

Programmation C avancé

Édition, programmation et mise au point dans le cas d'un environnement Unix

Stéphanie EVEN (Stephanie.Even@enstb.org)

Serge GUELTON (Serge.Guelton@enstb.org)

Ronan KERYELL (Ronan.Keryell@hpc-project.com)

High Performance Computing Architecture and Security (HPCAS)

Département Informatique, TÉLÉCOM Bretagne

&
HPC Project

Avril 2009

1.25

Le plan

- 1 Introduction
 - Bibliographie
 - Petite histoire
- 2 Langage
 - Généralités
- 3 Un ordinateur ? Mais qu'est-ce ?
 - Les bases de la numérologie
 - Calcul binaire
- 4 Déclarations
 - Généralités
 - Types des objets
 - Portée
- 5 Expressions
 - Sémantique opérateurs
 - Coercition de type
- 6 Instructions
- 7 Préprocesseur
- 8 Bibliothèques
- 9 Modèle mémoire & allocation
- 10 Entrées-sorties
- 11 GNOME
- 12 Éditeur exemple d'Emacs
 - Tuning de son clavier
- 13 Compilation
- 14 Modularisation
- 15 Mise au point
- 16 Dévermineur (debugger)
- 17 Analyse mémoire
- 18 Extensions
- 19 Du C pour le graphisme
- 20 C++
- 21 Délires
- 22 Le bêtisier
- 23 Concours de programmes incompréhensibles
- 24 Conclusion
- 25 Index
- 26 Table des matières

Domaine d'utilisation

(I)

- Langage à spectre large porté sur toutes les applications
 - ▶ Supercalculateurs parallèles avec $10\ 000 - 10^6$ processeurs
 - ▶ Micro-contrôleurs (proche du matériel)
 - ▶ Systèmes d'exploitation
 - ▶ Synthèse architecturale (SpecC, SystemC)
- De nombreuses bibliothèques pour faire des choses de haut niveau
 - ▶ Algorithmique et structures de données
 - ▶ WWW
 - ▶ Composants métiers

Pourquoi un cours de C ?

(I)

- Utilisé dans de nombreux projets
- Fait partie de l'inconscient collectif informatique
 - ▶ Langage informatique servant de base à de nombreux autres langages
 - ▶ Nombreux langages définis par rapport au C (C++, Java, C#, ObjectiveC, UPC, OpenCL, Cg, CUDA, Brook+, CTM, OpenCL, SystemC, SpecC, HandelC, HyperC, PompC, csh...)
- De nombreux autres domaines de l'ENST Bretagne impliquent des connaissances en C
 - ▶ Traitement du signal
 - ▶ Analyse d'image
 - ▶ Electronique
- Projet coupe de robotique (E=M6) difficile par le passé car peu d'élèves ayant fait du C dans la vraie vie ☺
But subliminal du cours : former l'équipe E=M6 ! ☺
Dédicace spéciale à Céline et son port parallèle ☺



Pourquoi un cours de C ?

(II)

- ⚠ Ce cours présuppose connaissance d'un langage informatique !

De nombreuses extensions

(I)

Un langage vivant

- Extensions orientées objet (C++, Objective C)
- Programmation parallèle (UPC, OpenMP)
- Cg, CUDA (cartes graphiques de nVidia), Brook+, CTM (AMD-ATI), OpenCL
- Extensions ad-hoc ou utilisation de classes ?

... et des restrictions sur du matériel réduit

- Micro-contrôleurs (pas de float, double...)
- 8088–80386 (far, near...)
- DSP



Problématique pédagogique

(I)

- Apprendre progressivement tout un langage et environnement de développement
 - ▶ De nombreuses interdépendances de concepts
 - ▶ Obligé de faire un enseignement en largeur d'abord
- Test un concept nouveau : le parcours pédagogique de l'exposé n'est pas forcément celui des transparents

Effets de bord positifs

- Tient les élèves en haleine car doivent faire des sauts de pages plus complexes
- Muscle plus les doigts



Hétérogénéité des élèves

(I)

- Parcours de plus en plus hétérogènes à cause (grâce) aux réformes de l'enseignement (semestrialisation et optionalité) et la variété des recrutements
- Phénomène aggravé grâce à la pénétration culturelle de l'informatique à la maison
- Comment faire le grand écart entre des débutants et des confirmés en langage C ?
- Gageure : arriver à intéresser les 2 publics ☺
- Plus qu'un cours de C : faire passer une « culture » informatique
- Pas de « petites classes » mais il faut des élèves actifs !



Le plan

- 1 Introduction
 - Bibliographie
 - Petite histoire
- 2 Langage
 - Généralités
- 3 Un ordinateur ? Mais qu'est-ce ?
 - Les bases de la numérologie
 - Calcul binaire
- 4 Déclarations
 - Généralités
 - Types des objets
 - Portée
- 5 Expressions
 - Sémantique opérateurs
 - Coercition de type
- 6 Instructions
- 7 Préprocesseur
- 8 Bibliothèques
- 9 Éditeur exemple d'Emacs
 - Tuning de son clavier
- 10 Compilation
- 11 Modularisation
- 12 Mise au point
 - Dévermineur (debugger)
 - Analyse mémoire
- 13 Extensions
 - Du C pour le graphisme
 - C++
- 14 Délires
 - Le bêtisier
 - Concours de programmes incompréhensibles
- 15 Conclusion
- 16 Index
- 17 Table des matières

Ressources & Bibliographie

(I)

Axiome

Ce support de présentation est « cliquable »

- Livres
 - ▶ « Méthodologie de la programmation en C : norme C 99 - API POSIX », Achille Braquelaire , 4^{ème} édition, Dunod, 2005 Sciences Sup
 - ▶ « Programmation avancée en C (avec exercices et corrigés) », Sébastien Varrette et Nicolas Bernard, février 2007, Hermes Science Publication
http://www-id.imag.fr/~svarrett/cours.html#poly_C
- <http://cslibrary.stanford.edu> Explications sur plein de concepts informatiques, dont un film en pâte à modeler sur les pointeurs ☺

Ressources & Bibliographie

(II)

- <http://www.laas.fr/~matthieu/cours/c-superflu> « Guide superflu de programmation en langage C » Matthieu HERRB, exégèse de quelques chausse-trappes
- http://www-clips.imag.fr/commun/bernard.cassagne/Introduction_ANSI_C.html
- <http://www-inf.int-evry.fr/COURS/CTOUTMOD> cours interactif sur C et Emacs
- <http://picolibre.int-evry.fr/projects/coursC> Cours plus complet
- Documentation info sur GNU, GCC, Emacs,... : taper C-h i dans Emacs
- La même chose en ligne sur <http://www.gnu.org/manual/>
 - ▶ <http://gcc.gnu.org/onlinedocs> compilateurs
 - ▶ <http://www.gnu.org/software/libc/manual> bibliothèque standard du C

Ressources & Bibliographie (III)

- ▶ <http://www.gnu.org/software/make/manual> gestion de projet Make
- ▶ <http://www.gnu.org/software/gdb/documentation> débogueur
- ▶ <http://www.gnu.org/software/ddd/manual> débogueur graphique
- ▶ <http://www.gnu.org/software/emacs/manual>
- [http://en.wikipedia.org/wiki/C_\(programming_language\)](http://en.wikipedia.org/wiki/C_(programming_language))
- <http://en.wikipedia.org/wiki/C99>
- FAQ (foires aux questions)
- Ingénierie par moteur de recherche : envoyer messages d'erreur dans le moteur ☺
- <http://www.open-std.org/JTC1/SC22/WG14>
<http://www.open-std.org/JTC1/SC22/WG14/www/docs/n1256.pdf>
Use the source, Luke... en 552 pages

Monitorat

| ∃ Le monitorat à l'ENST Bretagne le soir !

| En monitorat tu iras ! ☺

| ∃ Le monitorat à l'ENST Bretagne le soir !

| En monitorat tu iras ! ☺

Theorem

| ∃ Le monitorat à l'ENST Bretagne le soir !

| En monitorat tu iras ! ☺

Monitorat

| \exists Le monitorat à l'ENST Bretagne le soir !

| En monitorat tu iras ! 😊

| \exists Le monitorat à l'ENST Bretagne le soir !

| En monitorat tu iras ! 😊

Theorem

| \exists Le monitorat à l'ENST Bretagne le soir !

| En monitorat tu iras ! 😊

Monitorat

| \exists Le monitorat à l'ENST Bretagne le soir !

| En monitorat tu iras ! ☺

| \exists Le monitorat à l'ENST Bretagne le soir !

| En monitorat tu iras ! ☺

Theorem

| \exists Le monitorat à l'ENST Bretagne le soir !

| En monitorat tu iras ! ☺



Le plan

- 1 Introduction
 - Bibliographie
 - Petite histoire
- 2 Langage
 - Généralités
- 3 Un ordinateur ? Mais qu'est-ce ?
 - Les bases de la numérologie
 - Calcul binaire
- 4 Déclarations
 - Généralités
 - Types des objets
 - Portée
- 5 Expressions
 - Sémantique opérateurs
 - Coercition de type
- 6 Instructions
- 7 Préprocesseur
- 8 Bibliothèques
- 9 Éditeur exemple d'Emacs
 - Tuning de son clavier
- 10 Compilation
- 11 Modularisation
- 12 Mise au point
 - Dévermineur (debugger)
 - Analyse mémoire
- 13 Extensions
 - Du C pour le graphisme
 - C++
- 14 Délires
 - Le bêtisier
 - Concours de programmes incompréhensibles
- 15 Conclusion
- 16 Index
- 17 Table des matières

Préhistoire

(I)

Projet MULTICS : *MULTIplexed Information and Computing Service*

- MIT, Bell Telephone Laboratories de AT&T, General Electric
- Offrir puissance de calcul pour toute la ville de Boston : le Minitel avant l'heure
- Notion de « Computer Grid » (qui revient de nos jours...)
- Plus difficile que prévu ↗ abandonné mais grande influence dans la communauté

⊜ encore sous forme de secte <http://www.multicians.org> ☺



C & Unix Story

(I)

- En attendant la suite, Ken THOMSON de BTL écrit un jeu *Space Travel* qu'il fait tourner sur un PDP-7 (machine pas trop chère)
<http://en.wikipedia.org/wiki/PDP-7>
- Problème : pas d'environnement de développement sur PDP-7 et nécessité de faire de l'assemblage croisé sur Honeywell 635 roulant GECOS
- Machine de 32K mots de 18 bits : un microcontrôleur de nos jours ! ☺
- Pour faciliter le développement du jeu, développement d'un système d'exploitation pour le PDP-7 : système de fichier simple (*s5fs*), système de gestion de processus, interpréteur de commande (*shell*). Développé en assembleur (langage machine)
- Le système devient auto-suffisant et est nommé *Unix* en 1969, jeu de mots (*eunuchs*) aussi en opposition à *Multics*



C & Unix Story

(II)

- Portage d'Unix sur PDP-11 et développement de l'éditeur de texte `ed` et du système de composition de texte `runoff`
- Développement du langage interprété *B* utilisé pour développer les outils
- Dennis RITCHIE fait évoluer le langage en C dont le succès a largement dépassé le cadre d'Unix
- 1972 : 10 machines sous Unix...
- Unix réécrit en C en 1973 et la distribution version 4 contient elle-même `cc`
 - ▶ Simple & facile à assimiler
 - ▶ Langage proche de la machine (pour système d'exploitation)
 - ▶ ≈ Macro-assembleur
 - Gestion explicite de la mémoire
 - « Pointeurs »
 - Optimisation manuelle possible
 - ▶ Mais avec des caractéristiques de langage de haut niveau



C & Unix Story

(III)

- Structures de données complexes
- Portabilité
- ▶ Préprocesseur
- ▶ Portabilité
- ▶ Adaptabilité à différents contextes et styles de programmation « impérative »
- L'université de Berkeley récupère une licence (gratuite à cause d'un procès antitrust de 1956 entre AT&T et Western Electric Company)
- Travaux à SRI de Doug ENGELBART sur interfaces graphiques dans les années 1960 repris ensuite chez Xerox PARC
- La version 7 de 1979 est la première version réellement portable
- Beaucoup d'améliorations fournies par les utilisateurs eux-mêmes (de même que BSD & Linux maintenant) favorisé par le côté non commercial

C & Unix Story

(IV)

- MicroSoft et Santa Cruz Operation collabore sur un portage pour i8086 : Xenix
- Portage sur machine 32 bits (Vax-11) en 1978 : UNIX/32V qui est récupérée par Berkeley
(<http://www.lpl.arizona.edu/~vance/www/vaxbar.html> VaxBar)
- 1978, publication de « *The C Programming Language* » décrivant le C version K&R (Brian KERNIGHAN & Dennis RITCHIE)
- Rajout d'utilitaires (csh de Bill Joy) et d'un système de pagination
- La DARPA donne un contrat à Berkeley pour implémenter IP : BSD
 - ▶ Dernière version en 1993 : 4.4BSD. En tout : apport des *socket*, d'IP, d'un *fast file system* (FFS), des signaux robustes, la mémoire virtuelle
 - ▶ Société BSDI créée pour vendre 4.4BSD *lite* en 1994, débarrassé de tout code d'origine AT&T



C & Unix Story

(V)

- 1982 : loi antitrust qui éclate AT&T en baby-Bell dont le AT&T Bell Laboratories qui peut alors commercialiser Unix
 - ▶ 1982 : System III
 - ▶ 1983 : System V
 - ▶ 1984 : System V release 2 (SVR2)
 - ▶ 1987 : System V release 3 (SVR3) introduit les IPC (InterProcess Communications : mémoire partagée, sémaphores), les STREAMS, le *Remote File Sharing*, les bibliothèques partagées,...
 - ▶ Base de nombreux Unix commerciaux
- 1982 : Bill Joy quitte Berkeley pour fonder Sun Microsystems. Adaptation de 4.2BSD en SunOS qui introduit le *Network File System*, interface de système de fichier générique, nouveau mécanisme de gestion mémoire
- Langage C standardisé par l'ISO en 1989 (ISO/IEC 9899 :1990)
- Nouvelle version en 1999 : C99 (ISO/IEC 9899 :1999)
<http://www.open-std.org/JTC1/SC22/WG14/www/C99RationaleV5.10.pdf>



C & Unix Story

(VI)

- Commence à être bien répandue dans les compilateurs
- Normalisation en parallèles d'un UNIX abstrait, POSIX (*Portable Operating System Interface for uniX*)
<http://en.wikipedia.org/wiki/POSIX>
 - ▶ Définit des concepts en abstractions (processus, fichiers...)
 - ▶ Interface normalisé de programmation
 - API (Application Programming Interface)
 - Commandes shell et autres
 - Communications réseau et interprocessus
 - Temps réel
 - ▶ Dépasse le cadre d'UNIX : d'autres systèmes peuvent offrir une même interface sans avoir les concepts en natif : Windows, Mac OS X...
- Projet GNU poussé par Richard STALLMAN de la FSF pour développer des programmes libres de type POSIX
 - ▶ Toutes les commandes classiques UNIX
 - ▶ Compilateur C portable & recible gcc et pour d'autres langages

C & Unix Story

(VII)

- ▶ Éditeur Emacs
- ▶ A facilité le développement de systèmes d'exploitations complets style Linux

Notations typographiques utilisées dans ce cours (I)

- Les blancs (n'apparaissant pas...) sont typographiés de manière visible (merci L^AT_EX avec le style `listings!` ☺)
 - ▶ 3 jolies espaces
 - ▶ 2 tabulations
- Pas la peine de chercher ces caractères graphiques sur votre clavier ni de les écrire sur les copies d'examen ☺
- Mot clé du langage `goto`_{bed}
- Commentaires `/* Comme_en_terre */`
- Chaînes de caractères "chêne_de_caractère"

Le plan

- 1 Introduction
 - Bibliographie
 - Petite histoire
- 2 Langage
 - Généralités
- 3 Un ordinateur ? Mais qu'est-ce ?
 - Les bases de la numérologie
 - Calcul binaire
- 4 Déclarations
 - Généralités
 - Types des objets
 - Portée
- 5 Expressions
 - Sémantique opérateurs
 - Coercition de type
- 6 Instructions
- 7 Préprocesseur
- 8 Bibliothèques
- 9 Modèle mémoire & allocation
- 10 Entrées-sorties
- 11 GNOME
- 12 Éditeur exemple d'Emacs
 - Tuning de son clavier
- 13 Compilation
- 14 Modularisation
- 15 Mise au point
- 16 Dévermineur (debugger)
- 17 Analyse mémoire
- 18 Extensions
 - Du C pour le graphisme
 - C++
- 19 Délires
 - Le bêtisier
 - Concours de programmes incompréhensibles
- 20 Conclusion
- 21 Index
- 22 Table des matières

Le plan

- 1 Introduction
 - Bibliographie
 - Petite histoire
- 2 Langage
 - Généralités
- 3 Un ordinateur ? Mais qu'est-ce ?
 - Les bases de la numérologie
 - Calcul binaire
- 4 Déclarations
 - Généralités
 - Types des objets
 - Portée
- 5 Expressions
 - Sémantique opérateurs
 - Coercition de type
- 6 Instructions
- 7 Préprocesseur
- 8 Bibliothèques
- 9 Éditeur exemple d'Emacs
 - Tuning de son clavier
- 10 Compilation
- 11 Modularisation
- 12 Mise au point
 - Dévermineur (debugger)
 - Analyse mémoire
- 13 Extensions
 - Du C pour le graphisme
 - C++
- 14 Délires
 - Le bêtisier
 - Concours de programmes incompréhensibles
- 15 Conclusion
- 16 Index
- 17 Table des matières

Des instructions et expressions C... (I)

- Ordinateur exécute des instructions qui interagissent avec des données
- Ensemble ordonné d'instructions et de données ≡ programme
- Instructions en C constituées d'expressions

```
1 a = 4 ;  
2 c  
    =  
4    d [ i ] ;
```

Definition

Une instruction C est une expression terminée par un « ; »

- Expressions récursivement composées d'expressions

```
b = 3 + 4 * cos ( x ) ;
```

Des instructions et expressions C...

(II)

- Regroupement possible d'instructions dans un bloc entre « { » et « } »

... en passant par les fonctions C...

(I)

- Rassemble instructions de manière modulaire
- Prennent éventuellement des paramètres
- Fonctions définies par des utilisateurs
- Fonctions « standard »
 - ▶ Mathématique : `cos(x)`
 - ▶ Entrées/sorties, système,... : `exit(0)`
- **Sens de l'économie du C** : pas d'instructions spécifiques pour faire des entrées-sorties
 - Pas de **PRINT** à la Fortran, Basic, PERL, Python
 - Utilise des fonctions externes optionnelles
 - Simplifie le langage
 - Permet une adéquation avec les besoins
- Algorithmique et structures de données : `strdup(chaine)`
- ...
- **Sens de l'économie du C** : une procédure ou une sous-routine est une fonction qui ne renvoie rien

... en passant par les fonctions C... (II)

- ▶ Pas besoin d'instruction d'appel particulière (**CALL** en Fortran ou Basic)
- ▶ Subliminal : doit avoir un « effet de bord » pour servir à quelque chose (par exemple entrée/sortie)

... aux programmes C

(I)

- Sens de l'économie du C : pas de précision par mot clef style **PROGRAM** de Fortran

Definition

Un programme C est une fonction de nom `main()` qui prend des arguments textuels en paramètre et renvoie un code de retour d'erreur entier

- Système d'exploitation fournit arguments de ligne de commande comme paramètres



Programme minimal en C

(I)

```
1 int main(int argc, char *argv[]){
2     return 0;
}
```

Programme hello world en C

(I)

```
1 /* Pour pouvoir utiliser proprement
   la fonction de sortie : */
2 #include <stdio.h>
3 /* Définit entre autres les codes de retour : */
4 #include <stdlib.h>
5
6
7 int main(int argc, char *argv[]){
8     /* Affiche sur la « sortie-standard » */
9     puts("Hello, world!");
10    // Renvoie une marque de réussite :
11    return EXIT_SUCCESS;
12 }
```

- Dans la lignée de tous les « Hello World » en plein de langages
http://en.wikipedia.org/wiki/Hello_world#C
- Lieu de l'affichage dépend du contexte : terminal, imprimante, courriel, page WWW, néant...
- Fin optimiste... 🚨 Est-ce que l'affichage a vraiment eu lieu ?

Commentaires

(I)

- Penser à ALZHEIMER qui nous guette...
- Ne pas surcommenter non plus (surtout avec des commentaires faux ☺)
- Permet aussi de momentanément ôter des bouts de programme
- Tout texte (multi-ligne) entre /* et */
 Ne sont pas récursif ! /* Commentaire */ J'insiste /* eh non ! */
- Lignes commençant par // : arrivé dans C99 par culture C++ & Java

Vers un programme exécutable

(I)

Ordinateur incapable d'exécuter instructions de haut niveau ↗
compilation en instructions machine

- Préprocesseur : gère dans code source les #quelque-choses, macro-instructions... ↗ code source expansé
- Compilateur : traduit source expansé en langage d'assemblage (instructions machines)
- Assembleur : traduit langage d'assemblage en instructions binaires exécutables (code « objet »)
- Éditeur de lien : tisse programme exécutable définitif à partir de plusieurs fichiers codes objet (bibliothèques de fonctions standard, de démarrage de `main()`, etc)



Compilation d'un programme

(I)

- Appel du compilateur en direct rapidement compliqué
- Utilisation d'outils tels que `make` de gestion des phases de compilation à partir d'un `Makefile` de configuration
- **Utiliser si possible les comportements par défaut** (sans `Makefile` par exemple ! ☺)

```
1 $ make hello_world  
2 cc hello_world.c -o hello_world  
3 $ ./hello_world  
4 Hello, world!
```

Compilation d'un programme

(II)

- Et si on retape make ?

```
1 $ make_hello_world
2 make: « hello_world » est à jour.
$ ls -l
4 -rwxr-xr-x 1 keryell keryell 6918 2005-10-12 22:03 hello_world
-rw-r--r-- 1 keryell keryell 320 2005-10-12 21:44 hello_world.c
```

make fait de la compilation paresseuse : compiler que ce qui est nécessaire

Indispensable pour de gros projets !

