Le langage C

70's Début du langage C (Ritchie et Kernighan),

Origine liée au système UNIX (90% écrit en C)

1978 ``*The C Programming Language*``, (Kernighan et Ritchie),

--> on fait référence à cette première version en parlant du ``C K& R"

1983 Comité de normalisation ANSI

1989 Norme ``ANSI C``,

--> on fait référence à cette seconde version en parlant du ``C ANSI``

Utilisation du C

Développé en 1971 par Dennis Ritchie au sein des laboratoires Bell.

Langage d'implantation de systèmes d'exploitation (Unix et ses dérivées):

--> instructions proches de la machine ;

--> pointeurs typés mais non contrôlés ;

--> portabilité.

Langage adapté à l'écriture de petits et de moyens programmes:

--> efficacité du code généré ;

--> compilation séparée ;

--> nombreux outils disponibles.

Langage peu utilisé pour les grosses applications:

--> approche archaïque de la modularité ;

--> typage laxiste.

Pleine richesse sous Unix.

Le langage C++

80's Début du langage C++ (Stroustrup, ATT Lab)

⇒ Ajout des facilités de *Simula* au langage C

1979 Classes dans le langage C

1983 Premières utilisations de C++

1985 *The C++ Programming Language*, (B. Stroustrup)

1985-1994 Différentes versions de C++

Borland C++, Gnu C++

Premiers pas vers un standard ANSI/ISO:

1986 C++ version 2.0 (héritage multiple)

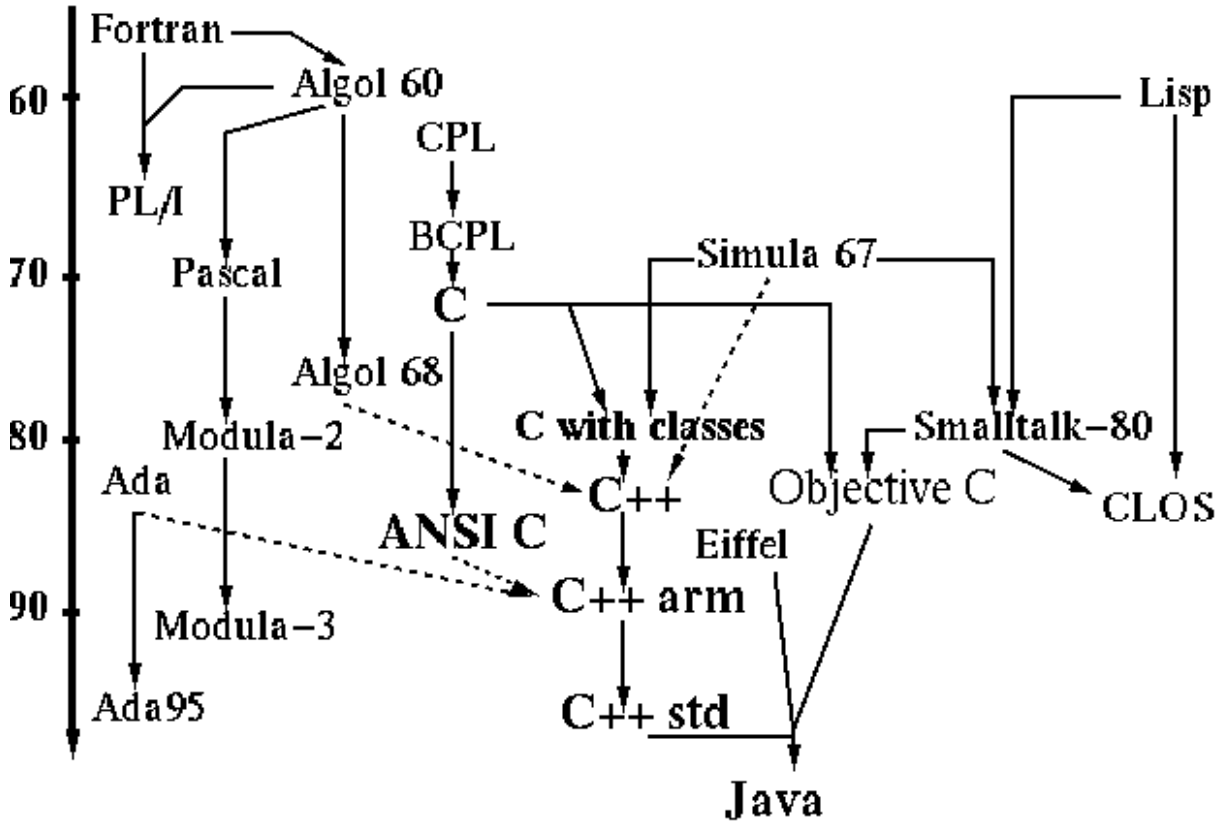
1989 Norme C ISO/ANSI

1991 C++ version 3.0 (template)

1994 C++ version 4.0 (exceptions)

1999 Norme C++ ISO/ANSI

Arbre généalogique de C++



C++ et les autres langages (1)

