Memo : syntaxe de programmation en C

Eléments requis pour le cours d'algorithmique IUP1 et TI2

Structure de programme

```
/* Rôle :  */
/* Auteurs : */
/* Partie déclarations  */
#include <stdio.h>                  /* déclaration des bibliothèques */
#include  "mylib.h"                 /* déclarations de bibliothèque personnelles */
typedef nomdetype definition ;      /* déclarations de type */
#define constante valeur            /* déclaration de constante */
/* variables globales */
/* Procédures */
/* Programme principal*/
int main(void)              /* un seul programme principal */
{                           /* partie déclaration de variables locales */
                            /* partie instructions du programme principal unique dans un projet*/
    return 0;
}
```

Types de données prédéfinis

Catégorie	Type déclaré	format de conversion avec scanf et printf
Numériques :		
entiers signés	short int	%i ou %d ou %o (octal) ou %x (hexa)
	int	%i ou %d ou %o (octal) ou %x (hexa)
	long int	%li ou %ld ou %lo ou %lx
entiers non signés	unsigned int	%ui
réels	float	%f ou %e
double précision	double	%lf ou %le
Caractère	char	%c
Chaîne de caractères	char nom[taille] ;	%s
Booléens	utiliser le type int avec 0 pour faux et autre valeur pour vrai	

Déclaration de variable

Rôle: identifier et réserver de la place en mémoire pour stocker une valeur

Syntaxe de déclaration de variables

<*type_variable*> <*nom_variable*>,<*autre_variable*> **;**

Exemples de déclaration de variables numériques

int annee ; float x1, x2 ;

avec initialisation de valeur **int** jour=5, mois=11 ; float tva=17.6 ;

Pointeur :

Rôle : permet de gérer des adresses de variables

Déclaration : <*type_variable_pointée*> *<*nom_variable*> ;

Exemples

```
float *ptr; int *pti;       /* pointeur sur réel, sur entier */
char *str[];                /* pointeur sur tableau de caractères */
```

Utilisation

Affecter une adresse ptr=&x ; pti= &i ;

Affecter une valeur à une variable pointée *ptr=1.5 ; *pti=2 ;

Il n'est pas facile d'afficher la valeur d'un pointeur

La variable réelle pointée par ptr est symbolisée par (*ptr)

Transtypage (casting)

Rôle : lire le contenu de la mémoire comme si c'était un type différent (mais compatible) que celui déclaré initialement pour cette valeur

Automatique bien souvent car C n'est pas un langage fortement typé

Volontaire syntaxe : <*var_type1*> = **(<*nom_type1*>)** <*var_type2*> ;

Exemple :

```
long int li ; int i ;       /* déclarations */
li = (longint) i ;          /* instructions */
```

Type de données complexes

Il ne peut y avoir d'affectation globale pour ce type de données : il faut traiter séparément chaque composant

Tableau

Rôle: utiliser pour une liste de taille finie et connue de variables toutes de même type
Il faut connaître : le nombre d'éléments, le type de l'élément ;
Déclaration : <*type_elem*> <*nom_tab*>**[**<*nb_elem*>**]** ;
Exemples : char mot[10] ; float table[5] ;
Accès à un élement du tableau : par son indice qui commence à 0
Exemples : mot[3]='a' ; ou table[4]=1.5 ;
Adresse du premier élément du tableau : &<*nom_tab*>[0]
Exemple : &mot[0] ou &table[0]

Structure

Rôle : regrouper plusieurs informations de types différent au sein d'une même structure de donnée
Déclarations :**typedef struct {** <*type_champs*> <*nom_champs*>**}** <*type_struct*> **;**
Exemples : **typedef struct** {char nom[50] ; float note } **etudiant** ;
etudiant et ; /* déclaration de variable de type etudiant */
etudiant *ptr ; /* pointeur sur structure de type etudiant */
Accès aux champs d'une structure
Syntaxe : <*nom_variable*>.<*nom_champs*> ou
(*<ptr_sur_struct>**).**<champs> ou <*ptr_sur_struct*>**->**<*champs*>
Exemples strcpy(et.nom, "Dupond"); et.note=12.5 ;
strcpy((*ptr).nom, "Durand"); ptr->note=14.5 ;

Déclaration de type de données

Utiliser les types pour définir des catégories d'informations utilisables dans l'ensemble des programmes

Syntaxe **typedef** <*type*> <*nom_de_type*> **;**
Exemples **typedef** char ***string*** **;** */** définit le type chaîne de caractères */
typedef `int tab[3]; /* type tableau de 3 entiers tab */`
typedef `tab *ptrTab; /* pointeur sur tableau de 3 entiers ptrTab*/`
Utillisation tab t1,t2 ; /* déclaration de deux variables de type tab*/

Règles de nommage pour les identificateurs

Définition : séquence de lettres ('a'..'z') ou ('A'..'Z') et de chiffres ('0'..'9')
Sans espace ni caractère de ponctuation sauf '_'
De longueur quelconque (<31)
Commencer par un caractère
Différenciation entre majuscules et minuscules

Opérateurs pour les expressions

Affectation syntaxe <*lvar*>**=** <*expression*> ; exemple jour =31 ;
Affectation couplée avec opérateur : **+=** **-=** ***=** **/=**
Exemple : N += P ; équivaut à N = N+P ;

Unaires -
Incrémentation syntaxe <*lvar*>**++** **exemple : i ++** équivaut à i=i+1 ;
décrémentation syntaxe <*lvar*> **--** **exemple j--** équivaut à j=j-1 ;
Opération sur bits Décalage à droite >> à gauche <<
Et bit à bit & Ou bit à bit |
Adresse **&** exemple : &a signifie adresse de a
Indirection ***** exemple : *p signifie variable pointée par p
Vérifier sizeoff ???