- Éviter de prendre un nom de commande qui existe déjà... 
test.c ☺

Déclarer des fonctions

(I)

- Entête : déclare nom, paramètres et types associés et type de valeur renournée
- Corps de la fonction : calcul à faire
- Exécution s'arrête sur **return** avec éventuellement valeur de retour
- On peut remplacer tout appel d'une fonction par corps de fonction en rajoutant **inline** devant déclaration de fonction
 - ▶ Gain éventuel en vitesse ☺
 - ▶ Augmentation taille de programme ☺

```
1 /* Ne renvoie rien : une procédure en fait */
2 void
3 begin_timer(void /* Pas d'argument */)
4 {
5     gettimeofday(&time_begin, NULL);
6 }
```

Déclarer des fonctions

(II)

```
1  /* return the elapsed time in seconds
2   since the last begin_timer(): */
3  double end_timer(void)
4  {
5    double run_time;
6    gettimeofday(&time_end, NULL);
7    /* Take care of the non-associativity in floating point :-)
8    run_time = time_end.tv_sec - time_begin.tv_sec
9    + (time_end.tv_usec - time_begin.tv_usec) / 1e6;
10   return run_time;
11 }
```

Déclarer des fonctions

(III)

```
1 double  
assurance_mensuelle(double montant, double taux)  
2 {  
3     /* Le taux est un pourcentage par an  
4     sur le montant total du prêt. */  
5     return montant * taux / 12;  
6 }  
7 }
```

Types et expressions

(I)

- Valeur d'expression calculée à partir d'un opérateur et des valeurs d'opérandes
- Une valeur a un type qui définit son domaine de validité
 - ▶ Booléen : la base de la vérité vraie
`true` & `false` (`false`)
 - ▶ Nombre entier : sous-ensemble fini de \mathbb{Z}
 $2/3$ (0)
 - ▶ Nombre flottant : sous-ensemble fini de \mathbb{D}
 $2/3$ ($\approx 0,6666 \dots$)
 - ▶ Des chaînes de caractères
`"comme_ceci"`
 - ▶ Plein d'autres types plus compliqués ou définis par le programmeur...

Variables et déclaration

(I)

- Entités capables de stocker des valeurs ou « objets »
-  Déclarations nécessaires avant usage contrairement à d'autres langages et du style

```
type nom ;  
type nom = valeur_initiale;
```

- Avant C99 : variables qu'en début de bloc d'instructions
- Culture C++ & Java ↘ C99 : distribution des déclarations comme on veut

Déclarer là où c'est le plus clair

- En C99 possibilité d'avoir des valeurs initiales non connues lors de la compilation

Variables et déclaration

(II)

```
1 foo(float f, float g)
2 {
3     float beat_freqs [2] = {f-g, f+g};
4     /* ... */
}
```

Affectation de variables

(I)

Utilisation de l'opérateur ( pas de l'instruction...) « = »

```
1 a = 3;  
2 d = (c = e/f)*2;
```

Opérateur qui renvoie valeur affectée (pas adresse), donc équivalent
à

```
1 c = e/f;  
2 d = c*2;
```

Visibilité des variables

(I)

- Variables globales

- ▶ Définie en dehors d'un bloc d'instruction
- ▶ 2 sortes de visibilités :
 - Définie globalement pour toutes les unités de compilation
 - Définie globalement à un seul fichier (unité de compilation)

- Variables locales

- ▶ Définie dans bloc d'instructions
- ▶ Dans une fonction : idem (cas particulier de bloc)

Visibilité des variables

(II)

- Paramètre de fonction

- ▶ Cas particulier de variable locale
- ▶ Valeur copiée de la valeur d'appel de la fonction
- ▶ Passage de paramètre par valeur ou recopie ↗ si on modifie dans la fonction le paramètre cela ne modifie pas la valeur d'appel
- ▶ Le contraire du langage Fortran

La vie est un long fleuve tranquille

(I)

- Flot d'instructions toujours séquentiel
 - ▶ Applications orientées flux de données, pipeline
 - ▶ Traitement du signal, rendu de pixels dans cartes graphiques...
 - ▶ ...mais bien trop limitatif dans cas général
- Besoin d'avoir du *contrôle de flot* pour varier cours exécution
 - ▶ Instructions conditionnelles
 - ▶ Boucles
 - ▶ Sauts

Tests

(I)

- Exécute de manière conditionnelle 1 branche parmi 1 ou 2

```
1  if (condition)
2    instruction_exécutée_si_vrai
3    else
4      instruction_exécutée_si_faux
```

- La partie **else** est optionnelle

```
1  if (a>b)
2    max=a;
3    else {
4      max=b;
5      echange=1;
6    }
```

Boucle tant que

(I)

- Itération sur condition vraie

```
1 while « condition )
2   «« instruction_corps_de_boucle_à_faire
```

- Variante avec condition testée seulement à la fin ( corps au moins exécuté au moins une fois)

```
1 do
2   «« instruction_corps_de_boucle_à_faire_au_moins_une_fois
  while « condition );
```

Boucle pour

(I)

- Sucre syntaxique de la boucle **while** permettant (par exemple) de gérer des indices de boucle

```
1  for (expression_initiale;  
2      expression_de_test;  
3          expression_avant_rebouclage)  
4      instruction_corps_de_boucle  
  
1  for (i=0; i<number_of_skewing ;i++){  
2      for(j=0;j<size;j++)  
3          pointers_to_fingers[j]=(int)((int)(drand48())*size)*sizeof(int*));  
4      testerreur("Unable_to_write_pointer_file_random\"%s\"",  
5                  write(fd,(char*)&pointers_to_fingers[0],buf.st_size)  
6                  ,buf.st_size,name);  
}
```

Bien présenter les programmes

(I)

- Être capable de lire (rapidement !) son programme... et ceux des autres !
- Important si travail collaboratif
- Langage C pas orienté ligne (sauf préprocesseur)
 - ▶ Source de guerre de religions entre langages ☺
 - ▶ Fortraneux : trouvent idiot qu'en C on doive mettre des « ; »
 - ▶ C et perl : trouvent idiot qu'en Fortran on soit assez prisonnier des lignes
 - ▶ Pythonneux : pensent que la syntaxe doit refléter la présentation (et réciproquement)
- ↗ On peut faire cryptique... ∃ des concours ☺
- Mais on peut utiliser la souplesse du C pour faire joli
- Différentes écoles (style BSD, Linux...)
- S'adapter en fonction des projets

Bien présenter les programmes

(II)

- Éditeurs de textes (les vrais ☺) et autres IDE à la rescousse pour aider la présentation
 - ▶ Outils souvent très configurables
 - ▶  WordPad, textedit et consorts *ne sont pas* des éditeurs de textes pour faire la programmation!!! ☺



Règle de base

(I)

- Un caractère dans un fichier coûte $\approx 1,5 \cdot 10^{-10}$ € 2008 ☺
- Ne pas hésiter à rajouter des sauts de lignes, des espaces épaisses...
- Quelques possibilités
 - ▶ Une déclaration de variable par ligne
 - ▶ Une instruction élémentaire (pas bloc) par ligne
 - ▶ Structure de contrôle : début de ligne
 - ▶ Accolade ouvrante : fin de ligne
 - ▶ Accolade fermante : seule sur une ligne
 - ▶ Étiquette : seule sur une ligne

Comme toutes les règles, juste des idées à adapter

 Dans le cadre d'un projet, d'une entreprise,... se mettre d'accord car mélange de style pénible ! ☺

Indentation

(I)

- Idée : utiliser des espaces en tête de ligne pour faire ressortir localité/modularité/hiérarchie des structures
- Intérêt du concept : portabilité de cette présentation en dehors de son propre éditeur ou IDE
-  Caractère « tabulation » interprété différemment selon les systèmes et configurations
 - ▶ Souvent tabulation ≈ 8 espaces
 - ▶ Tabulations pouvant être des déplacements à des positions fixes (ex. multiple de 8)
 - ▶ Perso : tabulations de 4
 - ▶ Quid si mélange de différents styles ? 😐

Indentation

(II)

```
1 static_void skew_the_file ( int_number_of_skewing , char *name )
{
2     int fd , i , j , skew , size_end , size_begin , size ;
3     struct stat buf ;
4
5     pointers_to_fingers = ( int ** ) malloc ( buf . st_size );
6
7     size = buf . st_size / sizeof ( int * );
8
9     if ( random_pointers ){
10        for ( i = 0 ; i < number_of_skewing ; i ++ ){
11            for ( j = 0 ; j < size ; j ++ )
12                pointers_to_fingers [ j ] = ( int )( drand48 () * size ) * sizeof ( int );
13                testerreur ( "Unable_to_write_pointer_file_random\\"%s\\\"",
14                write ( fd , ( char * ) & pointers_to_fingers [ 0 ] , buf . st_size )
15                , buf . st_size , name );
16        }
17    }
18
19    else_if ( zero_pointers ){
20        for ( i = 0 ; i < number_of_skewing ; i ++ ){
21            for ( j = 0 ; j < size ; j ++ )
22                pointers_to_fingers [ j ] = NULL ;
23                testerreur ( "Unable_to_write_pointer_file_random\\"%s\\\"",
24                write ( fd , ( char * ) & pointers_to_fingers [ 0 ] , buf . st_size )
25                , buf . st_size , name );
26    }
27}
```



Indentation

(III)

```
25     ----->-----> buf.st_size , name);
26 }
27 }
28 else{
29     testerreur ("Unable_to_read_pointer_file \"%s\"",
30             read(fd,(char*)pointers_to_fingers ,buf.st_size)buf.st_
31             name);

33 /* The_file_size_is_multiplied_by_number_of_skewing: */
34 for(i=1;i<number_of_skewing ;i++){
35     skew=i%(buf.st_size/sizeof(int *));
36     size_end=skew*sizeof(int *);
37     size_begin=buf.st_size-size_end;
38     testerreur ("Unable_to_write_pointer_file_begin \"%s\"",
39             write(fd,(char*)&pointers_to_fingers [skew],size_begin )
40             ,size_begin ,name);
41     testerreur ("Unable_to_write_pointer_file_end \"%s\"",
42             write(fd,(char*)&pointers_to_fingers [0],size_end)
43             ,size_end ,name);
44 }
45 }
46 testerreur ("Unable_to_close_pointer_file \"%s\"",close(fd),name);
47 }
```

Indentation

(IV)

⚠ Encore : dans le cadre d'un projet, d'une entreprise,... se mettre d'accord car mélange de style pénible ! ☺



Le plan

- 1 Introduction
 - Bibliographie
 - Petite histoire
- 2 Langage
 - Généralités
- 3 Un ordinateur ? Mais qu'est-ce ?
 - Les bases de la numérologie
 - Calcul binaire
- 4 Déclarations
 - Généralités
 - Types des objets
 - Portée
- 5 Expressions
 - Sémantique opérateurs
 - Coercition de type
- 6 Instructions
- 7 Préprocesseur
- 8 Bibliothèques
- 9 Modèle mémoire & allocation
- 10 Entrées-sorties
- 11 GNOME
- 12 Éditeur exemple d'Emacs
 - Tuning de son clavier
- 13 Compilation
- 14 Modularisation
- 15 Mise au point
 - Dévermineur (debugger)
 - Analyse mémoire
- 16 Extensions
 - Du C pour le graphisme
 - C++
- 17 Délires
 - Le bêtisier
 - Concours de programmes incompréhensibles
- 18 Conclusion
- 19 Index
- 20 Table des matières

Le plan

- 1 Introduction
 - Bibliographie
 - Petite histoire
- 2 Langage
 - Généralités
- 3 Un ordinateur ? Mais qu'est-ce ?
 - Les bases de la numérologie
 - Calcul binaire
- 4 Déclarations
 - Généralités
 - Types des objets
 - Portée
- 5 Expressions
 - Sémantique opérateurs
 - Coercition de type
- 6 Instructions
- 7 Préprocesseur
- 8 Bibliothèques
- 9 Éditeur exemple d'Emacs
 - Tuning de son clavier
- 10 Compilation
- 11 Modularisation
- 12 Mise au point
 - Dévermineur (debugger)
 - Analyse mémoire
- 13 Extensions
 - Du C pour le graphisme
 - C++
- 14 Délires
 - Le bêtisier
 - Concours de programmes incompréhensibles
- 15 Conclusion
- 16 Index
- 17 Table des matières

Un peu de numérologie

(I)

Langage C proche du matériel ↗ indispensable de comprendre comment sont représentés les nombres

1612 : John NAPIER, première trace des décimaux (nombres flottants...), logarithmes et méthodes de multiplication

1703 : Gottfried LEIBNIZ « *Au lieu de la progression de dix en dix, j'ai employé depuis plusieurs années la progression la plus simple de toutes, qui va de deux en deux* » : binaire. Simple car 2 chiffres (bit) ou 2 états.

$$8_{10} = 1000_2$$



Représentation des entiers

(I)

Représentation canonique de $a \in \mathbb{N}$ en base b :

$$a = (x_n x_{n-1} \cdots x_2 x_1 x_0)_b$$

avec $x_i \in \mathbb{N}/b\mathbb{N}$ et

$$a = \sum_{i=0}^n x_i \cdot b^i$$

- Choix de base adapté au contexte

- Humains européens standard avec 10 doigts : choix de la base 10
- Babyloniens : mélange de base 6 et 10
- Certaines communautés africaines comptent en base 12
- Boulier chinois : base 10 et 5
- Différentes bases pour le temps
 - Secondes et minutes : base 60 (combien a duré la minute de 23h59 le 31/12/2005 ?)
 - Heures : 12 ou 24

Représentation des entiers

(II)

- Jours : 7 dans la semaine 28–31 dans le mois
- Mois : 12

Ǝ des historiens des nombres !

Numérotation binaire

(I)

- Anciens ordinateurs analogiques : manipulation directe de grandeurs physiques (tension, pression, force, champ magnétique) ou temporelles (opérateurs statistiques) liées aux données
- Ordinateurs modernes utilisent composants à état binaire plus simples et plus robuste au monde extérieur
- ↗ Numérotation binaire (base 2) choisie en interne
 - ▶ Liaison avec l'algèbre de George BOOLE (19^{ème} siècle)
 - ▶ Chiffre binaire : *bit* (*BInary digiT*) ou *b*
 - ▶ Décliné pour débits d'information : *b/s*
- Néanmoins autres codages utilisés dans contextes particuliers
 - ▶ Symboles en communications : débit de symbole : *baud*
 - ▶ Codages redondants ou « mous » (aussi pour l'argent liquide !) pour aller vite : voir cours IAHP de master recherche informatique

Unités binaires

(I)

- Dispositifs de stockage binaires souvent adressés en binaire ↗ des capacités en puissance de 2
- « Bonjour Madame, je voudrais un ordinateur portable de 4 294 967 296 bits de mémoire » : peu pratique... ☺
- Constat que $2^{10} = 1024 \approx 10^3 = 1000$
- Constat aussi que $(\frac{3}{2})^{12} \approx 2^7 = 128$ ↗ base de la musique moderne européenne 12 quintes \approx 7 octaves, découpage de la gamme en 12 demi-tons ...
- Développement dans les uSI d'un pendant binaire
<http://physics.nist.gov/cuu/Units/binary.html>
http://fr.wikipedia.org/wiki/Pr%C3%A9fixe_binaire

Unités binaires

(II)

| Décimal | | Binaire | | |
|-----------|---------|----------|----------------|---------------------------|
| Facteur | Préfixe | Facteur | Préfixe | Détail |
| 10^3 | k | 2^{10} | Ki (kibi) | 1 024 |
| 10^6 | M | 2^{20} | Mi (mébi/mibi) | 1 048 576 |
| 10^9 | G | 2^{30} | Gi (gébi/gibi) | 1 073 741 824 |
| 10^{12} | T | 2^{40} | Ti (tébi/tibi) | 1 099 511 627 776 |
| 10^{15} | P | 2^{50} | Pi (pébi/pibi) | 1 125 899 906 842 624 |
| 10^{18} | E | 2^{60} | Ei (exbi) | 1 152 921 504 606 846 976 |

- Mémoires d'ordinateurs mesurées en unités binaires : 512 Mio
 - Mémoires magnétiques et débits réseaux en uSI classiques
- ~  Grosses confusions en vue... ☺

Autres bases dérivées du binaire

(I)

- Compter en binaire sur de grosses valeurs : lourd ! ☺
- Souvent données binaires codées sur nombre entier de bits
 - ▶ Entiers
 - ▶ Clés, signatures
 - ▶ ...
- Utiliser une base regroupant plusieurs bits par chiffre



Base 8 (octal)

(I)

$$\text{Base } 8 = 2^3$$

- Plus compact que le binaire
- Exprimable avec des chiffres normaux
- Pratique pour représenter des choses sur 3 bits
- Droit sur fichiers codés dans un nombre entier : drapeaux binaires
 - rwxr-xr-x 1 keryell keryell 6918 2005-10-12 22:03 hello_world

Pour chaque ugo (*user/group/other*) et autre ACL (Access Control List)

- ▶ r = 2^2
- ▶ w = 2^1
- ▶ x = 2^0

~ chiffre octal pour chaque ugo : 755₈

- *umask*

Base 8 (octal)

(II)

- ▶ Suppression de droits par défaut lors de création de fichier et directement codé en octal
- ▶ $0026_8 \equiv$ supprime droit d'écriture pour membres groupe et droit lecture+écriture pour autres
- Représentation des caractères (comportement par défaut de od)

Base 16 (hexadécimal)

(I)

- Binaire par tranche de 4 bits
- 10 chiffres + 6 lettres a (10) à f (15)
- Faisable avec afficheur 7 segments
- Très utilisé en informatique et électronique numérique
- Cas particulier BCD (binaire codé décimal), hexadécimal où on n'utilise que chiffres 0 à 9
 - ▶ Calculs plus compliqués
 - ▶ Conversions en décimal humain plus simple (électronique)
 - ▶ En voie de disparition

http://en.wikipedia.org/wiki/Binary-coded_decimal



Base 256

(I)

- Utilisé pour représenter addresses IPv4 numériquement

$$\begin{aligned}(193.50.97.146)_{256} &= ((256_{10} \times 193_{10} + 50_{10}) \times 256_{10} + 97_{10}) \times 256_{10} + 146_{10} \\ &= 3241304466_{10}\end{aligned}$$

Plus simple pour manipuler des préfixes de classe A(/8), B(/16) ou C(/24)

- Caractères affichables codés souvent par valeur 8 bits
 - Chaque caractère peut être vu comme chiffre en base 256
 - Détaillé plus tard dans le cours



Authentification avec mot de passe jetable S/Key (I)

- Utilisation de challenge sur 64 bits que l'utilisateur doit copier dans un dispositif d'authentification qui calculera le mot de passe à utiliser
- Difficile de recopier rapidement et sans erreur « E79F 3CA6 8C57 E381 » par exemple
- Exemple ultime : utiliser des chiffres de 11 bits qui sont des mots courts choisis dans vocabulaire anglais
 - ▶ Plus long à taper
 - ▶ Mais moyen mnémotechnique et CRC (mot anglais) ☺

TANK WELT MOT HAL FATE MUSH



Base 65536

(I)

- Utile pour manipuler adresses ou grosses valeurs : 32, 64, 128... bits
- Regroupement de valeurs hexadécimales par paquet de 16 bits
- Adresses IPv6
 - ▶ 128 bits : 2001:660:7302:e771:201:2ff:fef:a64ee
 - ▶ Raccourci :: pour une suite de blocs de 16 bits à 0 : 2001:7a8:b057::5
- Codage de caractères sur 16 bits : UTF-16 ou 1 caractère ≡ 1 chiffre

Le plan

- 1 Introduction
 - Bibliographie
 - Petite histoire
- 2 Langage
 - Généralités
- 3 Un ordinateur ? Mais qu'est-ce ?
 - Les bases de la numérologie
 - Calcul binaire
- 4 Déclarations
 - Généralités
 - Types des objets
 - Portée
- 5 Expressions
 - Sémantique opérateurs
 - Coercition de type
- 6 Instructions
- 7 Préprocesseur
- 8 Bibliothèques
- 9 Éditeur exemple d'Emacs
 - Tuning de son clavier
- 10 Compilation
- 11 Modularisation
- 12 Mise au point
 - Dévermineur (debugger)
 - Analyse mémoire
- 13 Extensions
 - Du C pour le graphisme
 - C++
- 14 Délires
 - Le bêtisier
 - Concours de programmes incompréhensibles
- 15 Conclusion
- 16 Index
- 17 Table des matières

Représenter des nombres négatifs

(I)

- Pas que des entiers naturels...
- Rajouter bit de signe ? Ou utiliser astuce arithmétique modulo b^n pour des nombres de n chiffres en base b ...
- Remarquer que $a + ((b - 1) - a) = b - 1$
- Permet de définir nombres quasi-opposés. Exemple en base 10 d'un complément à 9 :

$$\begin{array}{r} & 1 & 2 & 3 \\ + & 8 & 7 & 6 \\ \hline & 9 & 9 & 9 \end{array}$$

Complément restreint $\overline{123}_{10} = 876_{10}$ dans $\mathbb{Z}/b^n\mathbb{Z}$

Représenter des nombres négatifs

(II)

- $a + \bar{a} \equiv -1[b^n]$ et donc

$$a + (\bar{a} + 1) \equiv 0[b^n]$$

- Complément vrai (à b) $-a \equiv \bar{a} + 1[b^n]$

$$\begin{array}{r}
 & 4 & 5 & 2 \\
 - & 1 & 2 & 3 \\
 \hline
 & 3 & 2 & 9
 \end{array}
 \quad
 \begin{array}{r}
 & 4 & 5 & 2 \\
 + & 8 & 7 & 7 \\
 \hline
 & \boxed{1} & 3 & 2 & 9
 \end{array}$$

- Pratique

- ▶ Simple à calculer
- ▶ Pas besoin de rajouter opérateur de soustraction
- ▶ Le +1 fait en utilisant entrée retenue additionneur : gratis