Présence de la plupart des entités manipulées dans un langage applicatif:

- *types construits:* tableaux, structures, unions, énumérations,
- *pointeurs*,
- *instructions de contrôle*: conditionnelle, répétition, choix.

Ajout des concepts de la programmation par objets:

- encapsulation des données: classes,
- héritage, généricité, polymorphisme.

Ajouts (en plus des classes) par rapport au langage C:

- primitives de gestion de la mémoire dynamique,
- meilleure vérification de types,
- gestion des erreurs : exceptions.

Absence:

- d'*opérateurs* sur les chaînes,
- de *méthodes* d'accès aux fichiers,
- de *notion* de processus

C++ et les autres langages (2)

Mais ...

L'environnement de programmation de C++ permet de résoudre tout ou partie de ces problèmes. Ces mécanismes sont fournis par:

- des fonctions regroupées dans des bibliothèques *standard* (cf. Manuel 3 UNIX, Help Borland, Help Visual C++),
- les primitives du système d'exploitation sont utilisables *directement* par des appels de fonctions du système (cf. Manuel 2 UNIX, Help Borland, Help Visual C++),
- de nombreuses bibliothèques disponibles (ex : tools.h++ sous Unix, OWL, OLE sur PC, MFC sur PC et MAC).

Inconvénients de C++

- syntaxe rébarbative,
- peu lisible pour une personne non *familiarisée* au langage.

Caractéristiques du langage C (1)

- **Langage structuré**, conçu pour traiter les tâches d'un programme en les mettant dans des blocs.

- Il produit des **programmes efficaces**: il possède les mêmes possibilités de contrôle de la machine que l'assembleur et il génère un **code compact et rapide**.

- **Déclaratif**: normalement, tout objet C doit être déclaré avant d'être utilisé. S'il ne l'est pas, il est considéré comme étant du type entier.

- **Format libre**: la mise en page des divers composants d'un programme est totalement libre. Cette possibilité doit être exploitée pour rendre les programmes lisibles.

Caractéristiques du langage C (2)

- **Modulaire**: une application pourra être découpée en modules qui pourront être compilés séparément. Un ensemble de programmes déjà opérationnels pourra être réuni dans une librairie. Cette aptitude permet au langage C de se développer de lui même.

- **Souple**: peu de vérifications et d'interdits, hormis la syntaxe. Il est important de remarquer que la tentation est grande d'utiliser cette caractéristique pour écrire le plus souvent des atrocités.

- **Transportable**: les entrées/sorties sont réunies dans une librairie externe au langage.

Exemple de programme en C

```
#include <stdio.h>  /* directives au préprocesseur */
#define DEBUT -10
#define FIN 10
#define MSG "Programme de démonstration\n"

int carre(int x);   /* déclaration des fonctions */
int cube(int x);

main()                  /* programme principal */
{                       /* début du bloc de la fonction main */
 int i;                 /* définition des variables locales */

 printf(MSG);
 for ( i = DEBUT; i <= FIN ; i++ )
 {
   printf("%d   carré: %d   cube: %d\n", i , carre(i), cube(i) );
 }                      /* fin du bloc for  */
 return 0;
}                       /* fin du bloc de la fonction main */


int cube(int x)         /* définition de la fonction cube */
{
  return x * carre(x);
}

int carre(int x)        /* définition de la fonction carre */
{
  return x * x;
}
```

Règles d'écriture des programmes C

Afin d'écrire des programmes C lisibles, il est important de respecter un certain nombre de règles de présentation:

- ne jamais placer plusieurs instructions sur une même ligne.
- utiliser des identificateurs significatifs.
- grâce à l'indentation des lignes, on fera ressortir la structure syntaxique du programme. Les valeurs de décalage les plus utilisées sont de 2, 4 ou 8 espaces.
- on laissera une ligne blanche entre la dernière ligne des déclarations et la première ligne des instructions.
- une accolade fermante est seule sur une ligne (à l'exception de l'accolade fermante du bloc de la structure **do** ... **while**) et fait référence, par sa position horizontale, au début du bloc qu'elle ferme.
- aérer les lignes de programme en entourant par exemple les opérateurs avec des espaces.
- il est nécessaire de commenter les listings. Eviter les commentaires triviaux.

