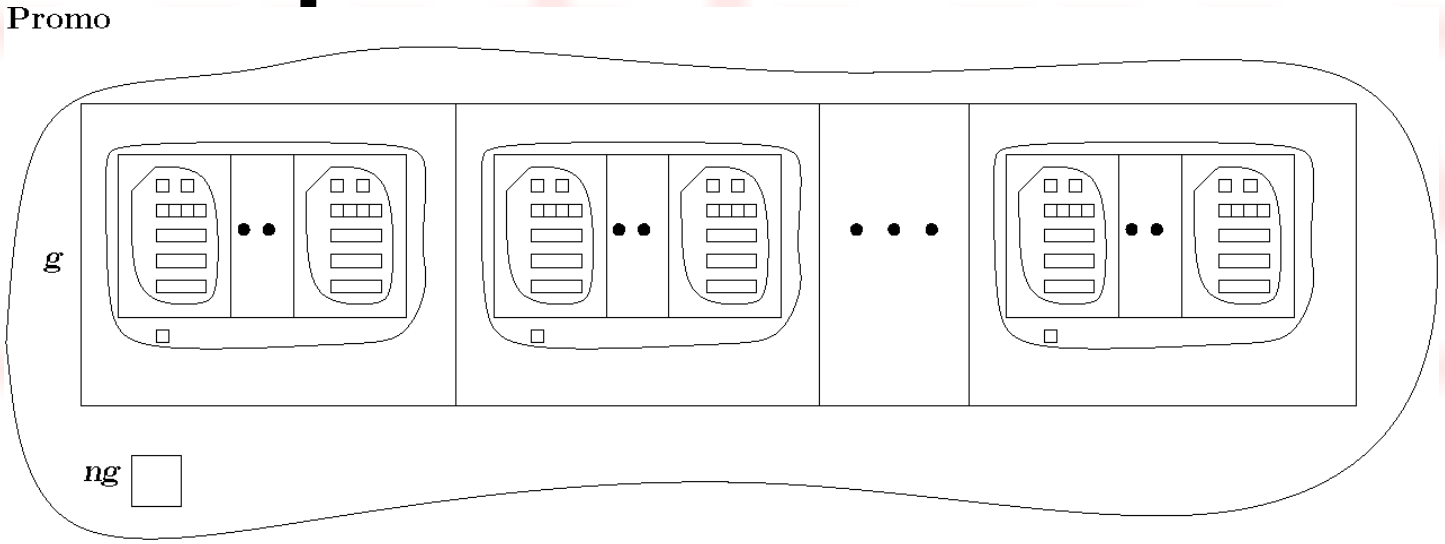
```
#define MAXC 25
#define MAXN 15
#define MAXE 40
#define MAXG 10
typedef char Chaine [MAXC] ;
typedef int TRang[MAXN] ;
typedef double TNotes[MAXN] ;
typedef Etudiant TEtudiant[MAXE];
typedef struct {      TEtudiant t;  int ne;  } GroupeTD;
typedef GroupeTD Groupes[MAXG];
```

```
typedef struct {
        long int id ;
        Chaine nom, prenom ;
        TNotes notes ;
        TRang rangs ;
} Etudiant;
```

```
typedef struct { Groupes g ; int ng; } Promo;
```

Structures : exemples

```
#define MAXG 10
typedef struct {
TEtudiant t;  int ne;
} GroupeTD;
```



typedef GroupeTD Groupes[MAXG]; typedef struct { Groupes g ; int ng; } Promo;

Promo P ; …

Accès au nombre d'étudiants du ième groupe ⇒ P.g[i].ne

Accès à la moyenne de la ième matière de l'étudiant j du groupe k ⇒ P.g[k].t[j].notes[i]

Accès au rang au semestre de l'étudiant j du groupe k ⇒ P.g[k].t[j].rang[0]