Nombres binaires en complément à 2

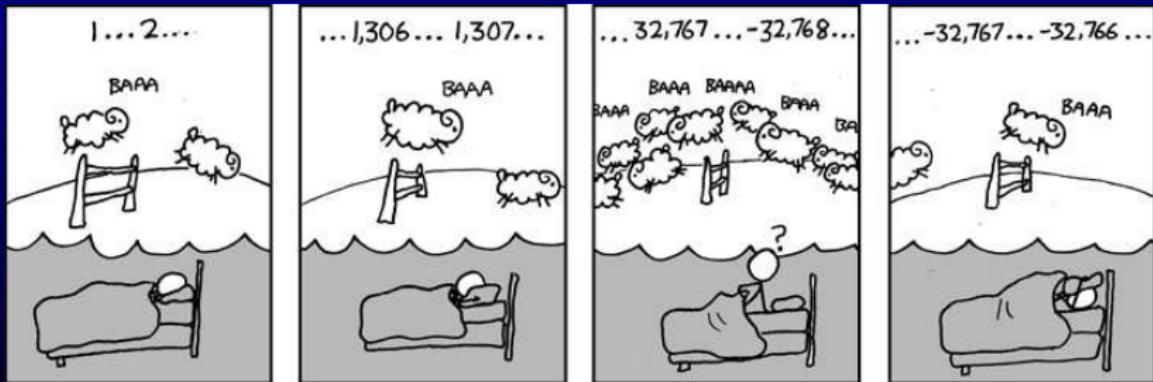
(I)

- Cas particulier électronique : 2 états
- Nombres naturels sur n bits : $[0, + 2^n - 1]$
- Complément restreint « à 1 » : simple inversion bit à bit (opérateur ~ en C)
- Complément vrai « à 2 » : (opérateur - en C)
- Convention : utilisation nombres signés en complément à 2 sur n bits $[-2^{n-1}, + 2^{n-1} - 1]$
 - ▶ $-2^{n-1} \equiv (1000 \dots 00)[2^n]$
 - ▶ $-1 \equiv (1111 \dots 11)[2^n]$
 - ▶ $0 \equiv (0000 \dots 00)[2^n]$
 - ▶ $+1 \equiv (0000 \dots 01)[2^n]$
 - ▶ $+2^{n-1} - 1 \equiv (0111 \dots 11)[2^n]$
- Bit de poids fort donne le signe
- Extension de précision de n à m bits, $m > n$
 - ▶ $+3_{10}$, $\boxed{0} 011_2 \rightsquigarrow \boxed{00000} 011$

Nombres binaires en complément à 2

(II)

- -3_{10} , $\boxed{1}101_2 \rightsquigarrow \boxed{11111}101$
- ↗ Bit de signe utilisé pour compléter bits manquants
- Utile aussi dans décalages à droite (\approx troncature sur nombre plus grand)
- Attention aux comportements qui peuvent être troublants (mais pratiques si on maîtrise...) <http://xkcd.com/571>



Le plan

- 1 Introduction
 - Bibliographie
 - Petite histoire
- 2 Langage
 - Généralités
- 3 Un ordinateur ? Mais qu'est-ce ?
 - Les bases de la numérologie
 - Calcul binaire
- 4 Déclarations
 - Généralités
 - Types des objets
 - Portée
- 5 Expressions
 - Sémantique opérateurs
 - Coercition de type
- 6 Instructions
- 7 Préprocesseur
- 8 Bibliothèques
- 9 Modèle mémoire & allocation
- 10 Entrées-sorties
- 11 GNOME
- 12 Éditeur exemple d'Emacs
 - Tuning de son clavier
- 13 Compilation
- 14 Modularisation
- 15 Mise au point
 - Dévermineur (debugger)
 - Analyse mémoire
- 16 Extensions
 - Du C pour le graphisme
 - C++
- 17 Délires
 - Le bêtisier
 - Concours de programmes incompréhensibles
- 18 Conclusion
- 19 Index
- 20 Table des matières

Le plan

- 1 Introduction
 - Bibliographie
 - Petite histoire
- 2 Langage
 - Généralités
- 3 Un ordinateur ? Mais qu'est-ce ?
 - Les bases de la numérologie
 - Calcul binaire
- 4 Déclarations
 - Généralités
 - Types des objets
 - Portée
- 5 Expressions
 - Sémantique opérateurs
 - Coercition de type
- 6 Instructions
- 7 Préprocesseur
- 8 Bibliothèques
- 9 Éditeur exemple d'Emacs
 - Tuning de son clavier
- 10 Compilation
- 11 Modularisation
- 12 Mise au point
 - Dévermineur (debugger)
 - Analyse mémoire
- 13 Extensions
 - Du C pour le graphisme
 - C++
- 14 Délires
 - Le bêtisier
 - Concours de programmes incompréhensibles
- 15 Conclusion
- 16 Index
- 17 Table des matières

Objets en C

(I)

- Zone de stockage d'information (mémoire, registre)
- Type précisant comportement en mémoire, valeurs possibles
- Mémoire : espace de stockage unique orienté « octet » (caractères de base), décomposable en bits
- Déclaration : définit *statiquement* type de variable, argument ou valeur de retour de fonction (type de fonction)

Types en C

(I)

- Types d'objet
 - ▶ Types scalaires : objets élémentaires proches du matériel
 - Types arithmétiques : pour faire des calculs au sens classique
 - Pointeurs : référence à d'autres objets ou fonctions
 - ▶ Types agrégés
 - Vecteurs : répétition ordonnée d'un même type
 - Structures : rassemble éléments hétérogènes
 - Union : fait coexister plusieurs types au même endroit mémoire
- Types de fonction : précise à la fois type valeur de retour et type arguments
- Types incomplets
 - ▶ Incomplétude totale **void**
 - ▶ Vecteur de taille non précisée
 - ▶ Structure non précisée

Types arithmétiques en C

(I)

- Intégraux (comportement de type entier naturel ou relatif)
 - ▶ Booléens (C99) : `bool`
 - ▶ Caractères : `[signed|unsigned]_char`
 - ▶ Entiers : `[unsigned]_[short|long|long long]_int`
 - ▶ Énumération : entiers nommés
- Nombres flottants : approximation de la continuité
 - ▶ Flottants réels : approximation de \mathbb{R} `float` | `[long]_double`
 - ▶ Flottants complexes (C99) : approximation de \mathbb{C}
`(float | [long]_double)_complex`

Déclaration de variable

(I)

attributs-type type identificateur [,identificateur];*

- Attribut de type

- ▶ Spécifications de rangement

- **auto** : dans la pile
 - **extern** : définie ailleurs
 - **register** : essaye de garder en registre (très rapide mais rare, donc chère...) Moins nécessaire avec bons compilateurs
 - **static** : local au fichier ou garde une vie après la mort

- ▶ Qualificatifs d'aide au compilateur

- **const** : garantie sur l'honneur valeur constante. Plus propre que d'utiliser préprocesseur
 - **restrict** : garantie sur l'honneur \neq alias de pointeurs (C99)
 - **volatile** : \exists monde en dehors du programme

- Identificateur : nom de l'objet

Identificateurs

(I)

- Suite de caractères alphanumériques
- En C99 utilisation possible de caractères étendus (UTF-8...) mais  si travail collaboratif, rachat d'entreprise mondialisé... ↗ programmes sources en anglais/américain. Mais ∃ des mots « étrangers » en américain ☺
- Premier caractère alphabétique ou _
- Sensible à la casse des lettres contrairement à Fortran
- ↗ Utilisation *possible* de la casse pour exprimer une sémantique

► Variables en minuscule `i`

- Utilisation du `_` pour séparer des mots `search_generic_record`
- Utilisation de capitales séparer des mots `searchGenericRecord`

► Constantes du préprocesseur en majuscule `N_ITER`

Choix à faire dans projet, entreprise,...

Identificateurs

(II)

- Préférer variables longues et claires plutôt que courtes et obscures
 - ▶ Utiliser IDE (Eclipse...) ou éditeur (Emacs ! ☺...) avec complétion automatique, menus...

Mots-clés réservés en C

(I)

- ⚠ Ne pas utiliser du langage C comme identifiant!

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

En gras : nouveautés dans C99 par rapport à C89

- Éviter aussi utilisation d'identifiants de bibliothèques standards sauf hack ou programmation système hard core
- Extensions possibles déjà prévues dans le langage C99 et POSIX.1 : éviter `E` suivie d'un chiffre ou majuscule, identifiant commençant par `is`, `to`, `LC_`, `str`, `mem`, `wcs`
- Ne pas définir identifiant commençant par `_` et suivi par une majuscule ou `_`

Mots-clés réservés en C

(II)

- Identifiant commençant par `sig` suivi par `_` ou majuscule

Comparaison avec différents dialectes du C :

<http://www.georgehernandez.com/xComputers/Cs/Keywords.htm>



Le plan

- 1 Introduction
 - Bibliographie
 - Petite histoire
- 2 Langage
 - Généralités
- 3 Un ordinateur ? Mais qu'est-ce ?
 - Les bases de la numérologie
 - Calcul binaire
- 4 Déclarations
 - Généralités
 - Types des objets
 - Portée
- 5 Expressions
 - Sémantique opérateurs
 - Coercition de type
- 6 Instructions
- 7 Préprocesseur
- 8 Bibliothèques
- 9 Éditeur exemple d'Emacs
 - Tuning de son clavier
- 10 Compilation
- 11 Modularisation
- 12 Mise au point
 - Dévermineur (debugger)
 - Analyse mémoire
- 13 Extensions
 - Du C pour le graphisme
 - C++
- 14 Délires
 - Le bêtisier
 - Concours de programmes incompréhensibles
- 15 Conclusion
- 16 Index
- 17 Table des matières

Caractères

(I)

- Utilisés pour saisir programmes avec éditeur et compiler
- Utilisés dans données manipulées par programmes
-  Pas forcément les même encodages... Programme anglais ASCII peut manipuler des données écrites en coréens UTF-8...
-  Bien comprendre la notion de caractères ! Nombres utilisés pour représenter choses ± affichables

Table des caractères ASCII

(I)

- Codage de caractère très utilisé (ISO-646)
- Base de nombreux autres encodages (UNICODE = ISO-10646)
- Inconscient collectif très important

<http://www.georgehernandez.com/xComputers/CharacterSets/ASCII.htm>

| Dec | Binary | Oct | Hex | Char | Remark |
|-----|---------|-----|-----|--------|---|
| 000 | 0000000 | 000 | 00 | CTRL-@ | \0 NUL (Null prompt) |
| 001 | 0000001 | 001 | 01 | CTRL-A | SOH (Start Of Heading, console interrupt) |
| 002 | 0000010 | 002 | 02 | CTRL-B | STX (Start of TeXt, maintenance mode on HP) |
| 003 | 0000011 | 003 | 03 | CTRL-C | ETX (End of TeXt) |
| 004 | 0000100 | 004 | 04 | CTRL-D | EOT (End Of Transmission, not same as ETB) |
| 005 | 0000101 | 005 | 05 | CTRL-E | ENQ (ENquiry, goes with ACK; old HP flow control) |
| 006 | 0000110 | 006 | 06 | CTRL-F | ACK (ACKnowledge, clears ENQ logon hand) |
| 007 | 0000111 | 007 | 07 | CTRL-G | \a BEL (BEL1) |
| 008 | 0001000 | 010 | 08 | CTRL-H | \b BS (BackSpace, works on HP) |
| 009 | 0001001 | 011 | 09 | CTRL-I | \t HT (Horizontal Tab) |
| 010 | 0001010 | 012 | 0A | CTRL-J | \n LF (Line Feed) or NL (New Line) or EOL (End Of Line) |
| 011 | 0001011 | 013 | 0B | CTRL-K | \v VT (Vertical Tab) |
| 012 | 0001100 | 014 | 0C | CTRL-L | \f NP (New Page, page eject) or FF (Form Feed) |
| 013 | 0001101 | 015 | 0D | CTRL-M | \r CR (Carriage Return). Mac EOL (Win EOL is \r\n). |

Table des caractères ASCII

(II)

| | | | | | |
|-----|---------|-----|----|--------|--|
| 014 | 0001110 | 016 | 0E | CTRL-N | S0 (Shift Out, alternate character set) |
| 015 | 0001111 | 017 | 0F | CTRL-O | SI (Shift In, resume default character set) |
| 016 | 0010000 | 020 | 10 | CTRL-P | DLE (Data Link Escape) |
| 017 | 0010001 | 021 | 11 | CTRL-Q | DC1 (X-ON, with XOFF to pause listings; ":okay to send") |
| 018 | 0010010 | 022 | 12 | CTRL-R | DC2 (Device Control 2, block-mode flow control) |
| 019 | 0010011 | 023 | 13 | CTRL-S | DC3 (X-OFF, with XON is TERM=18 flow control) |
| 020 | 0010100 | 024 | 14 | CTRL-T | DC4 (Device Control 4) |
| 021 | 0010101 | 025 | 15 | CTRL-U | NAK (Negative AcKnowlede) |
| 022 | 0010110 | 026 | 16 | CTRL-V | SYN (SYNchronous idle) |
| 023 | 0010111 | 027 | 17 | CTRL-W | ETB (End Transmission Block, not same as EOT) |
| 024 | 0011000 | 030 | 18 | CTRL-X | CAN (CANcel, MPE echoes) |
| 025 | 0011001 | 031 | 19 | CTRL-Y | EM (End of Medium, Control-Y interrupt) |
| 026 | 0011010 | 032 | 1A | CTRL-Z | SUB (Substitute) |
| 027 | 0011011 | 033 | 1B | CTRL-[| \e ESC (ESCApe, next character not echoed) |
| 028 | 0011100 | 034 | 1C | CTRL-\ | FS (File Separator) or IS4 (Info Separator 4) |
| 029 | 0011101 | 035 | 1D | CTRL-] | GS (Group Separator) or IS3 |
| 030 | 0011110 | 036 | 1E | CTRL-^ | RS (Record Separator, block-mode terminator) or IS2 |
| 031 | 0011111 | 037 | 1F | CTRL-_ | US (Unit Separator) or IS1 |
| 032 | 0100000 | 040 | 20 | | space. %20(+ in URLs) |
| 033 | 0100001 | 041 | 21 | ! | exclamation point, factorial, bang |
| 034 | 0100010 | 042 | 22 | " | double quote. "(XML too). \" |
| 035 | 0100011 | 043 | 23 | # | sharpsign, cross hatch, number sign. %23(bookmark in |
| 036 | 0100100 | 044 | 24 | \$ | dollar sign |
| 037 | 0100101 | 045 | 25 | % | percent sign. %25(escape in URLs) |

Table des caractères ASCII (III)

| | | | | | |
|-----|---------|-----|----|---|---|
| 038 | 0100110 | 046 | 26 | & | ampersand. &(XML too). %26(parameter delimiter in URLs) |
| 039 | 0100111 | 047 | 27 | ' | single quote, acute accent. &'(XML too) \' |
| 040 | 0101000 | 050 | 28 | (| left parenthesis |
| 041 | 0101001 | 051 | 29 |) | right parenthesis |
| 042 | 0101010 | 052 | 2A | * | asterisk, star, multiply |
| 043 | 0101011 | 053 | 2B | + | plus sign. |
| 044 | 0101100 | 054 | 2C | , | comma |
| 045 | 0101101 | 055 | 2D | - | hyphen, minus |
| 046 | 0101110 | 056 | 2E | . | period, full stop |
| 047 | 0101111 | 057 | 2F | / | slash, virgule, slant, forward slash, divide. %2F |
| 048 | 0110000 | 060 | 30 | 0 | |
| 049 | 0110001 | 061 | 31 | 1 | |
| 050 | 0110010 | 062 | 32 | 2 | |
| 051 | 0110011 | 063 | 33 | 3 | |
| 052 | 0110100 | 064 | 34 | 4 | |
| 053 | 0110101 | 065 | 35 | 5 | |
| 054 | 0110110 | 066 | 36 | 6 | |
| 055 | 0110111 | 067 | 37 | 7 | |
| 056 | 0111000 | 070 | 38 | 8 | |
| 057 | 0111001 | 071 | 39 | 9 | |
| 058 | 0111010 | 072 | 3A | : | colon |
| 059 | 0111011 | 073 | 3B | ; | semicolon |
| 060 | 0111100 | 074 | 3C | < | left angle bracket, less than. <(XML too) |
| 061 | 0111101 | 075 | 3D | = | equals sign |

Table des caractères ASCII

(IV)

| | | | | | |
|-----|---------|-----|----|---|--|
| 062 | 0111110 | 076 | 3E | > | right angle bracket, greater than. >(XML too) |
| 063 | 0111111 | 077 | 3F | ? | question mark. %3F(begin query string in URLs) |
| 064 | 1000000 | 100 | 40 | @ | "at" sign |
| 065 | 1000001 | 101 | 41 | A | 32 difference between UPPER and lower case. |
| 066 | 1000010 | 102 | 42 | B | |
| ... | | | | | |
| 090 | 1011010 | 132 | 5A | Z | |
| 091 | 1011011 | 133 | 5B | [| left square bracket |
| 092 | 1011100 | 134 | 5C | \ | backslash, reverse slant, reverse solidus |
| 093 | 1011101 | 135 | 5D |] | right square bracket |
| 094 | 1011110 | 136 | 5E | ^ | caret, circumflex. subs: ˆ ˆ ˆ |
| 095 | 1011111 | 137 | 5F | _ | underscores |
| 096 | 1100000 | 140 | 60 | ' | grave accent. Not in C. |
| 097 | 1100001 | 141 | 61 | a | |
| 098 | 1100010 | 142 | 62 | b | |
| ... | | | | | |
| 122 | 1111010 | 172 | 7A | z | |
| 123 | 1111011 | 173 | 7B | { | left curly brace |
| 124 | 1111100 | 174 | 7C | | vertical bar |
| 125 | 1111101 | 175 | 7D | } | right curly brace |
| 126 | 1111110 | 176 | 7E | ~ | tilde |
| 127 | 1111111 | 177 | 7F | | DEL (DELETE or rubout), cross-hatch box |

Table des caractères ASCII

(V)

- Propriétés intéressantes
 - ▶ Codable sur 7 bits
 - ▶ Il n'y a pas que des glyphes affichables ! Caractères protocolaires
 - ▶ Les chiffres se suivent : conversion caractère ↗ valeur facile (même propriété en EBCDIC)
 - ▶ Les lettres aussi... tri facile
 - ▶ Différence minuscules/majuscule = 32
 - ▶ DEL et les cartes perforées...
- Protocoles complexes construits autour ASCII : terminaux ANSI, Minitel...
 - ▶ Changements de couleurs, fontes, clignotement, inversion vidéo
 - ▶ Mouvement de curseurs
 - ▶ Extensions graphiques
- ⚡ aux fins de ligne des fichiers « textes »...
 - ▶ Unix : LF
 - ▶ MacOS : CR
 - ▶ Windows : CR + LF

Table des caractères ASCII

(VI)

Écrire portable, utiliser logiciels de conversion ou encodages
Emacs (suffixes -dos, -mac, -unix...)

- Utilisé comme base de majorité encodages
- Mais ∃ d'autres encodages
 - ▶ Telex : 5 bits
 - ▶ EBCDIC (IBM)
 - ▶ Ordinateurs Bull

Codage ISO-8859

(I)

- Extension à 8 bits de l'ASCII pour alphabets européens
- ISO-8859-1 (latin-1) pour le français... ou presque ☺ : manque les œ, Ÿ... (« au cœur de l'HAY-LES-ROSES »)
- ... corrigé en ISO-8859-15 (latin-9) qui inclut au passage l'€
- Tout loge dans un entier de 8 bits ☺

Codage UNICODE

(I)

- Projet ambitieux pour englober toutes langues terrestres (et autres : Seigneur des Anneaux, Klingon...)
- Radical codage en 32 bits (UTF-32)
- ∃ Sérialisation en codes plus petits avec optimisation caractères de base (guerres de religion...) + caractères échappement
 - ▶ UTF-16 : codage en 16 bits
 - ▶ UTF-8 : codage en 8 bits (de 1 caractère 8 bits à 4)
 - ▶ UTF-7 : codage en 7 bits
- + Variantes petit/grand-boutien...
- Surensemble d'ASCII ( pas d'ISO-8859)
- ~~> Rajout de caractères « larges » dans C99

Caractères de base

(I)

- **char** : entier d'au moins 8 bits
- Représente en général case mémoire la plus petite adressable directement par la machine
- Taille concrète en bit : CHAR_BIT
- Non spécifié si signé ou pas
- $\rightsquigarrow \exists$ explicitement en version signée (**signed char**) ou pas (**unsigned char**)
- Caractère utilisé comme unité de mesure de stockage via opérateur **sizeof()** donc par définition **sizeof(char)** vaut 1 ☺
`(gdb) print sizeof(long long int)
$1 = 8`
 $\rightsquigarrow \times \text{CHAR_BIT}$ pour taille en bittaille
- Limites réelles définie pour l'environnement dans <limits.h>

Caractères de base

(II)

```
1  /* Number of bits in a 'char'. */  
2  #define CHAR_BIT      8  
3  /* Minimum and maximum values a 'signed char' can hold. */  
4  #define SCHAR_MIN     (-128)  
5  #define SCHAR_MAX      127  
6  /* Maximum value an 'unsigned char' can hold. (Minimum is 0.) */  
7  #define UCHAR_MAX     255  
8  /* Minimum and maximum values a 'char' can hold. */  
9  #ifndef CHAR_UNSIGNED  
10 #define CHAR_MIN      0  
11 #define CHAR_MAX      UCHAR_MAX  
12 #else  
13 #define CHAR_MIN     SCHAR_MIN  
14 #define CHAR_MAX     SCHAR_MAX  
15 #endif
```



Nombres entiers

(I)

- 4 tailles d'entiers signés (par défaut)
 - ▶ **short**_{unsigned} **int** ou **short** (pour les flémards)
 - ▶ **int**
 - ▶ **long**_{unsigned} **int** ou **long** (pour les flémards)
 - ▶ **long**_{unsigned} **long**_{unsigned} **int** ou **long**_{unsigned} **long** (pour les flémards) en C99
(généralisation machines 64 bits)

On peut rajouter **signed** pour éclaircir

- 4 types non signés : rajouter **unsigned** au 4 précédents
- Relation directe entre dynamique et taille