ETAPES PERMETTANT L'EDITION, LA MISE AU POINT, L'EXECUTION D'UN PROGRAMME

*1- **Edition du programme source***, à l'aide d'un éditeur (traitement de textes). Le nom du fichier contient l'extension .C, exemple: PROG1.C.

*2- **Compilation du programme source,*** c'est à dire création des codes machine destinés au microprocesseur utilisé. Le compilateur indique les erreurs de syntaxe mais ignore les fonctions-bibliothèque appelées par le programme. Le compilateur génère un fichier binaire, appelé fichier objet: PROG1.OBJ.

*3- **Editions de liens*****:** le code machine des fonctions-bibliothèque est chargé, création d'un fichier binaire, appelé fichier exécutable: PROG1.OUT* (UNIX) ou PROG1.EXE (DOS, Windows).

*4- **Exécution du programme*****:** lancement de l'exécutable à partir d'une ligne de commande (UNIX). Lancement de l'exécutable par un double-clique sur son icône (Windows et Mac).

Séparateurs et commentaires

espace, tab, newline: sont ignorés, ils servent uniquement de séparateurs d'entités lexicales.

/* ... */ ou // (C++ uniquement) : commentaires, ignorés également.

```
/* ceci est un commentaire …
                                …sur plusieurs lignes */
// ne commente qu'une ligne
```

Attention: les commentaires ne s'imbriquent pas!

; : termine une déclaration ou une instruction simple,

, : sépare 2 éléments dans une liste,

(...) : liste d'arguments,

{ ... } : bloc ou liste d'initialisations,

[...] : dimension/sélection d'éléments de tableaux.

Identificateurs (1)

Les identificateurs nomment les objets C (fonctions, variables ...):

- C'est une suite de lettres ou de chiffres.
- Le premier caractère est obligatoirement une lettre.
- Le caractère _ (souligné) est considéré comme une lettre.
- Le C distingue les minuscules des majuscules.

  Exemple: canadiens, Canadiens, CanAdiens et CANADIENS sont des identificateurs valides et tous différents.
- La longueur de l'identificateur dépend de l'implémentation. La norme ANSI prévoit qu'au moins les 31 premiers caractères soient significatifs pour le compilateur.
- L'éditeur de liens peut limiter le nombre de caractères significatifs des identificateurs à un nombre plus petit.

Identificateurs (2)

- Un identificateur ne peut pas être un mot réservé du langage:

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

Recommandations:

- Utiliser des identificateurs significatifs.
- Réserver l'usage des identificateurs en majuscules pour le préprocesseur.
- Réserver l'usage des identificateurs commençant par le caractère _ (souligné) à l'usage du compilateur.

Les différents types de variables en C

Type	Bits	Min.	Max.
char	8	-128	127
signed char	8	-128	127
unsigned char	8	0	255
short	16	-32768	32767
unsigned short	16	0	65535
int	32	-2147483648	2147483647
unsigned	32	0	4294967295
long	32	-2147483648	2147483647
unsigned long	32	0	4294967295
float	32	1.17 E-38	3.4 E+38
double	64	2.22 E-308	1.7 E+308
long double	128	3.36 E-4932	1.1 E+4932

Initialisation de variables

Initialisation explicite:

Les variables peuvent être initialisées lors de leur définition.

```
int i , j = 5;   /* initialise la variable j a 5 */
char c = 'R';
char tab1[3] = { 'a' , 'b' , 'c' };
char tab2[ ] = { 'a' , 'b' , 'c' };
```

Le compilateur détermine pour tab2 le nombre d'éléments en fonction du nombre d'initialiseurs.

```
int tab3[3][2] = {{1,2}, {3,4}, {5,6}};
int tab4[3][2] = { 1, 2, 3, 4, 5, 6}; /* pareil que  tab3   */
int tab5[4]    = { 1, 2 };   /* tab5[2] = 0  et  tab5[3]=0  */
```

S'il y a moins d'initialiseurs que d'éléments déclarés, les éléments non initialisés le sont implicitement à 0.

Initialisation avec la valeur d'une expression:

D'une manière générale, une variable peut être initialisée avec la valeur d'une expression.

```
short a = sqrt(x)/10;
int b = 2*a + 3;
int nb = atoi(argv[1]);
```