Complément préprocesseur : Les macros standard prédéfinies

Il existe 8 macros prédéfinies en C

Leur nom commence par deux caractères de soulignement :

Macro	Texte de substitution
__LINE__	**Le numéro de ligne (à l'intérieur du fichier source donné) où la macro __LINE__ apparait**
__FILE__	**Le nom du fichier source dans lequel la macro __FILE__ apparait**
__func__	**Le nom de la fonction ans laquelle la macro __func__ apparait**
__DATE__	**La date de la compilation, dans le format « Mois Jour Année » Exemple : Dec 18 2002**
__TIME__	**L'heure de compiilation, dans le format « hh:mm:ss »**
__STDC__	**La constante entière 1 si le compilateur est conforma au standard ANSI**
__STD_HOSTED__	**La constante entière 1 si l'implémentatin courante est de type hébergée ; sinon 0**
__STD_VERSION__	**La constant entière 199901L si l'implémentation est conforma au C99, le standard C ANSI de janvier 1999**

Exemple :

```
static long ImplSalFrameCallbackDummy( void*, SalFrame*, USHORT, const void* )
{
    fprintf(stderr, ">*>_> %s\n",__func__);
      return 0;
}
```

Rappel :

Spécificateurs de classes de mémorisation

static : la variable n'est visible que depuis la fonction qui la contient (portée limitée)

Au contraire :

extern : portée globale, visible et utilisable partout

Autres types très utilisés

Type	Finalité	Fichier d'en tête
size_t	**Employé pour exprimer la taille d'un objet en nombre d'octets (équivalent en général à un *unsigned int*)**	**stddef.h stdio.h**
wchar_t	**Employé pour les caractères codés sur plusieurs octets, et suffisamment grand pour représenter les valeurs codées de tous les jeux de caractères étendus**	**stdlib.h wchar.h (défini avec C99)**
win_t (défini avec C99)	**Un type entier utilisé pour représenter les caractères étendus, y compris la macro WEOF**	**wchar.h (défini avec C99)**
ptrdiff_t	**Employé pour représenter la différence de deux pointeurs (équivalent habituellement à int)**	**stddef.h**

Les types dérivés (appelés aussi types complexes)

Ils comprennent :

- les pointeurs (hors programme)

- les tableaux :

déjà vus : un espace de stockage (volatile) dans lequel on range des données identiques, toutes de même type.

Illustration possible : une boite contenant n cases de même dimensions,
et ne pouvant contenir que des objets de même nature

+

– les structures : un espace de stockage (volatile) dans lequel on range des données de types différents (pas obligé)

Illustration possible : une boite comportant n compartiments de dimensions pouvant être égales ou différentes, et pouvant contenir des objets de natures différentes

– les unions : un espace de stockage pour un seul objet à un instant donné, mais dont la taille est liée au type ayant la plus grand longueur

Illustration (trouvez mieux): une consigne, dans lequel on peut mettre un seul sac à la fois, mais la taille de ce sac peut varier

Types énumérés -> mot clé **enum**

Ce type sert à définir des variables qui ne peuvent prendre,
au cours du déroulement du programme,
que certaines valeurs entières et discrètes

Les valeurs et le nom ce ces variables sont définis dans une énumération.

Le spécificateur de type commence par le mot clé enum.

Par défaut, le premier membre, sauf valeur explicite, représente la valeur 0.

Trois étapes :

Déclaration (dans un ***.h*** normalement)

/* dans le .c */
Initialisation

Utilisation

Exemple :

Définition

```
enum Code_couleurs {
noir = 0,
marron = 1,
rouge = 2,
orange = 3,
jaune = 4,
vert = 5,
bleu =6,
violet = 7,
gris= 8,
blanc =9
};
```

Déclaration (on crée une variable anneau1 de type enum):

```
enum Code_couleurs  anneau1;
```

On crée une variable, et on lui attribue une valeur. Exemple :

```
/* Utilisation */
anneau1 = rouge;
```

Signifie que l'on attribue la valeur 2 à la variable anneau1.

Équivalent possible :

#define ROUGE 2

suivi de

anneau1 = ROUGE ;

Autre exemple (on verra plus tard pour l'utilisation)

Theme Menu Types
Specify a type of menu.

```
enum {
   kThemeMenuTypePullDown = 0,
   kThemeMenuTypePopUp = 1,
   kThemeMenuTypeHierarchical = 2,
   kThemeMenuTypeInactive = 0x0100
};
typedef UInt16 ThemeMenuType;
```

Constants
kThemeMenuTypePullDown
A pull-down menu.

Available with Appearance Manager 1.0.1 and later.

kThemeMenuTypePopUp
A pop-up menu. Available with Appearance Manager 1.0.1 and later.

kThemeMenuTypeHierarchical
A hierarchical menu. Available with Appearance Manager 1.0.1 and later.

KthemeMenuTypeInactive
An inactive menu. Add this value to any other menu type if the entire menu is inactive.
Available with Appearance Manager 1.1 and later.

Discussion
You can pass constants of type ThemeMenuType in the inMenuType parameter of GetThemeMenuBackgroundRegion and DrawThemeMenuBackground.