Nombres entiers

(II)

```
1  /* Minimum and maximum values a 'signed short int' can hold */
2  #define SHRT_MIN      → (-32768)
3  #define SHRT_MAX      → 32767
4  /* Maximum value an 'unsigned short int' can hold. (Minimum is 0.) */
5  #define USHRT_MAX     → 65535
6  /* Minimum and maximum values a 'signed int' can hold */
7  #define INT_MIN       → (-INT_MAX - 1)
8  #define INT_MAX        → 2147483647
9  /* Maximum value an 'unsigned int' can hold. (Minimum is 0.) */
10 #define UINT_MAX      → 4294967295U
11 /* Minimum and maximum values a 'signed long int' can hold */
12 #if WORDSIZE == 64
13 #define LONG_MAX     → 9223372036854775807L
14 #else
15 #define LONG_MAX     → 2147483647L
16 #endif
17 #define LONG_MIN      → (-LONG_MAX - 1L)
18 /* Maximum value an 'unsigned long int' can hold. (Minimum is 0.) */
19 #if WORDSIZE == 64
20 #define ULONG_MAX    → 18446744073709551615UL
```

Nombres entiers

(III)

```
22 #else
23 #define ULONG_MAX—>4294967295UL
24 #endif
25 #ifdef __USE_ISOC99
26 /* Minimum and maximum values a 'signed long long int' can hold */
27 #define LLONG_MAX—>9223372036854775807LL
28 #define LLONG_MIN—>(-LLONG_MAX-1LL)
29 /* Maximum value an 'unsigned long long int' can hold. (Minimum is 0) */
30 #define ULLONG_MAX—>18446744073709551615ULL
31 #endif /* ISO_C99 */
```

- Débordements lors de calculs...
- Pas de contrôle lors de l'exécutions. **Sens de l'économie du C** : exécution rapide
- Mais peut aussi être subtilement utilisé (arithmétique modulo 2^n , signée ou pas...)
- **Sens de l'économie du C** : laisse programmeur libre ☺

Nombres entiers

(IV)

- Si grosse taille : calculs éventuellement plus long, stockage plus gros, temps de transfert processeur-disque/mémoire/réseau plus long...
- Choisir toujours au moins une taille suffisante pour stocker information (analyse de dynamique)
Nombre de bit utilisés pour coder $v \in \mathbb{N}^*$:

$$n_v = \lfloor \log_2 v \rfloor + 1$$

- Éventuellement compacter par changement de repère dans données algorithme



Types entiers étendus de taille connue

(I)

- Taille des entiers précédents dépendent machine & environnement cible
- Si besoin contrôle fin taille des données (éviter débordements, contrôleurs de périphérique, microcontrôleur) ? ☺
- Rajouts de types à taille fixe définis dans `<stdint.h>`
 - ▶ `int8_t`, `int16_t`, `int32_t`, `int64_t` : type signés avec exactement nombre de bits
 - ▶ `uint8_t`, `uint16_t`, `uint32_t`, `uint64_t` : idem non signé
- Taille au moins avec n bits : `int_least n _t` et `uint_least n _t`
- Concept intéressant : type le plus rapide et suffisamment gros (parfois, si plus gros, plus rapide) `int_fast n _t` et `uint_fast n _t`
- Pour tout les cas, limites disponibles `TYPE_MIN`, `TYPE_MAX`, `UTYPE_MAX`. Exemple :

Types entiers étendus de taille connue

(II)

```
1  /* Minimum of signed integral types having a minimum size . . . */
2  #define INT_LEAST8_MIN → (-128)
3  /* Maximum of signed integral types having a minimum size . . . */
4  #define INT_LEAST8_MAX → (127)
5  /* Maximum of unsigned integral types having a minimum size . . . */
6  #define UINT_LEAST8_MAX → (255)
```

Caractères larges

(I)

- Pour gérer des caractères étendus (UNICODE...) **char** trop petit
- Définition dans <wchar.h> de caractères étendus **wchar_t**
- Définition d'entiers de même taille **wint_t** pour éviter trop de conversions parfois



Booléens

(I)

- Traditionnellement en C, l'entier 0 valait « faux » et tout autre valeur signifiait « vrai »
- Rajout type `bool` dans C99 pour expliciter
-  Il existe de nombreux programmes qui définissaient déjà leur propre style `bool`... ↗ conflit !
- Pour éviter conflits probables, C99 introduit en fait type natif `_Bool`, moins probable
- Ensuite, définitions dans `<stdbool.h>` des types « programmeur ». Exemple dans GCC :

```
1 #define _bool_oooooooo_Bool  
2 #define _true_oooooooo1  
#define _false_oooo0
```

Énumérations

(1)

- Définition de constantes entières nommées

```
1 enum [type] { identifiant [= expression-const], ... } ;
```

- Par défaut commence à 0
- Ensuite par défaut, *constante précédente* + 1
- Exemples

```
1 enum e_record_type{  
2     CODER_RECORD,  
3     MATCHER_RECORD,  
4     COMPACT_RECORD  
5 };  
6 enum deal_with_base_pk_function{  
7     JOB_ADD_PK_IN_LIST=47,  
8     JOB_WRITE_A_PREROTED_SEARCH=53,  
9     JOB_WRITE_A_NEUTRAL_REF  
10};  
11 enum {LOG2_YAXIS_DEF=8}; // Type anonyme
```

Énumérations

(II)

- Avantage par rapport à une macro-constante du préprocesseur :
 - ▶ Évite de savoir compter de un en un ☺
 - ▶ Connue du débogueur : affichable, manipulable
 - ▶ Typage vérifiable par le compilateur

Astuce : transformation des **#define** en **enum** :

```
sed 's/#define[ \\\t][\ \t]*\([^\t\\\t ][^\\\t]*\)[ \\\t][ \\\t]*\(.*)$/enum { \1 = \2 };/'
```

Nombres flottants

(I)

- Besoin de représenter des valeurs plus « continues » et plus de dynamique que types entiers
- Sous-ensemble de $\mathbb{D} (\subset \mathbb{R})$
- Représentation souvent au format IEEE 754-1985

$$f = (-1)^S \times M \times 2^E$$

- ▶ S : bit de signe
- ▶ M : mantisse (entier positif)
- ▶ E : exposant
- Plusieurs tailles de flottants
 - ▶ Simple précision (**float**)
 - ▶ Double précision (**double**)
 - ▶ Précision étendue (**long double**)

Nombres flottants

(II)

- Codage en mémoire

| Format | Taille en bit | | | |
|--------|---------------|-------|-----------|-----------|
| | Total | Signe | Exposant | Mantisse |
| Simple | 32 | 1 | 8 | 24 |
| Double | 64 | 1 | 11 | 53 |
| Étendu | ≥ 80 | 1 | ≥ 15 | ≥ 64 |

- $1 + 8 + 24 = 33 \neq 32$: voir plus tard...
- Exposant codé en biaisé : $\mathcal{E}_{\text{réel}} = \mathcal{E}_{\text{stocké}} - \mathcal{E}_{\text{biais}}$
- Tri lexicographique sur bits compatible avec tri flottant ! Même si on ne gère pas le flottant on sait trier ☺

| Format | Minimum en dénormalisé | Minimum en normalisé | Maximum fini | 2^{-N} (grain) | Chiffres significatifs |
|--------|-----------------------------|-----------------------------|----------------------------|----------------------------|------------------------|
| Simple | $1,4 \cdot 10^{-45}$ | $1,2 \cdot 10^{-38}$ | $3,4 \cdot 10^{38}$ | $5,96 \cdot 10^{-8}$ | 6–9 |
| Double | $4,9 \cdot 10^{-324}$ | $2,2 \cdot 10^{-308}$ | $1,8 \cdot 10^{308}$ | $1,11 \cdot 10^{-16}$ | 15–17 |
| Étendu | $\leq 3,6 \cdot 10^{-4951}$ | $\leq 3,4 \cdot 10^{-4932}$ | $\geq 1,2 \cdot 10^{4932}$ | $\leq 5,42 \cdot 10^{-20}$ | $\geq 18\text{--}21$ |

Nombreux paramètres définis dans `<float.h>`

Nombres flottants

(III)

- Possibilité de déclencher exceptions (division par 0, débordement,...) (fonction exécutée sur événement)
- Rajout de quantités symboliques (déclarées dans `<math.h>`)
 - ▶ $+0$ et -0 . Néanmoins $+0 = -0$ est vrai
 - ▶ $+\infty$ et $-\infty$ (par exemple $\frac{1}{+0}$ et $\frac{1}{-0}$) (`HUGE_VAL...`)
 - ▶ NaN (*Not a Number*) pour $\frac{0}{0}$ ou $\sqrt{-1}$
Seul cas où $x \neq x$ lorsque x vaut NaN. Existe en signé et en version déclenchant exception (SNaN)
- ~ Peuvent simplifier programmation et calcul si bien géré (éviter tests cas particuliers...)
- \exists Nombreux choix d'arrondi (plus proche, +, -, vers 0,...)
- <http://grouper.ieee.org/groups/754> En cours de révision
http://en.wikipedia.org/wiki/IEEE_754r si vous voulez participer ☺



Plein de subtilités



Conversion flottants→entiers à l'arrache (I)

- En temps normal `int_oui=(int)_uf` doit faire le travail
- \exists nombreuses fonctions de conversion et d'arrondi dans la bibliothèque mathématique (`rint()`, `round()`...)

La bibliothèque du C permet de le faire tout seul mais pas toujours disponible (cf. microcontrôleur Coupe de Robotique 2008). Pour hackers :

```
1 int_uftoi(float_uf)
2 {
3     // Récupère les bits du flottant dans un entier :
4     uint32_t_dwu=((uint32_t_*)&f);
5     // La valeur spéciale où tous les bits sont à 0 code 0 :
6     if_(dwu==0)
7         return_0;
8     // Récupère l'exposant codé sur 8 bits et compense le biais :
9     char_expu=(dwu>>23)-127; // Suppose un char de 8 bits
10    if_(expu<0||expu>23)
11        /* Si l'exposant est négatif, de toute manière, le no-
```

Conversion flottants→entiers à l'arrache

(II)

```
12     .....moins que 1, donc arrondis à 0. Si c'est supérieur à 23,  
13     .....est supérieur à 2^{24} en non décide de le jeter et de r  
14     .....arbitrairement 0. Mmm... On pourrait gérer jusqu'à 2^{24}  
15     .....faudrait corriger le code ci-après */  
16     .....return 0;  
17     /* Construit le nombre avec le 1 de poids fort qui est économisé  
18     .....la norme IEEE-754, puis les 23 autres bits de la mantisse  
19     .....en fonction de l'exposant : */  
20     int val = (1<<exp) + ((dw&0x7FFFFFF)>>(23-exp));  
21     // En fonction du bit de signe, inverse le résultat :  
22     if (dw&0x80000000)  
23         return -val;  
24     else  
25         return val;  
26 }
```

Bon, évidemment, ceci ne gère pas toute la norme, le dénormalisé, les infinis, etc.

Nombres flottants \neq réels !

(I)

« What Every Computer Scientist Should Know About Floating-Point Arithmetic », David GOLDBERG, Computing Surveys, mars 1991, ACM

- Nombres flottants \equiv pale imitation de \mathbb{R} et même de \mathbb{D} ☺
- Nombreuses approximations
-  Propriétés algébriques de \mathbb{R} non vérifiées : non associatif

$$(1 \oplus 10^{40}) \ominus 10^{40} = 0$$

$$1 \oplus (10^{40} \ominus 10^{40}) = 1$$

- Changement des résultats possibles selon optimisations... ☺
- Notion d'équivalence séquentielle de programme entre différentes versions

Nombres flottants \neq réels !

(II)

- ▶ Forte : le programme obtenu donne le même résultat
- ▶ Faible : le programme obtenu donne le même résultat modulo les problèmes numériques précédents
- Choisir programmation prenant en compte ces caractéristiques
 - ▶ T_EX écrit en virgule fixe 16+16 bits pour portabilité multi-plateforme
☺
 - ▶ Compromis entre performances & précision
-  Compilateurs devraient en tenir compte (pas optimisations sauvages)
- Exemples
 - ▶ $(x - y)(x + y)$ plus précis (voire plus rapide) que $x^2 - y^2$
 - ▶ Algorithme somme de flottants

Algorithme de sommation de flottants (I)

- Solution triviale

```
1 double x[N];  
2 double s = 0;  
3 for(int i = 0; i < N; i++)  
4     s += x[i];
```

$$s = \sum_{i=0}^{N-1} x_i(1 + \delta_i)$$

avec $|\delta_i| < (N - i)\epsilon$

Algorithme de sommation de flottants

(II)

- Version KAHAN

```
1 double x[N];
2 double s = x[0];
3 double c = 0;           // Erreur d'arrondi
4 for(int i = 1; i < N; i++) {
    double y = x[i] - c; // Compense erreur précédente
    double t = s + y;   // Nouvelle somme
    c = (t - s) - y;   // Estime l'erreur arrondi
8 s = t;
}
```

$$s = \sum_{i=0}^{N-1} x_i(1 + \delta_i) + \mathcal{O}(N\epsilon^2 \sum_{i=0}^{N-1} |x_i|) \quad \text{avec} \quad |\delta_i| \leq 2\epsilon$$

Optimisations incontrôlées du programme fait des ravages ici...
car revient à algorithme trivial! 😊

Vers des nombres flottants normalisés

(I)

- Possible de représenter des nombres de plusieurs manières

$$\mathcal{M}' = 2^{-a} \mathcal{M}$$

$$\mathcal{E}' = \mathcal{E} + a$$

- Problème des codages redondants : comparaisons difficiles ☺
 - Idée 1 : normaliser ! Exemple : choisir le \mathcal{M} le plus grand pouvant loger dans les bits alloués pour la mantisse
 - Idée 2
 - ▶ $\forall \mathcal{M} \neq 0$: commence toujours par 1 en binaire
 - ▶ ↗ Ne pas stocker ce 1 évident...
- ↗ Flottant normalisé : gagne 1 bit de précision pour la mantisse ! ☺

Pourquoi des nombres flottants dénormalisés ? (I)

- Soustraction de 2 nombres normalisés, par exemple en simple précision

$$a = 2,05 \cdot 10^{-37}$$

$$b = 2,03 \cdot 10^{-37}$$

$$a - b = 2 \cdot 10^{-39}$$

$$a \ominus b = 0$$

$$a \neq b$$

Seule solution car M ne peut pas commencer par 1... ☺

- Idée : rajouter mode dénormalisé pour très petits nombres où M peut ne pas commencer par un 1
- Permet *underflow* (dépassement de capacité par le bas) progressif

Pourquoi des nombres flottants dénormalisés ? (II)

-  Si flottant dénormalisé non géré directement en matériel : génère exception et calculs terminés par... système d'exploitation ↗ performances ↴ ☺
-  Parfois autorisation exception \implies suppression pipeline (DEC Alpha) ☺

Dénormalisation flottante

(1)

- Parfois exception IEEE-754 générée lors de la dénormalisation
- Typiquement un programme de différences finie avec un domaine avec de petites valeur ϵ entouré de 0 :

$$x_{i,j}^n = \frac{x_{i-1,j}^v + x_{i+1,j}^v + x_{i,j-1}^v + x_{i,j+1}^v}{4}$$

~ Propagation d'ondes de dénormalisation

| | | | | | | | |
|---|---|------------|------------|------------|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | ϵ | ϵ | ϵ | 0 | 0 | 0 |
| 0 | 0 | ϵ | ϵ | ϵ | 0 | 0 | 0 |
| 0 | 0 | ϵ | ϵ | ϵ | 0 | 0 | 0 |
| 0 | 0 | ϵ | ϵ | ϵ | 0 | 0 | 0 |
| 0 | 0 | ϵ | ϵ | ϵ | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |



| | | | | | | | |
|---|---|---|------------|------------|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | d | d | d | d | 0 | 0 |
| 0 | d | d | d | d | d | 0 | 0 |
| 0 | d | d | d | ϵ | d | d | 0 |
| 0 | d | d | ϵ | d | d | 0 | 0 |
| 0 | d | d | d | d | d | 0 | 0 |
| 0 | 0 | d | d | d | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |



| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | d | d | d | 0 | 0 | 0 |
| 0 | d | d | d | d | d | 0 | 0 |
| d | d | d | d | d | d | d | 0 |
| d | d | d | d | d | d | d | d |
| d | d | d | d | d | d | d | d |
| d | d | d | d | d | d | d | d |
| 0 | d | d | d | d | d | 0 | 0 |
| 0 | 0 | d | d | d | 0 | 0 | 0 |

Dénormalisation flottante

(II)

- À pleurer sur machine parallèle si ordonnancement statique : tous les processeurs attendent le plus lent ! 😞
- Rajout d'un biais pour ne plus être au voisinage de 0. Mais perte de dynamique... Compromis

$$y_{i,j}^n = x_{i,j}^n + b \quad (1)$$

$$y_{i,j}^n = \frac{y_{i-1,j}^\nu + y_{i+1,j}^\nu + y_{i,j-1}^\nu + y_{i,j+1}^\nu}{4} \quad (2)$$

- Bonne nouvelle : GPU gèrent les nombres dénormalisés en matériel sans pénalité

Nombres complexes

(I)

- Rajout en C99 des flottants en version complexe C (en fait sous-ensemble de $\mathbb{D} + i\mathbb{D}$)
- Rejoint classe langages scientifiques (Fortran)
- (**float** | **[long]** | **double**) **complex**
- **double** **complex**
- **long double** **complex**
- Pas d'entiers en version complexe (mais extension GCC)
- Comme pour type **bool** pour éviter conflit avec vieilles déclaration propres, via `<complex.h>`

```
1 #define _Complex→————→_Complex
2 #define _Complex_I————→(_extension__1.0iF)
#	define _I _Complex_I
```

- Stockés en mémoire comme concaténation partie réelle et imaginaire
- Mais vraie opération interne
- Type **_Imaginary** et **imaginary** aussi

Pointeurs

(I)

⚠ LA subtilité du langage C ! ⚠

- Variable spéciale pour contenir adresse mémoire d'un objet

```
1 type□* identifiant;
```

définit *identifiant* comme variable de type pointeur vers *type*

- Profondeur arbitraire : on peut aussi manipuler adresse de cette variable

```
1 char□*** v;
```

v est une variable de type « pointeur vers une variable de type pointeur vers une variable de type pointeur vers une variable de type **char** »

- Corollaire : impose/implique en général objet pointé en mémoire et non en registre (⚠ optimisation ☺)

Pointeurs

(II)

- ⚠ Déclarer une variable de type pointeur alloue la place pour l'adresse, ne déclare pas objet pointé ni alloue place mémoire pour objet pointé ↗ erreur de débutant... ☺

<http://cslibrary.stanford.edu/104> Les pointeurs en pâte à modeler
☺



Vecteurs

(I)

- Suite ordonnée d'objets de même type
- Déclaration du style

1 *type* \sqcup *identifiant* [*dim₁*] [*dim₂*] \cdots [*dim_n*] ;

- Premier élément commence à 0 dans chaque dimension (à 1 en Fortran par défaut...), donc espace de définition

$$\mathcal{D}_{\text{identifiant}} = [0, \dim_1 - 1] \times [0, \dim_2 - 1] \times \cdots \times [0, \dim_n - 1] \cap \mathbb{N}^n$$

- Identifiant assimilé à un pointeur vers premier élément



Vecteurs

(II)

- Stockage contigu en mémoire. En multidimensionnel : varie plus lentement sur dernière dimension (contraire de Fortran).
Adresse élément

$$a_{i_1, i_2, \dots, i_n} = a_0 +$$

$$\left(((\cdots (i_1 \dim_2 + i_2) \cdots) \dim_{n-2} + i_{n-1}) \dim_n + i_n \right) \times \text{sizeof(type)}$$

```
1 float a[2][3]
```

rangé en mémoire :

| | | | | | |
|---------|---------|---------|---------|---------|---------|
| a[0][0] | a[0][1] | a[0][2] | a[1][0] | a[1][1] | a[1][2] |
|---------|---------|---------|---------|---------|---------|

- Important si mélange plusieurs langages ou optimisations

Vecteurs

(III)

- En C99 taille de tableau dynamique autorisée (comme en Fortran)

```
1 int  f(int  n){  
2     double  matrice[n*2][n*3];  
3     ...  
4 }
```

- ▶ C99 simplifie énormément programmation C
- ▶ Programmes plus propres
- ▶ Allocation dans un bloc (→ dans la pile)
- ▶ Alternative intéressante à `alloca()`

Vecteur en paramètre de fonction

(I)

- Vecteur passé en tant qu'adresse sur premier élément (identifiant vu précédemment)
- Comme n'importe quel paramètre en C : passage par copie... d'une adresse ici !
 - ▶ ☕ Mais à partir de cette adresse, accès aux éléments ↗ on peut modifier éléments d'un vecteur ! ⚡

```
1 void affiche(double v[]){
2     ...
3 }
4 ...
5 affiche(v);
```

- ▶ ⚡ Astuce : si on veut passer tous les éléments par copie, mettre vecteur dans une structure
- Comme en mono-dimensionnel simple suite d'éléments `double v[]` ou `double *v` font l'affaire

Vecteur en paramètre de fonction

(II)

- Possibilité en C99 de préciser au compilateur que vecteur passé existe en mémoire et a au moins cette taille (vérification anti-débordement)

```
1 void affiche(double v[static N]) {  
2     ...  
}
```

- En multidimensionnel, calcul d'adresse élément dépend taille dimension $\geq 2\dots \rightsquigarrow$ Obliger de passer tailles dimensions concernées (déjà en Fortran, nouveau en C99)

```
1 void affiche(double matrice [X] [Y]) {  
    ...  
}
```

- Nouveau aussi en C99, tailles variables & passables en paramètres (idem Fortran)

Vecteur en paramètre de fonction

(III)

```
1 void affiche(int x, int y, double matrice[x][y]) {
2     ...
3 }
```

Chaîne de caractères

(I)

- Interaction avec monde extérieur ↗ besoin de chaînes de caractères
- Sens de l'économie du C : ne pas rajouter type spécial dans langage (\neq string en Basic ou Java)
- ↗ Besoin d'une « convention collective »
 - ▶ Chaîne en C = vecteur de caractères
 - ▶ Pas de stockage de la taille contrairement à Fortran, BASIC, Java, PERL, Python...
 - ▶ Fin de chaîne = caractère nul ('\0')
 - ▶ Constructeur de chaîne constante dans le langage qui met le '\0' final



Chaîne de caractères

(II)

```
1 char chaine [] = "rien";
```

Alloue en mémoire les caractères



rien est de type tableau ou pointeur vers caractère

- ⚠ Bien comprendre cette convention !!!