Binaires

Expression numériques : plus **+** moins - produit ***** division **/** reste de division entière **%**
Comparateurs : égal **==** différent **!=** inférieur **<** inférieur ou égal **<=** supérieur **>** supérieur ou égal **>=** (attention : pas d'espace entre les deux caractères des comparateurs)
Exemple : a==3 est une expression conditionnelle qui se lit «a est-il égal à 3 ?»
Alors que a= 3 ; est une instruction d'affectation qui se lit «j'affecte 3 à a»
Opérateurs logiques : non **!** et **&&** ou **||**

Principales instructions

Commentaire */* **<***texte_du_commentaire***>** ***/***

Itérations

Avec compteur

for (<*expr_initc_ptr*> **;** <*expr_cond.repet_cptr*> **;** <*expr_cptr*>**)** {*bloc*}
on utilise un compteur, interdit de modifier le compteur dans le bloc
pas de exit goto ou branchement quelconque
Exemple : afficher tous les caractères minuscules

```
for (i='a' ; i<='z' ; i++) printf("%c ", i) ;
printf("\n") ; /* afficher toutes les lettres de l'alphabet */
```

Avec condition d'arrêt

Exprimer la condition d'arrêt : puis la traduire en condition pour continuer en respectant les lois de De Morgan

Syntaxe : **while (**<*condition*>**)** {<*bloc* >}

attention : aux initialisations avant l'itération, à bien modifier la valeur de la condition dans le bloc d'instruction sous peine d'avoir une itération infinie

Exemple

```
i= 'a' ;
while (i<='z') { printf("%c ", i);
                 i++ ;}
printf("\n") ;
```

Syntaxe :

```
do {<bloc>
   }
while ( <condition> ) ;
```

Conditionnelles

Simple **if (**<*condition*>**)** <*instr*> ;

Complète **if (**<*condition*>**)** <*instr1*> ; **else** <*instr2*> ;

Avec des blocs d'instructions **if (**<*condition*>**)** {<*bloc1*>}
else {<*bloc2*>}

Aiguillage

```
switch (<var_int> ) {
        case <valeur1> : <instr1> ; <instr2> ;break ;
        case <valeur2> : … ;
        default : <instr> ;
        }
```

Entrées- sorties

Saisie de variable : **scanf("<***format_de_saisie***>", &<***var***>) ;**
Affichage : **printf("<***message***>**\n **")** ou **printf("<***message format***>**\n**",** <*variable*>**) ;**
Saisie de caractère : **getc**(<*caractere*>) ;
Affichage de caractère **putc** (<*caractere*>) ;
Saisie de chaîne de caractères : **gets**(<*chaine*>) ;
Affichage de chaînes de caractères : **puts**(<*chaîne*>) ;

Principales fonctions et librairies

<stdio.h> gestion des entrées sorties (scanf, printf, getc, putc, gets, puts…)
<math.h> fonctions mathématiques (pow, sqrt, fabs, mod, log, sin…)
<string.h> traitement des chaînes de caractères (strcpy, strlen, strcmp…)
<stdlib.h> utilitaires (rand, malloc, free…)
<ctype.h> tests de classes des caractères (isspace, isupper, alpha…)
<limits.h> et <float.h> les limites définies par l'implémentation

Mots clés réservés

auto[3]	double[1]	int[1]	struct[1]
break[2]	else[2]	long[1]	switch[2]
case[2]	enum[1]	register[3]	typedef[3]
char[1]	extern[3]	return[2]	union[1]
const	float[1]	short[1]	unsigned[1]
continue[2]	for[2]	signed[1]	void[1]
default[2]	goto[2]	sizeof[2]	volatile
do[2]	if[2]	static[3]	while[2]

[1] Typage [2] Instruction [3] Indicateurs de classe de mémorisation

Procédures

Fonction

Syntaxe de profil

```
/* role de la procédure : … */
/* données : liste des paramètres avec leur type */
/* résultat : type du résultat */
<type_result> <nom_fonction> ( <type_param> <nom_param_formel>)
{       /* variables locales */
        /* instructions */
        return <var_de_type_result> ;
}
```

avec type_result de type simple

Procédure

Syntaxe de profil

```
/* role de la procédure : … */
/* données : liste des paramètres avec leur type */
/* résultat via : pointeur sur le type du résultat */
void <nom_procedure> ( <type_param> <nom_param_formel> )
{       /* variables locales */
        /* instructions */
}
```

Compilation

Pour compiler le programme source essai.c en l'objet exécutable, taper dans une fenêtre console :

```
gcc -o essai essai.c
```

si le programme contient des fonctions mathématiques, taper :

```
gcc -lm -o essai essai.c
```

pour garantir une bonne portabilité de votre code, taper :

```
gcc -ansi -Wall -o essai essai.c
```

et s'assurer qu'il n'y a plus aucun de message d'erreur ou de prévention.