CM 10 - LES APPORTS DES NORMES C90, C99 ET C11

G. Bianchi, G. Blin, A. Bugeau, S. Gueorguieva, R. Uricaru

2015-2016

Programmation 1 - uf-info.ue.prog1@diff.u-bordeaux.fr

université de BORDEAUX

GÉNÉRALITÉS

▷ La norme ANSI (C90) fournit à la fois la description du langage C et le contenu d'une bibliothèque standard

▷ Les extensions publiées par ISO en 1999 (C99) et 2011 (C11) ne sont pas implémentées entièrement par tous les compilateurs

▷ Compiler suivant un standard s'effectue avec l'option `-std=cXX` en remplaçant `XX` par 99 ou 11

▷ Le standard utilisé est symbolisé par `__STDC_VERSION__`

  ▷ Pas de valeur pour C90.

  ▷ `199901L` pour C99.

  ▷ `201112L` pour C11.

▷ Listing exhaustif dans les anenxes A et B de "Le guide complet du langage C" de Claude Delannoy

c90

Les sous-bibliothèques

▷ A chaque sous-bibliothèque est associé un fichier en-tête comportant essentiellement :

- ▷ Les prototypes des fonctions correspondantes
- ▷ Les définitions des macros correspondantes
- ▷ Les définitions de certains symboles utiles au bon fonctionnement des fonctions ou macros de la sous-bibliothèque (des types ou de synonymes définis par `typedef` ou des constantes symboliques)

Les sous-bibliothèques

▷ A chaque sous-bibliothèque est associé un fichier en-tête

▷ Les différents fichiers en-tête prévus par la norme

En-tête	Description
<assert.h>	Macros de mise au point
<ctype.h>	Test de catégories de caractères et conversions majuscules/minuscules
<errno.h>	Gestion des erreurs
<float.h>	Caractéristiques des types flottants
<limits.h>	Caractéristiques des types entiers (et caractère)
<locale.h>	Caractéristiques locales
<math.h>	Fonctions mathématiques
<setjmp.h>	Branchements non locaux
<signal.h>	Traitement de signaux

Les sous-bibliothèques

- ▷ A chaque sous-bibliothèque est associé un fichier en-tête
- ▷ Les différents fichiers en-tête prévus par la norme

En-tête	Description
<stdarg.h>	Gestion d'arguments variables
<stddef.h>	Définitions communes
<stdio.h>	Entrées-sorties
<stdlib.h>	Utilitaires : conversions de chaînes, nombres aléatoires, gestion de la mémoire, communication avec l'environnement, arithmétique entière, caractères étendus, chaînes de caractères étendus
<string.h>	Manipulation de chaînes et de suites d'octets
<time.h>	Gestion de l'heure et de la date

ASSERT.H

▷ Le fichier `assert.h` fournit une fonction de mise au point désactivable par la définition du symbole `NDEBUG`

```
void assert(int exptest)
```

▷ Si `exptest` est évaluée à `0`, le programme sera interrompu avec l'affichage d'un message d'erreur (sur stderr) de la forme :

```
Assertion failed : exptest, filename, line xxxx
```

▷ On y trouve :

- ▷ L'expression concernée : `exptest`
- ▷ Le nom du fichier source concerné : `filename`
- ▷ Le numéro de la ligne correspondante : `xxxx`

▷ Sinon, rien ne se passe

CTYPE.H

▷ Le fichier `Ctype.h` fournit des fonctions de tests de caractères et conversions majuscules - minuscules

▷ Les fonctions de test d'appartenance d'un caractère à une catégorie qui renvoit `0` dans le cas contraire

  ▷ `int isalnum(int c)` fournit une valeur non nulle si `c` est un caractère alphanumérique

  ▷ `int isalpha(int c)` fournit une valeur non nulle si `c` est un caractère alphabétique

CTYPE.H

▷ Le fichier `Ctype.h` fournit des fonctions de tests de caractères et conversions majuscules - minuscules

▷ Les fonctions de test d'appartenance d'un caractère à une catégorie qui renvoit `0` dans le cas contraire

  ▷ `int isdigit(int c)` fournit une valeur non nulle si `c` est un chiffre

  ▷ `int islower(int c)` fournit une valeur non nulle si `c` est une lettre minuscule

  ▷ `int ispunct(int c)` fournit une valeur non nulle si `c` est un caractère de ponctuation

CTYPE.H

▷ Le fichier `Ctype.h` fournit des fonctions de tests de caractères et conversions majuscules - minuscules

▷ Les fonctions de test d'appartenance d'un caractère à une catégorie qui renvoit `0` dans le cas contraire

  ▷ `int isspace(int c)` fournit une valeur non nulle si `c` est un espace blanc (i.e., `"\t "`, `"\v "`, `"\r "` ou `"\n "`)