- ▶ Si on copie une chaîne dans une autre, vérifier qu'il y a de la place, *y compris pour le '\0' final!*
Sinon : écrabouillage d'autre données... ☺
- ▶ !!! Ne pas oublier le '\0' final!!!
Sinon
 - Caractères à suivre inclus dans chaîne jusqu'au prochain '\0' (fuite d'information confidentielle) ☺
 - Caractères à suivre dans une zone mémoire inexistante (erreur mémoire ↗ fin de programme) ☺

Chaîne de caractères

(III)

- Sens de l'économie du C : pas de vérification de type, de taille de chaîne,...
~~>    Erreurs de programmation sources principales de piratages en C ☺
- Penser à la fonction standard de copie avec allocation mémoire `strdup()` (fonction la plus importante du langage C ! ☺)

Chaînes de caractères larges

(I)

- Internationalisation et caractères larges
- Constantes Larges

```
1 wchar_t grosse_chaine[] = L"très_gros"
```

- Pour communiquer avec monde extérieur, caractères larges (exemple UNICODE) codables en suite (entière) de caractères simples (exemple UTF-8)

►  Dans comités de normalisation, définir des sérialisations compatibles avec formats chaînes de caractère de *tous* (?) langages de programmation
~~ caractères UTF-8 autres que caractère nul ne contient pas de '\0'

► Choisir des sérialisations qui respecte ordre lexicographique de comparaison des caractères : possible de trier avec même résultat qu'UNICODE caractères larges avec fonction sur chaînes de caractères simples codage UTF-8

Structures de données

(I)

- Regroupement local séquentiel d'objets C hétérogènes
- Déclaration d'objets

```
1 struct {
2     double x;
3     double y;
4     int couleur;
5 } p;
```

Pénible si on veut déclarer de nombreuses variables de ce type... ☺

Structures de données

(II)

- Déclaration préalable de type

```
1 struct search_list{  
2     ut_search_buffer search_buffer;  
3     char *key;  
4     struct search_list *next;  
5 };
```

puis déclaration de variables de ce type

```
1 struct search_list *s;  
2 struct search_list search_list;  
3 search_list.key = "base";  
4 search_list.next = NULL;
```

- Espace de nommage des types structures et de nommage des variables distinct ↗ pas de conflit possible
- Possible de déclarer d'un coup 1 type et 1 variable
- Dans C99 possible de faire des structures non finies en terminant par vecteur de taille indéfinie

Structures de données

(III)

```
1 struct vecteur{  
2     size_t taille;  
3     double contenu[];  
4 }
```

Simplifie structures de données (pas obligé de passer par objet supplémentaire avec pointeur dessus)

-  Allocation concrète dans la machine non spécifiée...
 - ▶ Pour des raisons de limitations matérielles ou optimisation on peut utiliser plus de mémoire que somme de taille des champs
 - ▶ ∃ des extensions avec pragmas d'alignement (GCC) comme
- ```
1 struct S { short f[3]; } __attribute__((aligned(8)));
```
- ▶ Néanmoins ordre respecté en mémoire et si début commun dans 2 structures, stockage mémoire du début identique (utile pour unions)

# Type d'union de types

(I)

- Besoin de pouvoir stocker différents types d'objets dans une variable
- Contrairement aux structures, champs exclusifs

1    **union**  $\llbracket$  *nom-union*  $\rrbracket \{ \llbracket$  *type champ* ;  $\rrbracket \}^+$   $\rrbracket$  *identifiant* ;

-  Éviter suppositions de stockage (stockage des entiers par gros bout ou petit bout), de compactage, d'optimisation...

# Type d'union de types

(II)

```
1 enum stockage {Entier, Flottant};
2 struct element {
3 enum stockage type;
4 union {
5 int i;
6 double d;
7 } valeur; // Facultatif
8 } e;
9 e.type = Entier;
10 e.valeur.i = 3;
11 e.type = Flottant;
12 e.valeur.d = -3.87e-7;
```

On aurait pu aussi supprimer `valeur` dans structure car pas d'ambiguïté sur `i` ou `d`



# Type d'union de types

(III)

- Peut être utilisé pour récupérer/traduire/(dé)sérialiser/convertir données, protocoles,...

On écrit avec un type, on lit avec un autre



Bien mettre des gardes testant machine cible car programme non portable...

# Champs de bits

(1)

- Champ de Structure

- ▶ Stockage plus compact taillé au bit près
- ▶ Permet de cadrer des spécificités matérielles (périphériques, registres de contrôle...)

Dans /usr/src/linux-2.6.13/drivers/net/sundance.c

```
1 struct netdev_private{
2 // ...
3 unsigned int flowctrl :1;
4 unsigned int default_port:4; /* Last dev->if_port value */
5 unsigned int an_enable :1;
6 /* ... */
}
```

- Nécessite généralement plus de temps d'accès que le type natif arrondi au dessus en taille ↗ compromis espace-temps
- Pas type natif ⇒ pas d'adresse spécifique donc pas de pointeur possible sur un champ de bits

# Champs de bits

(II)

- Taille limitée à celle d'un entier
- $\exists$  extensions de C permettant champs de bits hors structure (synthèse matérielle...)



# Autres types standards

(I)

- Il existe de nombreux types définis dans bibliothèques standard du C
- Voir les manuels (`man`, `woman` sous Emacs...), `info` (`C-h i` sous Emacs...)
- `size_t` pour définir des tailles (style `sizeof()`)
- `ptrdiff_t` pour différences de pointeurs
- `fpos_t` pour position dans fichier
- ...

# Types de fonctions

(I)

- Nécessité en C de déclarer tout objet avant utilisation, y compris fonction
- Compilation en unités séparées (bibliothèques...) ou récursion : savoir appeler une fonction sans avoir sa définition complète
- ↗ Types de fonction : signature d'appel

```
1 void affiche2(int x, int y, double matrice[x][y]);
2 void affiche3(int , int , double *);
```

Première déclaration plus explicite à préférer

- `affiche2` pointe vers une adresse de fonction définie ailleurs
- Déclaration type de fonction avec paramètres optionnels

```
1 extern int printf(const char * restrict __format, ...);
```



# Types de fonctions

(II)

`printf` est une fonction définie dans une autre unité de compilation (**extern**) et renvoie un **int**. La fonction prend une chaîne de caractère de type format, s'engage à ne pas la modifier (**const**) et il n'y aura pas d'aliasing dessus en local (**restrict**). Ensuite, viennent paramètres optionnels

# Types incomplets

(I)

- Vecteur dont première dimension non dimensionnée
- Déclaration du seul nom de structure : type opaque de bibliothèque cachée au programmeur. Dans (pseudo)X11

```
1 struct _XDisplay ;
2 ...
3 struct _XDisplay *d = OpenDisplay(...);
4 ...
5 Dessine(d, ...);
```

Programme utilisateur peut récupérer et passer en paramètre `d` qui pointe vers donnée *opaque*

- Pointeur encore plus opaque

```
1 void *p;
```

# Types incomplets

(II)

Utiliser pour pointer vers n'importe quoi

⚠ Sens de l'économie du C : doit être transformé en autre type de pointeur avant déréférencement car mémoire pas typée en C (contrairement au LISP, Python, PERL...)

- Déclaration de procédures et/ou de fonctions sans paramètre

```
1 /* Ne renvoie rien : une procédure en fait */
2 void
3 begin_timer (void /* Pas d'argument */)
4 {
5 gettimeofday (&time_begin , NULL);
6 }
```

- Ne précise pas les arguments

```
1 int une_vague_fonction();
```

# FAQ Composition opérateurs de déclaration (I)

- Compliqué en C car syntaxe trop obscure sur ce point, mélange d'opérateurs préfixés, postfixés, de priorité... ☺
- Avec langage C parallèle MPL (plural en plus...) venait outil transformant description de type en anglais en type C/MPL ☺
- Matériaux de base
  - ▶ \* déclare pointeurs
  - ▶ [] déclare vecteurs/matrices
  - ▶ () déclare une fonction
- Comment déclarer tableau de pointeur ?

1    **char**  $\text{u}^*\text{u}$  **argv** [] ;

typiquement tableau de chaînes de caractères

- Comment déclarer fonction qui renvoie pointeur ?

1    **void**  $\text{u}^*\text{u}$  **malloc**  $\text{u}(\text{size\_t}\text{u}$  **size**) ;

# FAQ Composition opérateurs de déclaration (II)

- ↗ Opérateurs [] et () prioritaires sur opérateur \*
- Sens de l'économie du C : utiliser parenthèses pour changer priorité ! ↗ Aux dépens des neurones du programmeur... ☺
  - ▶ Comment déclarer pointeur de fonction (utile pour faire délégation, programmation objet en... C ☺) ?

1 **int**  $\cup$   $(\ast \cup p)$  ()

rapproche « pointeur » de p

- ▶ Comment déclarer pointeur vers matrice de double ?

1 **double**  $\cup$   $(\ast \cup p\_matrice)$  [N] [M]

- ▶ Idée générale : partir de la variable et appliquer opérateurs pour raffiner type en rajoutant parenthèses si besoin est

■ « Pointeur de fonction qui renvoie un pointeur vers un **int** et prend en paramètre un vecteur de **double** et un pointeur de fonction qui renvoie un pointeur vers un **float** et prend en paramètre un pointeur vers un tableau de **char** »



# FAQ Composition opérateurs de déclaration (III)

```
1 *p
2 (*p)()
3 int*(*p)()
4 int*(*p)(double[] , float*(*)(char(*)[]));
```

- ▶ Devient rapidement cryptique... ☺ ↗ passer par type intermédiaires (**typedef**)
- Pas possible de combiner vecteurs et fonctions directement (pas de vecteur de fonction, de fonction de fonction, de fonction qui renvoie un vecteur,...)
- Mais comme on peut passer par des pointeurs... ☺
- Programmation par essai-erreur peut aider ☺



# Le plan

- 1 Introduction
  - Bibliographie
  - Petite histoire
- 2 Langage
  - Généralités
- 3 Un ordinateur ? Mais qu'est-ce ?
  - Les bases de la numérologie
  - Calcul binaire
- 4 Déclarations
  - Généralités
  - Types des objets
  - Portée
- 5 Expressions
  - Sémantique opérateurs
  - Coercition de type
- 6 Instructions
- 7 Préprocesseur
- 8 Bibliothèques
- 9 Éditeur exemple d'Emacs
  - Tuning de son clavier
- 10 Compilation
- 11 Modularisation
- 12 Mise au point
  - Dévermineur (debugger)
  - Analyse mémoire
- 13 Extensions
  - Du C pour le graphisme
  - C++
- 14 Délires
  - Le bêtisier
  - Concours de programmes incompréhensibles
- 15 Conclusion
- 16 Index
- 17 Table des matières

# Portée des déclarations

(I)

- Portée (scope) : domaine d'accessibilité des objets
- Besoin d'avoir variables locales
  - ▶ Programmation plus claire
  - ▶ Moins de conflits dans gros programmes

```
1 for (int i=0; ...) {
2 ...
3 ...
4 ...
 }
```



si oubli de redéclarer variable locale à boucle interne...

- ▶ Récursion

# Portée des déclarations

(II)

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 double fibonacci(double n){
5 return n <= 1 ? 1 : fibonacci(n - 1) + fibonacci(n - 2);
6 }
7
8 int main(int argc, const char *argv[static 2]){
9 double v = atof(argv[1]);
10 printf("Fibonacci(%f) = %f\n", v, fibonacci(v));
11 return 0;
12 }
```

À utiliser pour de meilleures causes (complexité en  $\mathcal{O}(2^n)$  ☺ ici alors que mathématiquement puissance de matrice après diagonalisation matrice de suite en  $\mathcal{O}(\log n)$ ...)

- Principe de base : objet déclaré dans un bloc visible dans ce bloc et bloc imbriqué
  - ▶ Pas visible depuis bloc extérieur

# Portée des déclarations

(III)

- ▶ Masquable par déclaration d'un identifiant de même nom dans un bloc imbriqué

```
1 int i=6;
2 {
3 // i vaut 6 ici
4 {
5 double i=9;
6 // i vaut 9.0 ici
7 }
8 // i vaut de nouveau 6
9 }
```

- ▶ Corollaire : tout objet déclaré en dehors d'un bloc (donc les fonctions) est global au programme

- Tout objet doit être déclaré en C avant usage
- Si utilisation dans plusieurs fichiers, déclaration ( $\approx$  identique...) dans plusieurs fichiers
- Mais où se trouve la *vraie* version ? Où est le stockage de la variable ?

# Portée des déclarations

(IV)

- Rajout mot-clé **extern** devant déclaration pour dire que déclaration de ce fichier n'est pas la « vraie ». Code d'erreur fonctions UNIX  
`<errno.h>`

```
1 extern int errno;
```

- ▶  Gênant dans gros programmes : risque de conflits ! ☺( ↵ espaces de nommage en C++)
- ↵ Déclaration d'objets locaux à l'unité de compilation (fichier) en rajoutant mot-clé **static**
  - ▶ Sens de l'économie du C : réutilisation de **static** pour un autre usage que son sens premier... ☺ Syntaxe cryptique 

# Où déclarer les variables

(I)

- Différentes religions
  - ▶ Présentation + centralisée des variables ☺
  - ▶ Si on a besoin d'utiliser valeur existante quelque part : on l'utilise tout simplement ☺
  - ▶ Pollue espace de nommage ☺
- Déclaration au plus tard et rajout de blocs pour limiter portée
  - ▶ Évite de polluer espace de nommage ☺
  - ▶ Moins de visibilité en début de fonction sur ce qui est manipulé ☺
  - ▶ Si modification de programme et besoin valeur variable en dehors de sa visibilité : remonter déclaration variable au niveau bloc concerné. ↗   si masquage dans blocs plus internes...
  - ▶  Éviter néanmoins de jouer sur les masquage pour santé mentale des relecteurs...

# Vie réelle des objets

(I)

- Portée syntaxique : ce que programmeur veut...
- Programme compilé et optimisé
  - ▶ Si possible garder objets souvent dans registres processeurs (jamais assez nombreux ☺)
  - ▶ Possible de conseiller compilateur de garder variable souvent utilisée en registre en rajoutant **register** dans déclaration
  - ▶ Un même registre va stocker plusieurs objets différents et de types différents au cours de l'exécution
  - ▶ Compilateur essaye d'optimiser en utilisant graphe de dépendance, analyses de durée de vie, heuristiques...
  - ▶ ↗ Optimiser tout en préservant sémantique du programme
  - ▶ ⚠ Portée syntaxique ≠ différente
  - ▶ ⚠ ⚠ ⚠ Si déverminage (*debug*) du code, acte de dieu (non prévu par compilateur...) qui manipule variables du programme avec portée syntaxique et non réelle ↗ Pas forcément bonnes valeurs affichées !

# Vie réelle des objets

(II)

- ▶ Possible de préciser qu'une variable peut avoir son état changé en mémoire (périphérique matériel, DMA, programmation multithread, multi-processus avec segment mémoire partagée...) avec **volatile** rajouté dans déclaration ↵ synchronisation fine entre état interne (registres & caches) et mémoire...  Performances...

# Vie réelle des objets

(III)

- Variables dans un bloc et paramètres de fonction
  - ▶ Existence interne au bloc
  - ▶ Automatiquement allouées dans « pile » (attribut par défaut **auto**)
    - taille de pile parfois limitée ↗ erreur de segmentation en cas de dépassement (`man getrlimit` et shell)
    - Pile très spéciale dans certains micro-contrôleurs...
    - Possible d'avoir variables statiques (non automatique) dans fonctions Permet de garder état d'un appel à l'autre

```
1 int jolie_function(...){
2 static int nombre_de_jolis_appels=0;
3 ...
4 nombre_de_jolis_appels++
5 ...
6 }
```

Initialisation exécutée qu'une seule fois (heureux !)  
↗ En tenir compte en cas de récursivité

# Vie réelle des objets

(IV)

- ▶   Ne pas utiliser d'adresse vers objets automatiques en dehors de leur bloc de visibilité : contenu mémoire indéfini ↗ bugs sordides (si interruption...)
- ∃ Allocation dynamique permanente (cf. `malloc()` et autres)
- Extension C99 : *Thread-local storage* (TLS) pour avoir une instance par *thread* dans un programme parallèle

```
1 --threaduintui;
```

Cf discussion

<http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1364.htm>

[http://en.wikipedia.org/wiki/Thread-local\\_storage](http://en.wikipedia.org/wiki/Thread-local_storage)

-  Souvent sujet chaud des langages (comparer en PERL **my** et **local** !)

# Objets non modifiables

(I)

- Qualificatif **const** dans déclaration

```
1 int const taille = 100;
```

À utiliser de préférence à des car entité de première classe du langage (connue par débogueur...)

-  Subtilité avec pointeurs : est-ce pointeur ou ce qui est pointé qui est constant ?

► Avant \* (éloigné identifiant pointeur) : ce qui est pointé est constant

```
1 const char * message_bienvenue = "Bonjour !";
```

► Après \* (proche identifiant pointeur) : pointeur constant

```
1 double * const vecteur_standard = le_vecteur;
```

Rien n'empêche le contenu de `le_vecteur` d'être modifié

# Objets non modifiables

(II)

-  Néanmoins combinable avec **volatile** : pas modifié par ce programme mais monde extérieur (variables d'environnement, tic d'horloge...)

```
1 extern const volatile long int tic _hardware _counter;
```

# Des pointeurs à problèmes...

(I)

- Problème d'accès aux données par pointeur ou directement en mémoire : difficile pour compilateur de comprendre ce qui se passe (et pour programmeur ? ☺)
- Phénomène d'*aliasing* : 2 pointeurs différents peuvent concerner une zone mémoire commune...
-  Quid si pour optimiser, objet pointé mis en registre et modification par ailleurs objet en mémoire via pointeur ?
- Contourner les problèmes ?
  - ▶ Java : pas de pointeurs explicites
  - ▶ Fortran : alias interdits par la norme

# Exemple de C : pointeurs

(I)

- Compliquent (empêchent ?) analyses automatiques
- Donc moins d'optimisations

```
1 p = &a;
2 if (b > 0)
3 p = &b;
4 *p = 5;
```

Lequel de a ou b vaudra 5 ?

Peut être optimisé en

```
1 if (b > 0)
2 b = 5;
3 else
4 a = 5;
```

si compilateur « comprend » les pointeurs...



# Exemple de C : aliasing

(I)

Deux références (pointeurs) sur une même zone

```
1 void init(int size, float array[size], float* val)
2 {
3 float* end = array + size;
4 while(array < end) array ++ = *val + 1.0;
}
```

- Stockage de l'*adresse* de **val** en registre, et recharge à chaque tour car éventuellement dans **array** !  
Quid si **val**  =  &array[3] ?
- Parade ? **const** ? **register** explicite ?
- Pas toujours possible...
- Rajouter un test et écrire 2 codes différents :

# Exemple de C : aliasing

(II)

```
1 void init(int size, float array[size], float *val)
2 {
3 float *end = array + size;
4 float v = *val + 1.0;
5 if (val >= array && val < end) {
6 // Cas à problème :
7 while (array <= val) *array++ = v;
8 v += 1.0;
9 while (array < end) *array++ = v;
10 }
11 else {
12 // Pas de problème :
13 while (array < end) *array++ = v;
14 }
15 }
```

- Solution si on sait que ce cas n'arrivera jamais : le dire au compilateur ↗ pointeurs restreints

# Pointeurs restreint

(I)

- Nouveauté de C99
- Garantir au compilateur qu'il n'y aura pas d'*aliasing* : `restrict`
- Exemple précédent

```
1 void init(int size,
2 float array[size],
3 float * restrict val)
```

- Bibliothèques

```
1 #include <string.h>

3 void *memcpy (void * restrict dest,
4 const void * restrict src,
5 size_t n);
void *memmove (void *dest, const void *src, size_t n);
```

Dans `memmove()` 2 zones peuvent se chevaucher...

# Nommage de type

(I)

- Déclarer des types complexes (tableaux de pointeurs de fonctions...) : compliqué
- Trimbaler des **struct**, **union** ou **enum** dans chaque déclaration : lourd
- ↗ Déclaration de types nommés par **typedef**
- Ressemble à déclaration de variable

```
1 #include <stdlib.h>
2
3 enum {N=1000, M=10000};
4
5 typedef double matrice[N][M];
6 matrice a, b;
7
8 // Le type n'est pas forcément encore défini :
9 typedef struct arbre_binaire *arbre_binaire;
10
11 struct arbre_binaire {
12 /* Comme il n'y a pas le mot struct devant gauche, c'est le type de
13 l'espace de nommage classique qui est utilisé, en l'occurrence le
14 type du typedef : */
15 arbre_binaire gauche;
16 // Taille connue car on sait que c'est un pointeur !
17 arbre_binaire droite;
18 // Accroche n'importe quel type de valeur à l'arbre
```

# Nommage de type

(II)

```
20 void*valeur;
21 }
22 //Déclare un pointeur vers un noeud de l'arbre:
23 arbre_binaire *arbre2;
24 /*On pourra allouer un vrai arbre plus tard avec par exemple:
25 *parbre2 = malloc(sizeof(struct_arbre_binaire));
26 *parbre2->gauche = *parbre2->droit = *parbre2->valeur = NULL;
27 */
```

# Expression de type

(I)

- Besoin d'expression de type anonyme
- Déclarations de prototypes de fonctions plus cours
- Utile si besoin de connaître taille d'un type
- S'obtient simplement en supprimant nom de variable à déclaration

```
1 (gdb) print sizeof(double[10])
$1 = 80
3 *(gdb) print sizeof(double*[10])
$2 = 40
5 *(gdb) print sizeof(double(*[10]))
$3 = 4
7 *(gdb) print sizeof(double**([10]))
$4 = 4
9 *(gdb) print sizeof(double*(*)())
$5 = 4
11 *(gdb) print sizeof(double(*[10])(int , float))
$6 = 40
```

# Expression de type

(II)

**sizeof()** peut servir de moyen de vérification du type qu'on a voulu créer... :-)

# Initialisations de variables

(I)

- Variables peuvent être initialisées lors de leur déclaration
- ~ Évite d'utiliser variables non initialisées, programme plus clair
- Contrainte : valeur d'initialisation doit être une constante (évaluable par compilateur et éditeur de lien)

```
1 char chaine [] = "bonjour";
2 double vieux_pi = 22.0/7.0;
// Résolu par éditeur de lien :
4 char **p = &chaine;
```

# Initialisations avec types compliqués (I)

- Initialisation de vecteurs