Le type structure, noté " **struct** "

Syntaxe :

```
struct nom_structure {
        type    variable1 ;
        type    variable2 ;
        type    variable3;
};
```

Vocabulaire :

Une variable déclarée de type structure est appellée **enregistrement**

Cette structure comporte 3 **membres** (ou **champs**)

nom_structure est l'**étiquette** de la structure

Autres syntaxes de déclaration possibles :

```
struct article {
        char    nom[40];
        int     quantite;
        double prix;
};
```

article est l'étiquette (tag) de la structure

nom, quantité et prix sont les champs de cette structure

Portée de la déclaration :

```
struct article a1, a2 , *pArticle;

struct article tabArticle[100];
```

- a1 et a2 sont de type ***struct article***
- pArticle est un pointeur vers un objet de type ***struct article***
- le tableau tabArticle est composé de 100 éléments de type ***struct article***

Dernier cas, dans lequel on ne fait plus référence, plus tard, à un type structure.
Alors la déclaration peut ne pas comporter d'étiquette :

```
struct {unsigned char caract, attrbt;} xchar, xchn[100];
```

le type structure a deux membres : caract et attrbt

xchar et les éléments du tableau xchn sont du type de la structure sans étiquette.

```
struct HIThemeButtonDrawInfo {

  /*
   * The version of this data structure.  Currently, it is always 0.
   */
  UInt32              version;

  /*
   * The ThemeDrawState of the button being drawn or measured.
   */
  ThemeDrawState      state;

  /*
   * A ThemeButtonKind indicating the type of button to be drawn.
   */
  ThemeButtonKind     kind;

  /*
   * The ThemeButtonValue of the button being drawn or measured.
   */
  ThemeButtonValue    value;

  /*
   * The ThemeButtonAdornment(s) with which the button is being drawn
   * or measured.
   */
  ThemeButtonAdornment  adornment;
  union {
    HIThemeAnimationTimeInfo  time;
    HIThemeAnimationFrameInfo  frame;
  }                   animation;
};

typedef struct HIThemeButtonDrawInfo    HIThemeButtonDrawInfo;
typedef HIThemeButtonDrawInfo *         HIThemeButtonDrawInfoPtr;
```

Occupation mémoire d'une structure

Les champs d'une variable de type structure
sont rangés en mémoire dans l'ordre de leurs déclarations

L'adresse du premier membre est celle de la structure elle-même

ATTENTION : les adresses des autres champs et l'espace mémoire total nécessaire
peut varier d'une implémentation à l'autre car le compilateur optimise l'occupation
en insérant des espaces de longueur variable entre les différents champs (" padding ").

=> **toujours utiliser sizeof() pour obtenir la taille en mémoire d'une structure**

Enfin, la macro offsetof(type_structure, membre), définie dans ***stddef.h***
est du type ***size_t*** et retourne
le nombre d'octets entre l'adresse de début de la structure et "membre"

Comment accéder au membre(s) d'une structure ? (c'est à dire lire l'enregistrement)

Opérateur point **.**

Exemple :

/* Définition et initialisation */

```
struct article
{
   char nom[40];
   int quantite;
   float prix ;
};
```

struct article fleur ;

/* utilisation */

fleur**.**nom // désigne le tableau 'nom'

fleur**.**prix // désigne la variable de type réel prix

Remarques (hors programme) :

l'opérateur flèche (->) permet d'accéder facilement au membre d'une structure désigné par un pointeur

```
pArticle = &fleur;   // pArticle pointe sur fleur
pArticle->quantite; // Accès aux membres de fleur
pArticle-> prix;
```

Une structure ne peut pas être membre d'elle même

mais on peut définir des structures récursives

au moyen de membres qui sont des pointeurs vers le type de la structure elle-même

-> listes chainées , arbres binaires (LO21)

Unions

Une union permet d'utiliser un même emplacement mémoire pour stocker des variables de types différents.

Déclaration :

union nombre { long n ; double x};

Cette déclaration crée un nouveau type ***union*** avec l'étiquette nombre et les membres x et n.

Contrairement aux membres d'une structure, les membres d'une union commencent tous à la même adresse

La taille d'une union est la taille du plus grand de ses membres

Définition d'une variable de type union :

union nombre nx[10] /* déclare un tableau de 10 éléments de type union nombre. */

À n'importe quel moment, chaque élément du tableau peut contenir une des deux valeurs exclusivement.

On accède aux membres d'une union comme on accède aux membres d'une structure.

Exemple :

nx[0].x = 1.234 // affecte une valeur de type double à nx[0].

Définition d'un nom de type avec *typedef*

Le mot clé typedef permet de donner un nouveau nom à un type existant

Exemples :

```
typedef  unsigned char  UCHAR ;
```

```
typedef  struct { double x, y; } POINT;
```

-> l'identificateur est déclaré comme synonyme du nom du type.
Sans la présence de typedef, on déclarerait une variable, et pas un nom de type

UCHAR est utilisé comme abbréviation pour unsigned char et POINT pour une structure de type défini plus haut.

UCHAR c1 ;

UCHAR c2 ;

UCHAR tab[100] ;

POINT point , *pPoint ;

Exemples importants :

/* definition d'une enumeration pour les booleens */

enum Boolean { FALSE, TRUE };

/* definition d'un nouveau type our les booleens */

typedef enum Boolean BOOL;

BOOL remplacera enum Boolean , avec Boolean qui designe une énumération.

/* Declaration de variables */

enum Boolean var1 = TRUE;
BOOL var2 = FALSE;