  ▷ `int isupper(int c)` fournit une valeur non nulle si `c` est une lettre majuscule

CTYPE.H

▷ Le fichier `Ctype.h` fournit des fonctions de tests de caractères et conversions majuscules - minuscules

▷ Les fonctions de transformation de caractères

▷ `int tolower(int c)` fournit, si `isupper(c)` est vrai, le caractère correspondant de la catégorie minuscules, s'il existe ; `c` sinon

▷ `int toupper(int c)` fournit, si `islower(c)` est vrai, le caractère correspondant de la catégorie majuscules, s'il existe ; `c` sinon

MATH.H

▷ Le fichier `math.h` fournit des fonctions mathématiques

▷ Les fonctions trigonométriques

  ▷ `double cos(double x)` fournit la valeur de `cos(x)`, la valeur de `x` étant exprimée en radians

  ▷ `double sin(double x)` fournit la valeur de `sin(x)`, la valeur de `x` étant exprimée en radians

  ▷ `double tan(double x)` fournit la valeur de `tan(x)`, la valeur de `x` étant exprimée en radians

MATH.H

- Le fichier `math.h` fournit des fonctions mathématiques
- Les fonctions exponentielle et logarithme
  - `double exp(double x)` fournit la valeur de `exp(x)`
  - `double log(double x)` fournit la valeur de `ln(x)`, log arithme népérien de `x` (la bijection réciproque de la fonction exponentielle)
  - `double log10(double x)` fournit la valeur de `log(x)`, la valeur du logarithme à base 10 de `x`

MATH.H

▷ Le fichier `math.h` fournit des fonctions mathématiques

▷ Les fonctions puissance

  ▷ `double pow(double x, double y)` fournit la valeur de x^y

  ▷ `double sqrt(double x)` fournit la valeur de la racine carrée arithmétique (valeur non négative) de `x`

MATH.H

▷ Le fichier `math.h` fournit des fonctions mathématiques

▷ D'autres fonctions utiles

▷ `double ceil(double x)` fournit (sous la forme d'un double) le plus petit entier qui ne soit pas inférieur à `x`

▷ `double fabs(double x)` fournit (sous la forme d'un double) la valeur absolue de `x`

▷ `double floor(double x)` fournit (sous la forme d'un double) le plus grand entier qui ne soit pas supérieur à `x`

STDLIB.H

▷ Le fichier `stdlib.h` fournit des fonctions utilitaires

▷ Des fonctions de conversion de chaîne (⚠ la norme ne précise pas quel doit être le comportement de ces fonctions en cas d'erreur)

  ▷ `double atof(const char* str)` fournit le résultat de la conversion en `double` du début de la chaîne d'adresse `str`

  ▷ `int atoi(const char* str)` fournit le résultat de la conversion en `int` du début de la chaîne d'adresse `str`

  ▷ `long atol (const char* str)` fournit le résultat de la conversion en `long` du début de la chaîne d'adresse `str`

STDLIB.H

▷ Le fichier `stdlib.h` fournit des fonctions utilitaires

▷ Des fonctions de génération de séquences de nombres pseudo aléatoires

  ▷ `int rand(void)` fournit un nombre entier pseudo-aléatoire, compris dans l'intervalle `[0, RAND_MAX]`

  ▷ `void srand(unsigned int seed)` modifie la "graine" utilisée par le générateur de nombres pseudo aléatoires (pour une valeur de graine donnée, la suite des valeurs obtenues par des appels successifs à `rand` est toujours la même)

STRING.H

▷ Le fichier `string.h` fournit des fonctions manipulations de suites de caractères

▷ Des fonctions de copie (⚠ si les deux chaînes ont des parties communes, le comportement du programme est indéterminé)

▷ `char* strcpy(char* dest, const char* src)` copie la chaîne d'adresse `src` à l'emplacement d'adresse `dest` (y compris le `"\0"` de fin) et retourne l'adresse `dest`

▷ `char* strncpy (char* dest, const char* src, size_t n)` copie au maximum `n` caractères de la chaîne d'adresse `src` à l'emplacement d'adresse `dest` en complétant éventuellement par des `"\0"` et retourne l'adresse `dest`

STRING.H

▷ Le fichier `string.h` fournit des fonctions manipulations de suites de caractères

▷ Des fonctions de concaténation (⚠ si les deux chaînes ont des parties communes, le comportement du programme est indéterminé)

▷ `char* strcat (char* dest, const char* src)` recopie la chaîne d'adresse `src` (y compris le `"\0"` de fin) à la fin de la chaine d'adresse `dest` et retourne l'adresse `dest`