```
1 /* chaine est un tableau en mémoire globale dont les éléments
2 sont initialisés à la valeur des caractères */
3 char chaine [] = "ça bouge pour moi";
4 char studieux [] = {'p', 'é', 'n', 'i', 'b', 'l', 'e', '\0'};
5 // On aurait pu faire pire avec studieux[8] !
6 /* chaine_immuable est un pointeur vers une chaîne constante
7 dans les instructions du programme... */
8 char *chaine_immuable = "dans le code !";
```

```
10 int matrice [2][3] = {
11 {1, 2, 3},
12 {4, 5, 6}
};
```

- Initialisations incomplètes en C99

```
1 double gros_vecteur [1000000] = {4.5,
2 5.6,
3 [123456] = 7.89E-10,
4 [500000] = 4.6
5 };
```

# Initialisations avec types compliqués (II)

- ▶ Par défauts, éléments non spécifiés sont mis à 0
- ▶ Si tout à 0, optimisation : non stockage des valeurs
- ▶  Comparer avec

```
1 doublegros_vecteur_de_nuls[1000000];
```



Occupation mémoire :

```
-rw-r--r-- 1 keryell keryell 56 2005-11-06 11:06 gros_vecteur.c
-rw-r--r-- 1 keryell keryell 8000709 2005-11-06 11:06 gros_vecteur.o
-rw-r--r-- 1 keryell keryell 38 2005-11-06 11:08 gros_vecteur_de_nuls.c
-rw-r--r-- 1 keryell keryell 713 2005-11-06 11:08 gros_vecteur_de_nuls.o
```

- Unions & structures : possible de préciser les champs en C99

# Initialisations avec types compliqués (III)

```
1 enum stockage {Entier, Flottant};
2 typedef struct {
3 enum stockage type;
4 union {
5 int i;
6 double d;
7 } valeur; // Facultatif
8 element;
9 };
10
11 element e = {
12 .type = Entier,
13 .valeur.i = 3
14 };
15
16 typedef struct {
17 int x, y;
18 } vecteur [10];
19
20 vecteur v = {
21 {1, 3},
22 {2, 4},
23 {3, 5},
24 {4, 6},
25 {5, 7},
26 {6, 9},
27 {7, 8},
28 {8, 7},
29 {9, 6},
30 {10, 5}
31 };
```

# Initialisations avec types compliqués

(IV)

```
21 uu[5] uu={uu.y uu=uu3},
uu[7] uu={uu.x uu=uu1,uu.y uu=uu2uu},
23 uu{uu.x uu=uu3,uu.y uu=uu4uu}
};
```

# Variables temporaires

(I)

- Déclarées dans blocs d'instructions
- Dans code exécutable ↗ possible d'initialiser avec des expressions qui ne sont pas constantes

# Le plan

- 1 Introduction
  - Bibliographie
  - Petite histoire
- 2 Langage
  - Généralités
- 3 Un ordinateur ? Mais qu'est-ce ?
  - Les bases de la numérologie
  - Calcul binaire
- 4 Déclarations
  - Généralités
  - Types des objets
  - Portée
- 5 Expressions
  - Sémantique opérateurs
  - Coercition de type
- 6 Instructions
- 7 Préprocesseur
- 8 Bibliothèques
- 9 Modèle mémoire & allocation
- 10 Entrées-sorties
- 11 GNOME
- 12 Éditeur exemple d'Emacs
  - Tuning de son clavier
- 13 Compilation
- 14 Modularisation
- 15 Mise au point
- 16 Dévermineur (debugger)
- 17 Analyse mémoire
- 18 Extensions
  - Du C pour le graphisme
  - C++
- 19 Délires
  - Le bêtisier
  - Concours de programmes incompréhensibles
- 20 Conclusion
- 21 Index
- 22 Table des matières

# Définition des expressions

(I)

- Constituant élémentaire des instructions d'un programme C
- Expressions définies récursivement à partir d'autres expressions élémentaires ou non
- Partie qui fait
  - ▶ Généralement des calculs
  - ▶ Mais aussi rien ☺
  - ▶ Ou des effets de bord sur la mémoire (écritures)

# Expressions élémentaires littérales

(I)

- Constantes entières

- Base 10 (décimale) : classique !
- Base 8 (octale) : utiliser chiffres 0 à 7 avec un 0 devant  
 $022 = 18$   
 Erreur d'inattention et oubli du 0 devant! ☺
- Base 16 (hexadécimal) : chiffres décimaux puis a ou A à f ou F avec un 0x ou 0X devant  
`0Xdeadbeef`
- Extension GCC : 0b101100 pour entrer du binaire
-  Les nombres sont stockés dans l'ordinateur de manière normalisée (binaire) ∀ format d'entrée!!! ☺

## Suffixes modifiants type

- Constante non signée avec suffixe u ou U  
`0x23U`
- Type **long** avec suffixe l ou L 75L
- Type **long long** en doublant suffixe l ou L  
`29LLU`

# Expressions élémentaires littérales (II)

- Constantes flottantes
  - ▶ Notation scientifique avec chiffres avec . - et éventuellement e ou E pour préciser l'exposant en puissance de dix  
-6.023E-23
  - ▶ Nouveau en C99, flottants précisés en hexadécimal, exposant en puissance de 2 précisé par p ou P (le E est déjà pris par hexadécimal... ☺)  
 $-0xe.fP-3$  vaut  $-14.9375 \times 2^{-3} = -1.8671875$   
Permet de préciser exactement constantes flottantes sans approximation base 10 vers base 2 après virgule...
  - ▶ Par défaut type **double**, mais modification de précision possible
    - Suffixe f ou F pour **float**
    - Suffixe l ou L pour **long double**
- Constantes complexes
  - ▶ Entrées sous forme d'expression constante  
1.2 + 3.4\*I
- Constante caractère contenu entre 2 ,

# Expressions élémentaires littérales (III)

- ▶ Caractère quelconque sauf ', \ et *newline*
  - ▶ Caractère avec échappement '\'', '\"', '\\', '\n' (*newline*, '\t' (*tabulation*... cf séquences du tableau ASCII du transparent 3))
  - ▶ Caractère en octal \123
  - ▶ Caractère en hexadécimal \x7f
  - ▶ Caractères UNICODE en hexadécimal \u0123 et \U01230123 nouveau en C99
- Constantes chaînes de caractères
    - ▶ Style "une\"chaîne\"\_!" de type pointeur vers caractère et valeur adresse vers premier caractère
    - ▶  La chaîne contient le caractère '\0' final !
    - ▶ Chaîne existe en caractères larges wchar\_t comme L"gros"
    - ▶ Possible de couper chaînes en plusieurs lignes

# Expressions élémentaires littérales

(IV)

```
1 char longue_chaine [] = "début \
2 et fin";
3 char autre_chaine [] = "début"
4 et fin";
```

# Expressions élémentaires symbolique

(I)

- Énumérations : symbole TRUC dans `enum { TRUC }`
- Identifiant de vecteur : pointeur d'adresse constante qui pointe vers premier élément
- Identifiant de fonction : pointeur d'adresse constante vers code début fonction
- Labels : extension de GCC pour hacker code machine (compilation à la volée...) qui est l'adresse du label

```
1 ici:
2 #####adresse##=&ici;
```

# Constantes agrégats

(I)

- Permet de construire des constantes pour des types union, structures ou vecteurs

```
1 calcule(✉element){
2 ↪.type✉=✉Entier ,
3 ↪.valeur.i✉=✉3
4 }
```

# Constantes L-valeurs (lvalues)

(I)

- *lvalue* (L-valeur) est une valeur pouvant apparaître dans une affectation à gauche d'un « = »
- Exemple d'utilisation de L-valeur constante comme adresse d'un périphérique en mémoire type écran bitmap acceptant des entiers à partir adresse (**int**\*)<sub>0xE0000000</sub> :

```
1 ((int*)0xE0000000)[pixel] = couleur;
```

# Constructions d'expression

(I)

- Construction récursive d'expressions à partir d'opérateurs et expressions élémentaires
- Opérateur acceptant  $o$  opérande est dit d'arité  $o$
- Opérateur d'arité 1 (unaire) peut être combiné
  - ▶ À gauche de l'opérande, préfixe

1 -3

▶ À droite de l'opérande, postfixe

1 f()

- Opérateur d'arité mis entre 2 opérandes : infixe

1 4 + 5

# Priorité des opérateurs

(I)

- Éviter ambiguïté au maximum  
 $1+2*3$  vaut 9 ou 7 ?
- Coller aux règles mathématiques classiques
- ↗ Généraliser et rajouter la notion d'ordre de priorité
  - ▶ Mathématiquement,  $*$  prioritaire par rapport à  $+$
- Problème lorsqu'opérateurs de même priorité en concurrence  
 $1-2+3$  vaut 2 ou -4 ?
  - ↗ Rajouter notion d'associativité
    - ▶ Associativité de gauche à droite : opérateurs arithmétiques classiques
    - ▶ Associativité de droite à gauche : opérateur - unaire arithmétique classique

# Priorité des opérateurs

(II)

| Opérateur                                                      | Associativité   | Arité       |
|----------------------------------------------------------------|-----------------|-------------|
| <code>() [] -&gt; .</code>                                     | gauche à droite | $\approx 2$ |
| <code>! ~ ++ -- + - (type) * &amp; sizeof</code>               | droite à gauche | 1           |
| <code>* / %</code>                                             | gauche à droite | 2           |
| <code>+ -</code>                                               | gauche à droite | 2           |
| <code>&lt;&lt; &gt;&gt;</code>                                 | gauche à droite | 2           |
| <code>&lt;&lt;= &gt;&gt;=</code>                               | gauche à droite | 2           |
| <code>== !=</code>                                             | gauche à droite | 2           |
| <code>&amp;</code>                                             | gauche à droite | 2           |
| <code>^</code>                                                 | gauche à droite | 2           |
| <code> </code>                                                 | gauche à droite | 2           |
| <code>&amp;&amp;</code>                                        | gauche à droite | 2           |
| <code>  </code>                                                | gauche à droite | 2           |
| <code>a ? b : c</code>                                         | droite à gauche | 3           |
| <code>= += -= *= /= %= &lt;&lt;= &gt;&gt;= &amp;= ^=  =</code> | droite à gauche | 2           |
| <code>,</code>                                                 | gauche à droite | 2           |

- Rajouter des parenthèses si cela diffère du défaut ou pour clarifier ou assurer ☺
- En ligne : `man operator`

# Conversions de type

(I)

- C est un langage à typage statique : types des objets déterminés lors de la compilation
- Si des types d'opérandes ne correspondent pas, C définit des règles de conversions implicites rajoutées dans le programme

- ▶ Conversion entier vers flottant

```
1 double a = 3 + 2.5;
2 // a vaut 5.5
```

- ▶ Conversion flottant vers entier

```
1 int i = 2.5;
2 // i vaut 2
```

- ▶ Conversion dans un type plus petit : ne garde que les bits de poids faible qui logent

```
1 char c = 0x1234
2 // c vaut 0x34
```

# Conversions de type

(II)



Si ce n'est pas fait exprès... ☺ Mais très pratique si fait exprès pour arithmétique modulo

- ▶ Conversion pour calcul vers type le plus précis
- ▶ ...

# Le plan

- 1 Introduction
  - Bibliographie
  - Petite histoire
- 2 Langage
  - Généralités
- 3 Un ordinateur ? Mais qu'est-ce ?
  - Les bases de la numérologie
  - Calcul binaire
- 4 Déclarations
  - Généralités
  - Types des objets
  - Portée
- 5 Expressions
  - Sémantique opérateurs
  - Coercition de type
- 6 Instructions
- 7 Préprocesseur
- 8 Bibliothèques
- 9 Éditeur exemple d'Emacs
  - Tuning de son clavier
- 10 Compilation
- 11 Modularisation
- 12 Mise au point
  - Dévermineur (debugger)
  - Analyse mémoire
- 13 Extensions
  - Du C pour le graphisme
  - C++
- 14 Délires
  - Le bêtisier
  - Concours de programmes incompréhensibles
- 15 Conclusion
- 16 Index
- 17 Table des matières

# Opérateur .

(I)

- Permet d'accéder à champ de structure ou d'union

```
1 typedef struct{
2 int x, y;
3 } point;
4
5 point a;
6 a.x = 3;
7 a.y = 5;
8 d = sqrt(a.x*a.x + a.y*a.y);
```

# Opérateur ->

(I)

- Si allocation dynamique, souvent construction de type

```
1 point * p;
2 p = malloc(sizeof(point));
3 (*p).x = 3;
4 (*p).y = 5;
5 d = sqrt((*p).x * (*p).x + (*p).y * (*p).y);
6 ...
7 free(p);
```

Lourd car priorité de . supérieure à celle de \* ☺

- ↗ Abréviation avec notation ->

```
1 p->x = 3;
2 p->y = 5;
3 d = sqrt(p->x*p->x + p->y*p->y);
```

Assez mnémotechnique...

# Indication vecteur et matrices []

(I)

- Permet d'accéder à élément d'un vecteur ou d'une matrice

vecteur [indice]

```
1 #include <stdio.h>
2 int main(int argc, char *argv[])
3 {
4 for (int i = 0; i < argc; i++)
5 printf("L'argument %d vaut \"%s\"\n", i, argv[i]);
}
```

- Abréviation d'arithmétique sur pointeur : expressions suivantes équivalentes

```
1 v[i] = 3;
2 *(v + i) = 3;
3 i[v] = 3; // !!! :-)
```

# Indication vecteur et matrices []

(II)

- Généralisation dans le cas des matrices

1 *identifiant*[*dim<sub>1</sub>*] [*dim<sub>2</sub>*] ··· [*dim<sub>n</sub>*]

- ⚠️ ⚠️ Matrice bidimensionnelle et vecteur de vecteur par exemple s'accèdent syntaxiquement pareils mais ne sont pas équivalents !

1 **char** *matrice*[2][3] = {{0,1,2}, {0x10, 0x11, 0x12}};

|               |               |
|---------------|---------------|
| ⋮             | ⋮             |
| 7ffffdfd751b0 | matrice[0][0] |
| 7ffffdfd751b1 | matrice[0][1] |
| 7ffffdfd751b2 | matrice[0][2] |
| 7ffffdfd751b3 | matrice[1][0] |
| 7ffffdfd751b4 | matrice[1][1] |
| 7ffffdfd751b5 | matrice[1][2] |
| ⋮             | ⋮             |

# Indication vecteur et matrices []

(III)

```
1 // A priori beaucoup plus dynamique
2 char **vecteur_de_vecteur;
3 vecteur_de_vecteur = calloc(2, sizeof(char *));
4 vecteur_de_vecteur[0] = calloc(5, sizeof(char));
5 // 2ème ligne plus grande
6 vecteur_de_vecteur[1] = calloc(8, sizeof(char));
7 vecteur_de_vecteur[1][4] = 7;
```

# Indication vecteur et matrices []

(IV)

|          |    |                                          |
|----------|----|------------------------------------------|
| 01a23010 | 30 | vecteur_de_vecteurs[0](0000000001a23030) |
| 01a23011 | 30 |                                          |
| 01a23012 | a2 |                                          |
| 01a23013 | 01 |                                          |
| 01a23014 | 00 |                                          |
| 01a23015 | 00 |                                          |
| 01a23016 | 00 |                                          |
| 01a23017 | 00 |                                          |
| 01a23018 | 50 | vecteur_de_vecteurs[1](0000000001a23050) |
| 01a23019 | 30 |                                          |
| 01a2301a | a2 |                                          |
| 01a2301b | 01 |                                          |
| 01a2301c | 00 |                                          |
| 01a2301d | 00 |                                          |
| 01a2301e | 00 |                                          |
| 01a2301f | 00 |                                          |
| 01a23030 | 00 | vecteur_de_vecteurs[0][0]                |
| 01a23031 | 00 | vecteur_de_vecteurs[0][1]                |
| 01a23032 | 00 | vecteur_de_vecteurs[0][2]                |
| 01a23050 | 00 | vecteur_de_vecteurs[1][0]                |
| 01a23051 | 00 | vecteur_de_vecteurs[1][1]                |
| 01a23052 | 00 | vecteur_de_vecteurs[1][2]                |
| 01a23053 | 00 | vecteur_de_vecteurs[1][3]                |
| 01a23054 | 07 | vecteur_de_vecteurs[1][4]                |
|          | .  |                                          |
|          | .  |                                          |
|          | .  |                                          |

# Appels de fonctions & procédures (I)

- ⚠ Paramètres effectifs passés par valeur (copiés dans paramètres formels de la fonction)

1 *identifiant( expr<sub>1</sub> , . . . , expr<sub>n</sub>)*

- ▶ ↗ Fonction ne peut pas modifier paramètres effectifs d'appel ⚠
- ▶ ↗ Passer en paramètre adresse de (pointeur vers) quelque chose qu'on veut modifier ⚠

- Dans le cas de vecteurs, seule adresse est passé (pas de copie d'éléments)
- Dans le cas de structure, toute la structure est copiée ↗ pratique mais ⚠ performances...
- ⚠ ⚠ Ordre d'évaluation des paramètres effectifs non spécifiée

1 `appel(affiche(a), affiche(b), affiche(c));`

non déterministe ! ↗ ⚠ si effet de bord dans paramètres effectifs...

# Arithmétique

(I)

Opérateurs classiques sur entiers, flottants et complexes

- + addition
- - soustraction (infixe)
- - négation (préfixe)
- \* multiplication
- / division (si calculs en entiers, troncature vers 0, quel que soit le signe)
- % modulo (reste de la division entière définie précédemment, opérateur seulement pour des types entiers)



# Arithmétique binaire

(I)

Opérations entre entiers pris bit à bit  $c = a \text{ op } b$ , pour chaque bit  $i$  :

$$c_i = a_i \text{op} b_i$$

- $\&$  et booléen ( $c_i = a_i \wedge b_i$ )

$$0b01011(11) \wedge 0b01101(13) = 0b1001(9)$$

- $|$  ou inclusif booléen ( $c_i = a_i \vee b_i$ )

$$0b1001(9) \vee 0b101(5) = 0b1101(13)$$

- $\sim$  ou exclusif booléen ( $c_i = a_i \oplus b_i$ )

$$0b1001(9) \oplus 0b101(5) = 0b1100(12)$$

# Arithmétique binaire

(II)

- $\sim$  négation ( $c_i = \neg a_i$ ), complément restreint Sur un char :

$$\neg 0b11101001 = 0b00010110$$

- Opérateurs de décalage

- ▶  $<<$  décalage à gauche nombre de  $b$  chiffres binaires

$$a << b = a \times 2^b$$

$$0b101(5) << 2 = 0b10100(20)$$

- ▶  $>>$  décalage à droite nombre de  $b$  chiffres binaires

$$a >> b = \lfloor \frac{a}{2^b} \rfloor$$

$$0b10110(22) >> 2 = 0b101(5)$$

Corollaire subtile : si  $a$  est de type signé, les bits rajoutés à gauches sont la réplication du bit de signe (notation en complément à 2)

# Arithmétique binaire

(III)

*b* doit être positif et de taille compatible avec nombre de bits de *a*

- Opérations extrêmement puissante à qui sait les utiliser ! ☺

  - ▶ Robot E=M6 : gestion de port parallèle

    - Mettre 1 bit *b* à 1

```
1 parallel = parallel | (1 << b);
2 parallel_out(parallel);
```

    - Mettre 1 bit *b* à 0

```
1 parallel = parallel & ~(1 << b);
2 parallel_out(parallel);
```

  - ▶ Garder *b* bits de poids faible

```
1 c = a & ((1 << b) - 1);
```

  - ▶ Extraire bits *b* à *c* de *a*

```
1 d = (a >> b) & ((1 << (c + 1 - b)) - 1);
```

On peut aussi utiliser des champs de bits pour faire ce genre de choses

# Optimisations en binaire

(I)

Quelques exemples trouvés dans virtualiseur QEMU qui doit implémenter des instructions de processeurs sur d'autre processeurs

- Trouver nombre de 0 binaires en tête d'un entier
  - ▶ Si le processeur sait le faire : utiliser instruction assembleur qui peut être cachée dans

```
1 T0=__builtin_clz(T1);
```

- ▶ Solution naïve en  $\mathcal{O}(n)$  avec boucle

```
1 for(int i= sizeof(int)*CHAR_BIT-1; i>=0)
 if((T1&(1<<i))!=0)
 break;
 T0=sizeof(int)*CHAR_BIT-1-i;
```

- ▶ Solution astucieuse  $\mathcal{O}(\log n)$  (avec affectations conditionnelles...)

# Optimisations en binaire

(II)

```
1 /* Binary search for leading zeros */
2 T0=1;
3 if ((T1>>16)==0) {
4 T0=T0+16;
5 T1=T1<<16;
6 }
7 if ((T1>>24)==0) {
8 T0=T0+8;
9 T1=T1<<8;
10 }
11 if ((T1>>28)==0) {
12 T0=T0+4;
13 T1=T1<<4;
14 }
15 if ((T1>>30)==0) {
16 T0=T0+2;
17 T1=T1<<2;
18 }
19 T0=T0-(T1>>31);
```

# Optimisations en binaire

(III)

- Exemple du comptage de population de 1 dans un entier SPARC64 `popc` nativement en  $\mathcal{O}(\log n)$  réalisable par

```
1 uint32_t uT0;
3 T0=u(T1&0x55555555)+(T1>>1)&0x55555555);
T0=u(T0&0x33333333)+(T0>>2)&0x33333333);
5 T0=u(T0&0x0f0f0f0f)+(T0>>4)&0x0f0f0f0f);
T0=u(T0&0x00ff00ff)+(T0>>8)&0x00ff00ff);
7 T0=u(T0&0x0000ffff)+(T0>>16)&0x0000ffff);
```

# Applications codage binaire : multispin-coding (I)

- Exploitation du parallélisme en bits
- Ranger plusieurs petites données par mot machine
- 4 opérations sur 64 bits/cycle  $\equiv$  256 opérations sur 1 bit/cycle !
- Opérations binaire style `^`, `&`, `|`, `~` sans problème
- Jeux d'instructions :
  - ▶ 1 Alpha 21164 à 600 MHz  $\equiv$  76,8 GIPS 1 bit, 9,6 GIPS 8 bits
  - ▶ 1 Pentium 4 SSE3 à 4 GHz : 2 opérations 128 bits/cycle  $\equiv$  1 TIPS ( $10^{12}$  opérations par secondes) 1 bit
- Idée : plutôt que de résoudre 1 problème à la fois, éclate problème en binaire pour calculer 256 tranches de problèmes binaires à la fois

Exemple : bibliothèques de cassage de codes cryptographiques (détection mots de passe faibles avec John the Ripper), traitement d'image, traitement du signal, codage, optimisation de programmes...



# Application utilisant des additions 9 et 6 bits (I)

- Compactage dans 32 bits `a_xxs_yys` :

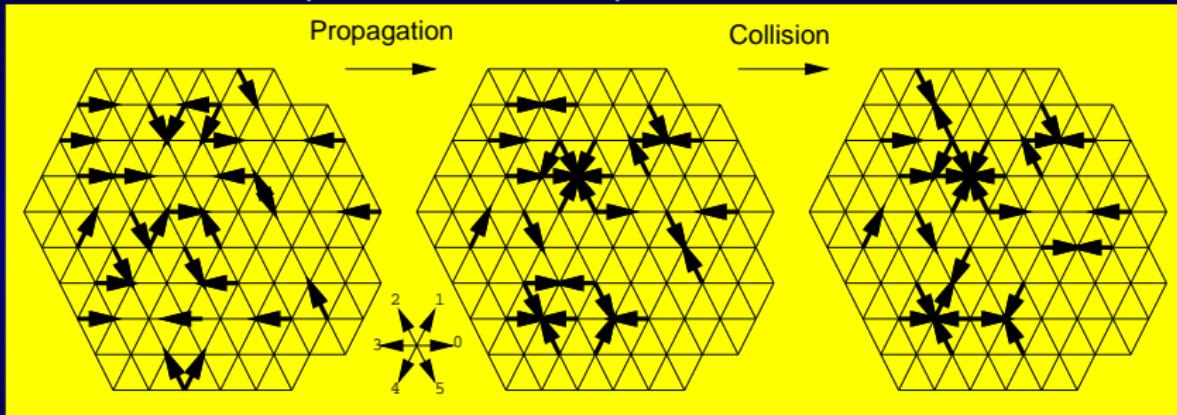
| quality | q_a | 0 | q_x | 0 | q_y |
|---------|-----|---|-----|---|-----|
| 8       | 6   | 1 | 8   | 1 | 8   |

- q\_a sur 6 bits, q\_x et q\_y sur 8 bits
- Opérations sur q\_x et q\_y sur 9 bits ↗ stockage sur 9 bits aussi (évite l'extraction)
- Garde des 0 délimiteurs absorbant les retenues (& *masque*)
- Besoin de tester  $(q_x, q_y) \in [-128, 127]^2$  :
  - Changement repère (biais +128) ↗  $(q'_x, q'_y) \in [0, 255]^2$
  - Test de `a_xxs_yys & ((1<<8) + (1<<18)) == 0` : 1 instruction !

# Gaz sur réseau

(I)

- Sites contenant des particules se déplaçant quantiquement
- Interactions entre particules sur chaque site



- Tableau de sites contenant 1 bit de présence d'1 particule allant dans 1 direction
- Symétrie triangulaire
- Compactage de 32 ou 64 sites/int par direction

# Gaz sur réseau

(II)

```

1 a_=lattice [RIGHT];
2 b_=lattice [TOP_RIGHT];
c_=lattice [TOP_LEFT]; // Particules qui montent à gauche
4 d_=lattice [LEFT]; // Particules qui vont à gauche
e_=lattice [BOTTOM_LEFT];
6 f_=lattice [BOTTOM_RIGHT];
s_=solid; // Une condition limite
8 ns_=~s;
r_=lattice [RANDOM]; // Un peu d'aléa
10 nr_=~r;
/* A triplet */
12 triple_= (a^b)&(b^c)&(c^d)&(d^e)&(e^f);
/* Doubles */
14 double_ad_= (a&d&~(b|c|e|f));
double_be_= (b&e&~(a|c|d|f));
16 double_cf_= (c&f&~(a|b|d|e));
/* The exchange of particles : */
18 change_ad_= triple_| double_ad_| (nr&double_cf);
change_be_= triple_| double_be_| (r&double_cf)| (nr&double_ad);
20 change_cf_= triple_| double_cf_| (r&double_ad)| (nr&double_be);

```

# Gaz sur réseau

(III)

```
22 bl = blow[N_DIR];
23 s &= ~bl;
24 ns &= ~bl;
25 /* Where there is blowing, collisions are no longer valuable : */
26 lattice [RIGHT] = (((a^change_ad)&ns) | (d&s))
27 | bl&blow[RIGHT];
28 lattice [TOP_RIGHT] = (((b^change_be)&ns) | (e&s))
29 | bl&blow[TOP_RIGHT];
30 lattice [TOP_LEFT] = (((c^change_cf)&ns) | (f&s))
31 | bl&blow[TOP_LEFT];
32 lattice [LEFT] = (((d^change_ad)&ns) | (a&s))
33 | bl&blow[LEFT];
34 lattice [BOTTOM_LEFT] = (((e^change_be)&ns) | (b&s))
35 | bl&blow[BOTTOM_LEFT];
36 lattice [BOTTOM_RIGHT] = (((f^change_cf)&ns) | (c&s))
37 | bl&blow[BOTTOM_RIGHT];
```

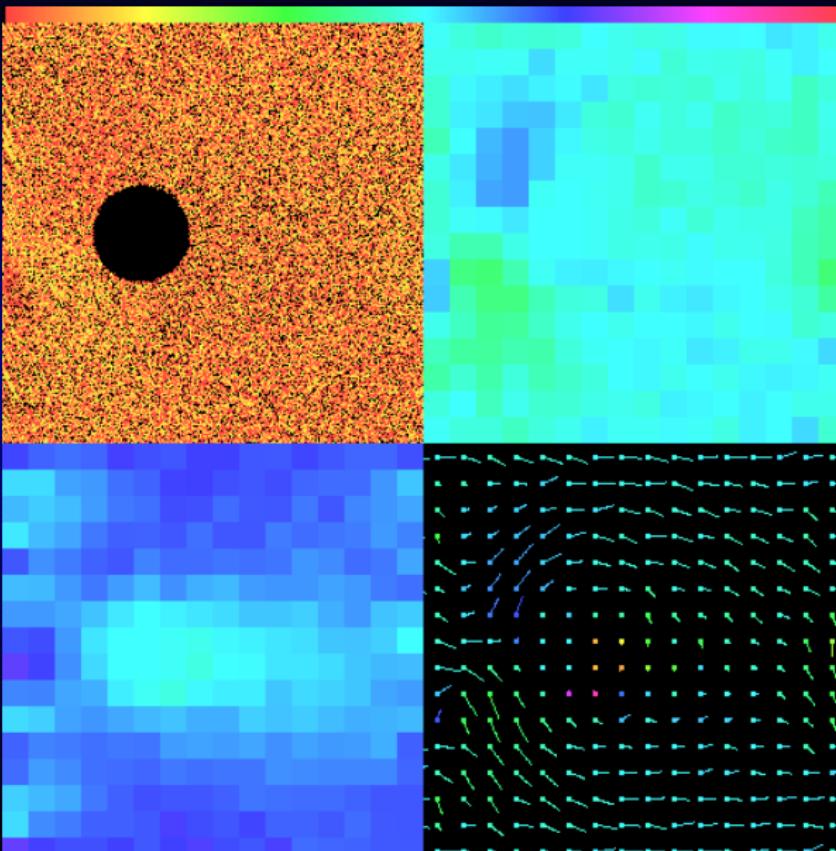
# Gaz sur réseau

(IV)

- Cylindre

# Gaz sur réseau

(V)



# Expressions logiques

(I)

- Renvoient 1 pour vrai et 0 pour faux
- Opérateurs de comparaison classiques
  - ▶ <
  - ▶ <= inférieur ou égal
  - ▶ == test d'égalité  à ne pas confondre avec opérateur d'affectation... 
  - ▶ != test d'inégalité
  - ▶ >= supérieur ou égal
  - ▶ >
- Négation booléenne préfixe : !  à ne pas confondre avec opérateur sur bits ~
- Opérateur booléen et

1     $expr_1 \sqcup \&& \sqcup expr_2$

- ▶ Travaille sur opérateurs booléens

# Expressions logiques

(II)

- ▶ ↗ Optimisation : si  $expr_1$  est faux, résultat faux indépendant d' $expr_2$
- ▶ ⚡ En C, si première expression fausse, la seconde *n'est pas* évaluée
- ▶ Important si effets de bords...
- ▶ Pratique : permet programmation subtile avec moins de **if ()**
- ▶ Semblable à opérateur **&&** du shell
- ▶ ⚡ À ne pas confondre avec opérateur sur bits **&** qui évalue 2 opérandes

Si booléens non normalisés :  $1 \& 2$  vaut 0, donc

Vrai & Vrai = Faux... ☺

Normalisation :  $!(!a \vee !b)$

- Opérateur booléen *ou* inclusif

1  $expr_1 \vee \vee expr_2$

- ▶ Travaille sur opérateurs booléens
- ▶ ↗ Optimisation : si  $expr_1$  est vrai, résultat vrai indépendant d' $expr_2$

# Expressions logiques

(III)

- ▶  En C, si première expression vraie, la seconde *n'est pas* évaluée
  - ▶ Important si effets de bords...
  - ▶ Pratique : permet programmation subtile avec moins de `if ()`
  - ▶ Semblable à opérateur `||` du shell
  - ▶  À ne pas confondre avec opérateur sur bits `|` qui évalue 2 opérandes
- Pas d'opérateur booléen *ou* exclusif
    - ▶ Pas d'optimisation permettant de supprimer une évaluation
    - ▶ ↗ Utiliser opérateur sur bits `^` mais  si son normalisé...
    - ▶ Pour normaliser utiliser `!a ^ !b`

# Expression conditionnelle

(I)

```
1 expressioncond ? expressionsi-vraie : expressionsi-fausse
```

- Selon la condition renvoie résultat d'une expression ou de l'autre
- Pratique pour mettre des tests là où on ne peut pas mettre d'instruction
- Exemple affichant un chiffre binaire en français

```
1 puts(n==0?"zero":"un");
```

# Opérateurs à effet de bord

(I)

- Affectation

- ▶  $a = b$  ;
- ▶ Originalité du C : affectation est une expression et non une instruction : renvoie valeur affectée
- ▶  Si on affecte un vecteur à un autre, pas de copie d'éléments, juste nouveau vecteur pointe vers ancien vecteur
- ▶ Par contre affectation de structure copie la structure (idem dans appel de fonction)

- Affectation avec opération arithmétique

- ▶ Syntaxe de type  $a \ op= b$  équivalente à  $a = (a) \ op \ (b)$  sauf que  $a$  n'est évalué qu'une fois

```
1 // Affiche une chose :
2 *pointe_et_affiche () += 3;
// Affiche deux choses :
4 *pointe_et_affiche () = *pointe_et_affiche () + 3;
```

Donc  si effets de bords...

# Opérateurs à effet de bord

(II)

- ▶ Aussi une expression
- Incrémentation/décrémentation
  - ▶ Existe en version

Préfixe : agit avant renvoi de valeur

- 1  $\text{++v}$  incrémente  $v$  et renvoie sa valeur
- 2  $--v$  décrémente  $v$  et renvoie sa valeur

Postfixe : agit après renvoi de valeur

- $v++$  renvoie la valeur de  $v$  et incrémente  $v$
- $v--$  renvoie la valeur de  $v$  et décrémente  $v$

- ▶ Exemple classique de concision

```
1 char* p = "cherche la fin";
2 while (*++p != '\0')
3 ;
4 // p pointe sur le '\0' final
```

# Taille d'un objet

(I)

- **sizeof()**
- Renvoie une taille en *caractères* de type `size_t` définie avec

```
1 #include <stddef.h>
```

Penser à multiplier par `CHAR_BIT` pour avoir la taille en bits...

- Autre macro intéressante `offsetof(une_struct, un_champ)` qui renvoie le déplacement du champ par rapport au début de la structure



# Le plan

- 1 Introduction
  - Bibliographie
  - Petite histoire
- 2 Langage
  - Généralités
- 3 Un ordinateur ? Mais qu'est-ce ?
  - Les bases de la numérologie
  - Calcul binaire
- 4 Déclarations
  - Généralités
  - Types des objets
  - Portée
- 5 Expressions
  - Sémantique opérateurs
  - Coercition de type
- 6 Instructions
- 7 Préprocesseur
- 8 Bibliothèques
- 9 Éditeur exemple d'Emacs
  - Tuning de son clavier
- 10 Compilation
- 11 Modularisation
- 12 Mise au point
  - Dévermineur (debugger)
  - Analyse mémoire
- 13 Extensions
  - Du C pour le graphisme
  - C++
- 14 Délires
  - Le bêtisier
  - Concours de programmes incompréhensibles
- 15 Conclusion
- 16 Index
- 17 Table des matières

# Coercition : conversion explicite de type (I)

## Definition

La coercition est l'usage délibéré et sensé de menaces exprimées ouvertement pour influencer les choix stratégiques. Cette définition large permet d'inclure les différentes façons de mettre en pratique la coercition tout en conservant l'idée principale d'influence c'est-à-dire d'agir en sorte que l'entité dont on veut influencer le comportement va renoncer à ses choix initiaux et adopter les solutions préconisées par l'agent qui met en pratique cette coercition.

Julien DUVAL,

[http://www.departmentofintelligence.com/fr/science\\_politique/scpo9.htm](http://www.departmentofintelligence.com/fr/science_politique/scpo9.htm)



# Coercition : conversion explicite de type (II)

À utiliser lorsqu'on n'est pas persuadé du bon type

*La coercition prend ses racines suite à l'échec de la persuasion. La persuasion demande comme l'indique Massimo Piattelli Palmarini « d'anticiper sur les motivations et les schémas mentaux de nos interlocuteurs pour mieux en triompher ». Persuader poursuit il, « c'est provoquer un changement de la volonté d'autrui, mais attention, seulement à travers un transfert de croyances ou d'opinions ». La coercition ne joue pas sur ce ressort mais sur celui de la contrainte. L'objectif est « d'amener quelqu'un à faire quelque chose » et peu importe les moyens utilisés.*



# cast en anglais

(I)

## Definition

Alors Brice, aujourd'hui on va voir la programmation, tu pars bien haut, bien abstrait d'un void et tu descend sur un entier signé. Et tu cast, et tu cast et tu cast.

Parodie par Jérôme HERMAN

Équivalent de la coercition ou le transtypage en français



# Transtypage explicite ou *cast*

(I)

1  $(type) \llcorner expression$

## Intérêt multiple

- Promotions

```
1 double \llcorner a;
2 int \llcorner m, n;
3 a \llcorner= \llcorner (double) \llcorner m \llcorner / \llcorner (double) \llcorner n;
```

- Permet troncatures simplement

```
1 int \llcorner i, \llcorner c;
2 double \llcorner a, \llcorner b;
3 a \llcorner= \llcorner (int) \llcorner b;
c \llcorner= \llcorner (char) \llcorner i;
```

- Toutes sortes de puissantes sordicités ☺ ↗

# Transtypage explicite ou *cast*

(II)

```
1 int64_t n;
2 int16_t f;
double d;
4 n=*((int64_t*)&d;
// n contient la représentation binaire de d
6 f=((int16_t*)&d)[2];
// f contient les bits 32 à 47 de d
8 (char*)&d)[7]=1;
/* bricole d en lui rajoutant 1
 à son dernier octet de mantisse */
10 /*
```

Extrêmement pratique en électronique, système embarqué,  
manipulation de codage...

- Évidemment pas portable ☺ mais toujours mieux que de l'écrire  
en assembleur ☺ ↗ À contrôler avec pré-processeur... 
-  tous les ordinateurs ne rangent pas octets, voire bits des  
entiers dans le même sens... 

# Pointeurs & manipulation d'adresses (I)

- & (préfixe) permet de récupérer adresse (référence) d'un objet
- \* (préfixe) permet de déréférencer un pointeur (accéder à objet données pointées)
- Ne peut pas avoir l'adresse d'un champ de bits car adresse mémoire en C toujours en octet, pas adresse de bit
- Pour faire du passage d'arguments par référence dans fonction, utiliser des références

```
1 void foo (double *a, int *b){
2 ...
3 *a=3.4;
4 *b=42; // La réponse !
5 ...
6 }

8 double bar;
9 int daurade;
10 ...
11 foo (&bar, &daurade);
```

# Pointeurs & manipulation d'adresses

(II)

|           |    |                        |
|-----------|----|------------------------|
| 08049b40  | 2a | daurade (0000002a)     |
| 08049b41  | 00 |                        |
| 08049b42  | 00 |                        |
| 08049b43  | 00 |                        |
| 08049b48  | 33 | bar (400b333333333333) |
| 08049b49  | 33 |                        |
| 08049b4a  | 33 |                        |
| 08049b4b  | 33 |                        |
| 08049b4c  | 33 |                        |
| 08049b4d  | 33 |                        |
| 08049b4e  | 0b |                        |
| 08049b4f  | 40 |                        |
| fffb75170 | 48 | a (08049b48)           |
| fffb75171 | 9b |                        |
| fffb75172 | 04 |                        |
| fffb75173 | 08 |                        |
| fffb75174 | 40 | b (08049b40)           |
| fffb75175 | 9b |                        |
| fffb75176 | 04 |                        |
| fffb75177 | 08 |                        |



# Arithmétique sur pointeurs

(I)

- Sémantique spéciale d'arithmétique sur pointeurs
- Suppose qu'un pointeur pointe vers un (hypothétique) vecteur d'objet ↗ ressemble
- Si  $p$  pointe vers un élément
  - ▶  $p - 1$  ou  $\&p[-1]$  pointe vers élément précédent
  - ▶  $p + 1$  ou  $\&p[1]$  pointe vers élément suivant
- On peut utiliser opérateurs de comparaisons sur pointeurs pour savoir si objets pointeurs sont avant ou après dans un vecteur
- Si  $p$  et  $q$  sont pointeurs sur objets de même type,  $q - p$  est la distance en nombre d'objets entre  $p$  et  $q$
-  Ne pas confondre arithmétique sur pointeurs avec arithmétique sur adresse
  - ▶ Comme en C la mémoire est accédée par caractère, équivalence dans le cas de pointeurs sur des caractères

# Arithmétique sur pointeurs

(II)

- Concrètement si

```
1 obj* p;
2 int d;
```

alors

$$p+d \equiv (\text{obj}*) (d * \text{sizeof}(\text{obj}) + (\text{char}*) p)$$

-  Certains processeurs ont des contraintes « d'alignement » sur objets vers une certaine taille pour simplifier et accélérer accès. Par exemple si accès à un **double** à adresse non multiple de 8 : *bus error* sur SPARC

# Liste d'expressions

(I)

```
1 expression1 , expression2
```

- Évaluation des 2 expressions
- Renvoie seulement valeur dernière expression
- Permet de mettre plusieurs expressions là où il n'y a place que pour une et où instruction interdite
- Exemple

```
1 for (i=0, j=1, k=n; i<m; i++, j+=3, k-=n)
 ...
```



# Le plan

- 1 Introduction
  - Bibliographie
  - Petite histoire
- 2 Langage
  - Généralités
- 3 Un ordinateur ? Mais qu'est-ce ?
  - Les bases de la numérologie
  - Calcul binaire
- 4 Déclarations
  - Généralités
  - Types des objets
  - Portée
- 5 Expressions
  - Sémantique opérateurs
  - Coercition de type
- 6 Instructions
- 7 Préprocesseur
- 8 Bibliothèques
- 9 Modèle mémoire & allocation
- 10 Entrées-sorties
- 11 GNOME
- 12 Éditeur exemple d'Emacs
  - Tuning de son clavier
- 13 Compilation
- 14 Modularisation
- 15 Mise au point
- 16 Dévermineur (debugger)
- 17 Analyse mémoire
- 18 Extensions
- 19 Du C pour le graphisme
- 20 C++
- 21 Délires
- 22 Le bêtisier
- 23 Concours de programmes incompréhensibles
- 24 Conclusion
- 25 Index
- 26 Table des matières

# Instructions

(I)

Brique constituante d'un programme

- Sans type ni valeur
- 3 sortes d'instructions
  - ▶ Instructions élémentaires
  - ▶ Instructions composées avec structures de contrôles
  - ▶ Instructions de saut

Très simples comparées aux expressions...

# Instructions élémentaires

(I)

- Expression suivie par un ;

```
1 q=3;
2 inutile;
// Encore plus inutile ici :
4 ;
5 for(;;)
6 // Mais pas là :
7 ;
8 /* On ne s'en sortira jamais de cette boucle infinie ... */
```

- On peut regrouper des instructions dans des blocs entre { et }
- On peut rajouter des étiquettes (labels) devant des instructions pour référence ultérieure (saut)

# Instructions élémentaires

(II)

```
1 uuvolatile uuint uuentrée;
2 uu/* Les labels sont bien pratiques pour simplifier
uu la gestion des erreurs : */
4 gère_entrée:
5 uulire();
6 uu// Méchante attente active:
7 uufor(;;)
8 uuuu if uu(entrée==3)
uuuuuuuu goto uugère_entrée;
```

# Tests if

(I)

```
1 if \sqcup (expression)
2 $\sqcup\sqcup\sqcup$ instruction-si-vrai
3 else
 $\sqcup\sqcup\sqcup$ instruction-si-faux
```

- Branche **else** optionnelle
- Considérée comme vraie *expression* de valeur non nulle

# Tests if

(II)

-  Dans expressions de test en général à ne pas confondre

► Si  $v$  vaut  $w$  alors...

1    **if**  $v == w$

► Met  $w$  dans  $v$  et si  $v$  est non nul alors...

1    **if**  $v = w$

# Aiguillage switch

(I)

- Compare *expression* par rapport à plusieurs valeurs constantes entières

```
1 switch(caractere){
2 case 'é':
3 majuscule = 'É';
4 voyelle++;
5 break; // Sort du switch
6
7 case ' ':
8 blanc++;
9
10 default:
11 non_alpha++;
12 }
```

- Partie **default** est optionnelle
- ⚠ Si pas de **break** dans un **case**, on exécute aussi la suite...

# Boucle while

(I)