▷ `char* strncat (char* dest, const char* src, size_t n)` copie au maximum `n` caractères de la chaîne d'adresse `src` à l'emplacement d'adresse `dest` en complétant éventuellement par des `"\0"` et retourne l'adresse `dest`

STRING.H

▷ Le fichier `string.h` fournit des fonctions manipulations de suites de caractères

▷ Des fonctions de comparaison

  ▷ `int strcmp(const char* str1, const char* str2)` compare lexicographiquement les chaînes situées aux adresses `str1` et `str2` et fournit une valeur négative (resp. positive et égale à zéro) si la chaîne d'adresse `str1` < (resp. > ou =) chaîne d'adresse `str2`

  ▷ `int strncmp(const char* str1, const char* str2, size_t n)` foncionne comme `strcmp`, en limitant la comparaison à un maximum de `n` caractères pour chacune des deux chaînes (les caractères au-delà du `"\0"` n'étant jamais comparés)

STRING.H

▷ Le fichier `string.h` fournit des fonctions manipulations de suites de caractères

▷ Des fonctions de recherche

▷ `char* strchr(const char* str, int c)` fournit un pointeur sur la première occurrence du caractère résultant de la conversion de `c` en `char`, dans la chaîne d'adresse `str`, ou un pointeur `NULL` si ce caractère n'y figure pas

▷ `char* strrchr(const char* str, int c)` fournit un pointeur sur la dernière occurrence du caractère résultant de la conversion de `c` en `char`, dans la chaîne d'adresse `str`, ou un pointeur `NULL` si ce caractère n'y figure pas

STRING.H

▷ Le fichier `string.h` fournit des fonctions manipulations de suites de caractères

▷ Des fonctions de recherche

▷ `size_t strspn(const char* str1, const char* str2)` fournit la longueur du "segment initial" de la chaîne d'adresse `str1` formé entièrement de caractères appartenant à la chaîne d'adresse `str2`

▷ `char* strstr(const char* str1, const char* str2)` recherche la première occurrence, si elle existe, dans la chaîne d'adresse `str1` de la chaîne d'adresse `str2` (le `"\0"` ne participe pas à la recherche) et fournit l'adresse de la chaîne ainsi localisée si elle existe ou un pointeur `NULL` sinon (si `str2` est une chaîne vide, la fonction fournit l'adresse `str1`)

STRING.H

▷ Le fichier `string.h` fournit des fonctions manipulations de suites de caractères

▷ Des fonctions de recherche

  ▷ `char* strtok(char* str, const char* delim)` permet "d'éclater" la chaîne d'adresse `str` à l'aide des délimiteurs contenus dans la chaîne `delim`

```
int main(void)
{
   char str[50] = "bob l'eponge, bob@eponge.fr";
   const char s[2] = ", ";
   char* token = strtok(str, s);
   while( token != NULL ){
      printf( "%s\n", token);
      token = strtok(NULL, s);
   }
   return EXIT_SUCCESS;
}
```

STRING.H

▷ Le fichier `string.h` fournit des fonctions manipulations de suites de caractères

▷ Une fonction utile

  ▷ `size_t strlen(const char* str)` fournit la longueur de la chaîne d'adresse chaine où `size_t` est un type représentant un entier non signé

c99

CONTRAINTES SUPPLÉMENTAIRES

▷ La norme C99 supprime quelques tolérances subsistant dans le C90

▷ En C99, le type de retour d'une fonction doit toujours être mentionné, alors qu'en C90, l'absence de type est interprétée comme int

▷ En C99, toute fonction utilisée dans un fichier source et non définie préalablement dans ce même fihier, doit être déclarée

▷ En C90, il est possible d'introduire une instruction `return` sans expression, alors que la fonction doit fournir un résultat. Cette tolérance n'est plus admise en C99

DIVISION D'ENTIERS

▷ En C90, il subsiste une ambiguïté pour les opérateurs `/` et `%` lorsqu'au moins l'un de leurs opérandes est négatif

▷ La norme C99 impose que la troncature du quotient ait toujours lieu vers zéro

- ▷ `11/3` ($\sim 3,666$) vaut `3`, en C99 comme en C90
- ▷ `-11/3` vaut `-3` en C99, `-3` ou `-4` en C90
- ▷ `11/-3` vaut `-3` en C99, `-3` ou `-4` en C90
- ▷ `-11/-3` vaut `3` en C99, `3` ou `4` en C90
- ▷ `11%3` vaut `2` en C99 comme en C90
- ▷ `-11%3` vaut `-2` en C99, `-2` ou `1` ($+(-4 * 3)$) en C90
- ▷ `11%-3` vaut `2` en C99, `2` ou `-1` ($+(-4 * -3)$) en C90
- ▷ `-11%-3` vaut `-2` en C99, `-2` ou `1` ($+(-4 * -3)$) en C90

EMPLACEMENT DES DÉCLARATIONS

▷ En C90, les déclarations doivent figurer au début d'un bloc, avant les premières instructions exécutables

▷ Cette contrainte n'existe plus en C99 où une variable doit simplement être déclarée avant d'être utilisée comme dans cet exemple : `{ int n ; n =.... int p=12, q ; q = n + 2*p ;`

▷ C99 permet de déclarer une ou plusieurs variables dans la partie initialisation d'une boucle `for`, au sein même de la boucle, ce qui permet de libérer l'espace correspondant dès la sortie de la boucle : `for (int i=0, j=3 ; i<12 ; i++) { ... }`

▷ Les emplacements de `i` et de `j` seront libérés en sortie de boucle

▷ ⚠Il n'est pas possible d'initialiser deux variables de types différents : `for (int i=0, double x=0. ; ... ) { ... }`

TABLEAUX À TAILLE VARIABLE OU VARIABLE LENGTH ARRAY

▷ En C90, la dimension d'un tableau ne pouvait être qu'une expression constante

▷ En C99, la taille d'un tableau peut être définie par une variable

  ▷ Dans les déclarations

```
void foo(int n, int q){
    int t[2*n+q];
    ...
}
```

  ▷ Dans les prototypes et déclarations de fonctions (à condition que la taille soit définie avant le tableau)

```
int foo(int n, int array[n]); // valide
int bar(int array[n], int n); // invalide
```

NOUVELLES FONCTIONS MATHÉMATIQUES

▷ La norme C99 généralise toutes les fonctions mathématiques de nom `xxxx` (concernant le type `double`) aux deux autres types flottants : `xxxxf` pour le type `float` et de `xxxxl` pour le type `long double`

▷ Par exemple, la fonction `atan` (type `double`) est complétée par `atanf` (type `float`) et `atanl` (type `long double`)

▷ Les principales nouvelles fonctions liées aux logarithmes

▷ `exp2f`, `exp2` et `exp2l` : puissance réelle de 2

▷ `log2f`, `log2` et `log2l` : logarithme à base 2

▷ La norme C99 recommande (sans l'imposer) l'utilisation de la norme IEEE-754 pour la représentation des nombres flottant

LE MOT-CLEF INLINE

▷ En ajoutant `inline` devant la déclaration d'une fonction, on demande au compilateur de remplacer l'appel de la fonction par son corps, exactement comme les macros, mais tout en gardant les avantages des fonctions

```
inline double square(double x) {
    return x*x;
}
int f(double x){
    double var = square(x); // will be replaced by double var=x*x;
    ...
}
```

▷ Une fonction `inline` doit être définie dans un fichier d'entête

▷ Rien ne force le compilateur à remplacer l'appel de la fonction par son corps

Le mot-clef restrict

▷ Lorsque l'on est certain qu'un pointeur donné est le seul à permettre l'accès à un emplacement mémoire, il est possible de le préciser au compilateur à l'aide du mot `restrict`

▷ Cela peut lui permettre d'effectuer certaines optimisations

▷ Rien ne force le compilateur à en tenir compte

C11

Gestion des contraintes d'alignement

▷ C11 fournit le moyen d'imposer à une variable une contrainte d'alignement plus restrictive que celle que lui attribuerait l'environnement

```
char _Alignas (long) c;
```

▷ Imposera à la variable `c` d'être alignée sur une frontière correspondant à un `long`

FONCTIONS VÉRIFIANT LE DÉBORDEMENT MÉMOIRE

▷ La norme C11 propose (de façon facultative) un ensemble de fonctions « de substitution » créées initialement par Microsoft pour lutter contre des problèmes de sécurité connus

▷ Chacune de ces fonctions porte le nom de la fonction substituée, suffixé par `_s`

  ▷ `gets_s`, `strcpy_s`, `strcat_s`, `memcpy_s` et `memmove_s` qui permettent de limiter le nombre de caractères introduits

  ▷ `strncpy_s` et `strncat_s`, qui assurent la présence d'un `"\0"`

  ▷ `strlen_s` qui permet de fixer une longueur maximale de la zone considérée (pour éviter des erreurs d'adressage)

▷ Lorsque ces fonctions sont implémentées, la variable `__STDC_LIB_EXT1` est définie et vaut `1`

DOGGY BAG

TO TAKE AWAY ...

▷ Un grand nombre de fonctionnalités sont déjà présentes dans la librairie standard

▷ Beaucoup d'entre-elles demandent au programmeur de respecter des contraintes et conditions sous peine de comportement indéterminé

▷ Une partie non négligeable des normes n'est pas forcément implémentée par les différents compilateurs

Questions?