```
1 while (abs(e) >= epsilon)
2 e = f(v);
```

- Condition évaluée *avant* exécution corps de boucle
- Boucle si vrai (si expression de valeur non nulle)
- De manière générale dans boucles en C, **break**; permet de sortir de la boucle courante (pour autres cas, utiliser par exemple **goto**)
- De manière générale dans boucles en C, **continue**; permet de passer à itération suivante de la boucle courante

# Boucle do

(I)

```
1 do
2 i++;
while(i < f);
```

- Condition évaluée *après* exécution corps de boucle
- ↗ Corps de boucle exécuté au moins une fois

# Boucle for

(I)

- Succès syntaxique autour de boucle while

```
1 for(expr-init; expr-test; expr-fin-corps)
2 instruction-corps
```

- Sur entrée de boucle, évalue *expr-init*. En C99, cette expression peut en fait contenir une initialisation de variable à visibilité sur boucle
- Itère tant que *expr-test* est vraie (valeur non nulle). Évaluée avant toute itération
- *expr-fin-corps* Évaluée en fin de chaque itération, après corps de boucle

```
1 for(int i=0; i<taille; i++)
2 printf("Valeur de v[i] = %f\n", i, v[i]);
```

# Instructions de saut

(I)

- Lorsque la vie ne doit plus être un long fleuve tranquille...

- Échappement

- ▶ **continue**; passe à itération suivante dans une boucle
- ▶ **break**; arrête boucle ou **switch()** courants
- ▶ Retour de fonction ou procédure

```
1 return expression;
```

arrête fonction en cours et renvoie valeur de l'expression qui doit avoir type compatible avec déclaration de fonction

Cas particulier : une procédure n'a pas besoin d'appeler **return** ; mais peut aussi l'utiliser sans expression

- Branchement

```
1 goto label;
```

permet de continuer exécution à l'endroit du *label*

# Instructions de saut

(II)

```
1 for(...)
2 for(...) {
3 ...
4 if (erreur_commise)
5 goto erreur;
6 }
7 erreur:
8 fprintf(stderr, "Erreur rencontrée : "
9 "valeur %f incorrecte\n", valeur);
10 exit(-1);
```

- ▶  Base des programmes spaghetti! ☺
- ▶ Abus dangereux pour la santé (mentale)

# Le plan

- 1 Introduction
  - Bibliographie
  - Petite histoire
- 2 Langage
  - Généralités
- 3 Un ordinateur ? Mais qu'est-ce ?
  - Les bases de la numérologie
  - Calcul binaire
- 4 Déclarations
  - Généralités
  - Types des objets
  - Portée
- 5 Expressions
  - Sémantique opérateurs
  - Coercition de type
- 6 Instructions
- 7 Préprocesseur
- 8 Bibliothèques
- 9 Modèle mémoire & allocation
- 10 Entrées-sorties
- 11 GNOME
- 12 Éditeur exemple d'Emacs
  - Tuning de son clavier
- 13 Compilation
- 14 Modularisation
- 15 Mise au point
  - Dévermineur (debugger)
  - Analyse mémoire
- 16 Extensions
  - Du C pour le graphisme
  - C++
- 17 Délires
  - Le bêtisier
  - Concours de programmes incompréhensibles
- 18 Conclusion
- 19 Index
- 20 Table des matières

# Préprocesseur

(I)

## Besoins

- Avoir du code portable
  - Factoriser du code redondant à grain plus petit que fonction
  - Compilation conditionnelle
  - Transformations de programmes
- ↗ Utiliser un préprocesseur entre programme source et compilateur lui-même

# Préprocesseur C (CPP)

(I)

- Syntaxe orientée ligne (contrairement au C lui-même)
- Ligne commence par #
- Possible de mettre une instruction sur plusieurs lignes avec \

```
1 #define uHISTO_ADDRESS(a,ux,uy) \\\n2 (((a)<<(LOG2_XAXIS_HISTO_DEF)\\\n3 +(LOG2_YAXIS_HISTO_DEF))\\\n4 +((x)<<LOG2_YAXIS_HISTO_DEF)\\\n +(y))
```

- Pour mise au point, possibilité d'avoir source dans préprocesseur seulement avec gcc -E
- Fonction d'Emacs pour afficher région passé dans préprocesseur

# Inclusion de fichiers

(I)

```
1 /* Inclus ici un fichier d'entête standard */
2 #include <stdio.h>
3 // Et ici un fichier plus personnel au projet :
4 #include "correle.h"
5 // On peut inclure un fichier défini dans une macro
6 #define portabilite "blue-gene.h"
7 #include portabilite
```

- Récursion possible : un fichier peut aussi inclure d'autres fichiers
-  aux récursions avec inclusions infinies

# Macroconstantes et macrofonctions

(I)

- Définitions et usages

- Macroconstantes

```
1 #define EQUIV_HISTO_TYPE unsigned short int
2 typedef EQUIV_HISTO_TYPE histo;
```

- Macrofonctions

```
1 #define HISTO_ADDRESS(a,ux,uy) \
2 (((a) << LOG2_XAXIS_HISTO_DEF) \
3 + ((x) << LOG2_YAXIS_HISTO_DEF)) \
4 + (y))
5
6 #ifdef RUN_LT_TP
7 #define quality_increment(v) v+=quality
8 #else
9 #define quality_increment(v) v+=1
10#endif
11 quality_increment(rc_c);
```

# Macroconstantes et macrofonctions

(II)

- Effacement

```
1 #undef PLURAL
```

- Souvent identifiants en majuscule pour indiquer macro
- Possibilités de définir et annuler des macro à la compilation

```
1 gcc -DNDEBUG -DBITSIZE=64 -D'X2(a)=(a)*(a)' -U RIEN ...
```

- ⚠️ ⚠️ Macrofonction  $\neq$  fonction

```
1 #define CARRE(x) x*x
```

```
#define MEILLEUR_CARRE(x) (x)*(x)
```

```
3 inline double SUPER_CARRE(double x){ return x*x; }
```

- CARRE(a++ + 1) est remplacé par a++ + 1 \* a++ + 1
- MEILLEUR\_CARRE(a++ + 1) est remplacé par (a++ + 1) \* (a++ + 1)
- SUPER\_CARRE(a++ + 1) donne le bon résultat

# Macroconstantes et macrofonctions

(III)

- Générateur de constantes chaînes de caractères avec #

```
1 #define CHAINE(x) #x
```

CHAINE(toto) est transformé en "toto" Plus subtil si on veut la valeur d'une macro :

```
1 /* To generate a string from a macro : */
2 #define STRINGIFY_SECOND_STAGE(symbol) #symbol
3 /* If not using this 2-stage macro evaluation, the generated string
4 is the value of the macro but the name of the macro... Who said
5 simple language ? :-/ */
6 #define STRINGIFY(symbol) STRINGIFY_SECOND_STAGE(symbol)
```

- Concaténation avec ##

```
1 #define DEFTYPE(x,y) typedef x y x##y
2 DEFTYPE(long,int);
 longint ui;
```

# Macroconstantes et macrofonctions

(IV)

- Possible d'avoir des macros avec nombre variables d'arguments (`__VA_ARGS__...`)
- Nombreuses constantes prédéfinies pour messages de débogage sympathiques ou tests de version
  - ▶ `__FILE__` nom du fichier courant
  - ▶ `__TIME__` heure de compilation
  - ▶ `__STDC_VERSION__` version du C
  - ▶ ...

Permet de faire des messages de type

# Macroconstantes et macrofonctions

(V)

```
1 #ifndef NDEBUG
2 #define __assert(ex)——{ if (!(ex)){ \
3 void)fprintf(stderr , \
4 "Assertion failed : file \"%s\" , line %d\n" , \
5 __FILE__ , __LINE__); \
6 abort();}}
7 #define assert(ex)——_assert(ex)
8 #else
9 #define __assert(ex)
define assert(ex)
10#endif
```

- `cpp -dM` montre définition de toutes macros dans un programme
- $\exists$  Pseudo-variable (pas macro !) `__func__` nom fonction courante (nouveau en C99)

# Compilation conditionnelle

(I)

- Possibilité de sauter des lignes en fonction de conditions
- Instructions
  - ▶ #if condition booléenne
  - ▶ #ifdef si une macro est définie
  - ▶ #ifndef si une macro n'est pas définie
  - ▶ #else
  - ▶ #endif
  - ▶ #elif condition booléenne
- Pour écrire condition, nombreux opérateurs du C disponibles plus opérateur define pour tester la définition d'une macro



# Compilation conditionnelle

(II)

```

1 #if defined DEBUG_FCP_SOFT || defined VERIFY_MATCH
2 int schpkno = 0;
3 PLURAL int a, u, v;
4 #endif
5 #if NUMBER_OF_HISTOGRAMS != 1
6 /* Shifted version : */
7 compute_histogram_adress(h_p,
8 shifted_histogram_address,
9 shifted_a_xxs_yys);
10 #else
11 h_p = NULL;
12 #endif

```

- Très pratique pour éviter inclusions sans fin de fichiers. Exemple de /usr/include/stdio.h :

```

1 #ifndef _STDIO_H
2 #define _STDIO_H 1
3 ...
4 #endif /* !_STDIO_H */

```

# Le plan

- 1 Introduction
  - Bibliographie
  - Petite histoire
- 2 Langage
  - Généralités
- 3 Un ordinateur ? Mais qu'est-ce ?
  - Les bases de la numérologie
  - Calcul binaire
- 4 Déclarations
  - Généralités
  - Types des objets
  - Portée
- 5 Expressions
  - Sémantique opérateurs
  - Coercition de type
- 6 Instructions
- 7 Préprocesseur
- 8 Bibliothèques
- 9 Modèle mémoire & allocation
- 10 Entrées-sorties
- 11 GNOME
- 12 Éditeur exemple d'Emacs
  - Tuning de son clavier
- 13 Compilation
- 14 Modularisation
- 15 Mise au point
  - Dévermineur (debugger)
  - Analyse mémoire
- 16 Extensions
  - Du C pour le graphisme
  - C++
- 17 Délires
  - Le bêtisier
  - Concours de programmes incompréhensibles
- 18 Conclusion
- 19 Index
- 20 Table des matières

# Bibliothèques

(I)

- Peu de choses dans langage C autre que minimum
- Tout le « superflu » passé en bibliothèques standard
  - ▶ Entrées-sorties
  - ▶ Traitement chaînes de caractères normaux & larges
  - ▶ Fonctions mathématiques
  - ▶ Gestion fonctions à nombre variable d'arguments
  - ▶ ...
- Nombreuses autres bibliothèques disponibles, souvent aussi disponibles dans d'autres langages : factorisation apprentissage
  - ▶ POSIX : interaction avec système d'exploitation
  - ▶ Interfaces utilisateurs
    - GNOME
    - KDE <http://developer.kde.org> avec Qt
  - ▶ Bibliothèques scientifiques
  - ▶ Bibliothèques métiers
  - ▶ ...

# Bibliothèque standard

(I)

- Beaucoup de fonctions, types, variables, macros...
- 24 entêtes .h déclarants diverses fonctions, variables et structures de données
- Possible d'inclure ces fichiers dans n'importe quel ordre et autant de fois qu'on veut ☺
- Implémentation libre : la GNU libc info libc qui inclut ISO C, POSIX.2 et quelques extensions
- Multistandard selon macros qu'on définit
  - ▶ \_POSIX\_SOURCE
  - ▶ \_POSIX\_C\_SOURCE
  - ▶ \_BSD\_SOURCE
  - ▶ \_LARGEFILE64\_SOURCE
  - ▶ \_FILE\_OFFSET\_BITS
  - ▶ \_ISOC99\_SOURCE
  - ▶ \_GNU\_SOURCE : la totale !

# Bibliothèque standard

(II)

## ► \_THREAD\_SAFE

- Beaucoup trop de choses pour tout retenir...
- ...mais savoir catégories qui existent et savoir trouver information
- info libc et man
- Éviter de redéfinir des fonctions standard, sauf pour obscurcir un programme ☺



# Contenu bibliothèque standard glibc

(I)

- Gestion des erreurs
- Détails du langage tels que
  - ▶ NULL
  - ▶ Nombre d'arguments variables dans fonctions
  - ▶ Bornes des types arithmétiques
  - ▶ Assertions et débogage
- Gestion de la mémoire, allocation mémoire virtuelle
- Gestion des caractères, classification et conversion
- Manipulation de chaînes de caractères et tableaux
- Entrées-sorties et fichiers sur des *streams* (FILE \*) avec tampons intermédiaires pour limiter appels systèmes
- Entrées-sorties bas niveau : appels systèmes qui opèrent sur descripteurs de fichiers
- Manipulation de systèmes de fichiers



# Contenu bibliothèque standard glibc

(II)

- Tuyaux (*pipes*) et files d'attentes (*FIFOs*) de communication inter-processus
- Sockets (tuyaux de communication inter-machines)
- Gestion des terminaux d'interaction (écran, clavier, fenêtre)
- Fonctions mathématiques
- Fonctions arithmétiques de bas niveau (lire/écrire nombres)
- Recherches d'éléments et tris de tableaux
- Recherches avec expressions régulières sur chaînes
- Temps et dates
- Manipulation de jeux de caractères
- Adaptation des programmes aux différents pays et langues
- Sauts non-locaux pour aller plus loin qu'un simple goto (exceptions...)
- Gestion des signaux

# Contenu bibliothèque standard glibc (III)

- Gestion des arguments des programmes (définition, outils d'analyse...)
- Création de processus
- Contrôle de processus, de groupes de processus
- Systèmes de nommages (NSS) : où sont définis utilisateurs...
- Gestion utilisateurs et groupes
- Gestion configuration systèmes (paramètres matériels)

# Le plan

- 1 Introduction
  - Bibliographie
  - Petite histoire
- 2 Langage
  - Généralités
- 3 Un ordinateur ? Mais qu'est-ce ?
  - Les bases de la numérologie
  - Calcul binaire
- 4 Déclarations
  - Généralités
  - Types des objets
  - Portée
- 5 Expressions
  - Sémantique opérateurs
  - Coercition de type
- 6 Instructions
- 7 Préprocesseur
- 8 Bibliothèques
- 9 Éditeur exemple d'Emacs
  - Tuning de son clavier
- 10 Compilation
- 11 Modularisation
- 12 Mise au point
  - Dévermineur (debugger)
  - Analyse mémoire
- 13 Extensions
  - Du C pour le graphisme
  - C++
- 14 Délires
  - Le bêtisier
  - Concours de programmes incompréhensibles
- 15 Conclusion
- 16 Index
- 17 Table des matières

# Modèle mémoire du C

(I)

- Variables globales
  - ▶ Non initialisées : ne prennent pas de place dans fichier programme exécutable mais initialisées à 0 au démarrage
  - ▶ Initialisées : prennent place dans fichier programme exécutable
- Variables automatiques : locales aux fonctions ↵ allouer dans la *pile* (pour gérer récursion et pouvoir empiler appels)
- Variables statiques dans fonction : comme variable globale (initialisation et valeur qui subsiste d'un appel de fonction à l'autre)
- Comment créer des objets dynamiques à la demande et qui ont une durée de vie autre que syntaxique ?
  - ▶ ↵ Zone d'allocation dynamique séparée : *tas*
  - ▶ Utilise puissance des pointeurs du C pour gérer cette *zone* ☺
  - ▶ De manière générale on peut avoir autant d'espaces de stockage qu'on veut (cf `mmap()`)

# Allocation dynamique

(1)

Fonctions disponibles en mettant dans son programme

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

- Allocation mémoire

```
1 void* malloc(size_t size);
 struct s* p = malloc(sizeof(struct s));
```

- ▶ Mémoire non initialisée
- ▶  Si plus de mémoire, renvoie NULL

- Allocation mémoire de taille multiple

```
1 void* calloc(size_t nmemb, size_t size);
2 // p pointe vers vecteur de taille 10 de s :
 struct s* v[] = calloc(10, sizeof(struct s));
```

Mémoire initialisée à 0 aussi

- Libération de la mémoire (remise dans pot commun)

# Allocation dynamique

(II)

```
1 void free(void *ptr);
 free(v);
```

 Suppose que zone allouée par un `malloc` ou consort dans le passé

- Changement de taille

```
1 void *realloc(void *old_ptr, size_t new_size);
```

- Utilisation de vérifications plus poussées si variable d'environnement `MALLOC_CHECK_...` existe

`man malloc` et surtout `info libc`

# Chaînes de caractères

(1)

- Existent en version caractères longs en remplaçant dans nom str par wcs
- Mesure la longueur d'une chaîne

```
1 size_t strlen(const char* chaîne)
 ^
```

La longueur n'est pas stockée dans les chaînes en C...  
Mesure ≡ chercher le premier caractère nul ('\0')

! Très coûteux!

- Copie de mémoire

```
1 void* memcpy(void* restrict T0,
2 const void* restrict FROM, size_t SIZE)
```

- Copie de chaîne

# Chaînes de caractères

(II)

```
1 char*strcpy(char*restrictT0,
2 const char*restrictFROM)
```

- ⚠️⚠️⚠️ À place disponible endroit de destination... Source principale de piratage/bug ☺
- Copie limitée de chaîne de caractère

```
1 char*strncpy(char*restrictT0,
2 const char*restrictFROM, size_t uSIZE)
```

Utile pour limiter copie à taille de destination. ⚠️ Ne pas oublier place pour '\0' final...

☰ Nombreuses autres fonctions

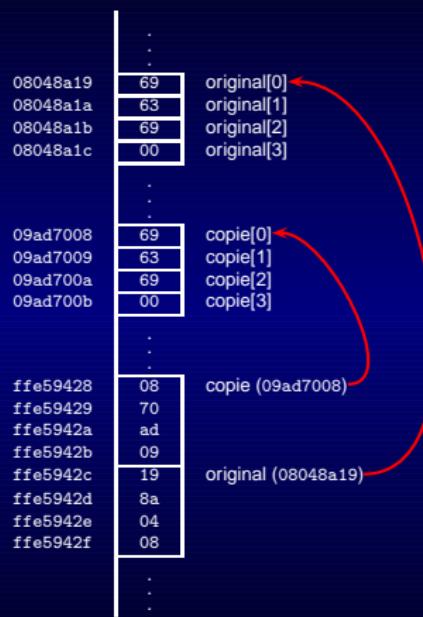
# strup : LA fonction la plus utile du C (I)

- Souvent besoin de dupliquer une chaîne (original alloué automatiquement sur pile ou non modifiable...)
- Demande mesure longueur, allouer avec `malloc()` sans oublier '\0' final puis recopier
- `strup()` fait tout ça d'un coup! ☺

```
1 char*original="ici";
2 char*copie=strup(original);
```

# strup : LA fonction la plus utile du C

(II)



# Le plan

- 1 Introduction
  - Bibliographie
  - Petite histoire
- 2 Langage
  - Généralités
- 3 Un ordinateur ? Mais qu'est-ce ?
  - Les bases de la numérologie
  - Calcul binaire
- 4 Déclarations
  - Généralités
  - Types des objets
  - Portée
- 5 Expressions
  - Sémantique opérateurs
  - Coercition de type
- 6 Instructions
- 7 Préprocesseur
- 8 Bibliothèques
- 9 Modèle mémoire & allocation
- 10 Entrées-sorties
- 11 GNOME
- 12 Éditeur exemple d'Emacs
  - Tuning de son clavier
- 13 Compilation
- 14 Modularisation
- 15 Mise au point
  - Dévermineur (debugger)
  - Analyse mémoire
- 16 Extensions
  - Du C pour le graphisme
  - C++
- 17 Délires
  - Le bêtisier
  - Concours de programmes incompréhensibles
- 18 Conclusion
- 19 Index
- 20 Table des matières

# Entrées-sorties sur *stream*

(I)

- Fonctions de haut niveau

- ▶ Accès tamponnés pour éviter de faire trop d'appels systèmes coûteux
- ▶ Possible de faire un `fflush()` explicite pour forcer écriture sur la sortie
- ▶ Notion de flux d'E/S (*streams*)
  - `stdin` : entrée standard (clavier, fichier...)
  - `stdout` : sortie standard (console, fenêtre, fichier...)
  - `stderr` : messages d'erreur, non tamponnée
- ▶ Possibilité de « dé-lire » des caractères

- Création (ouverture) de flux

```
1 FILE *mon_fichier = fopen("fichier", "r");
```

- Fermeture

```
1 fclose(mon_fichier);
```

# Entrées-sorties sur *stream*

(II)

- Sortie d'un caractère sur stdout

```
1 int_putchar_(int_c)
```

- Sortie d'une chaîne de caractères sur stdout suivie d'un saut de ligne

```
1 int_puts_(char_*chaîne)
```

- Lecture caractère sur stdin

```
1 int_getchar_(void)
```

- Lecture ligne caractère sur un flux

```
1 char_*fgets_(char_*s, int_COUNT, FILE_*STREAM)
```

COUNT évite attaques par débordement...

- Lire la documentation...



# Affichage formaté avec printf

(1)

- Déclarée avec `#include <stdio.h>`
- Permet d'afficher dans un certain format

```
1 int printf(const char *format, ...);
```

- ▶ Nombre variable d'arguments, utilisés selon format
- ▶ Renvoie nombre de caractères écrits
- ▶ Affiche le format en traitant des balises %

```
1 printf("Le caractère %c a pour code octal %o,\n"
 " décimal %d et hexadécimal %x\n", 'c', 'c', 'c', 'c');
```

- Quelques balises

- ▶ %d affiche en décimal
- ▶ %o affiche en octal
- ▶ %x affiche en hexadécimal minuscule
- ▶ %f affiche un nombre flottant
- ▶ %e affiche un nombre flottant en notation scientifique

# Affichage formaté avec printf

(II)

- ▶ %c affiche un caractère
- ▶ %s affiche une chaîne de caractère
- ▶ %% affiche un % ☺
- Largeur du champ précisée par un nombre après %
- Précision avec nombre après .  
%10.4f affiche potentiellement 10 chiffres avec 4 chiffres après la virgule
- Modificateur de taille tel que
  - ▶ %lld pour un **long long int**
  - ▶ %ls pour une chaîne en UNICODE
- Nombreuses options, très puissant lorsqu'on maîtrise
-   Toujours utiliser un format avec printf() et consort. Ne pas utiliser à la place de puts() ou autre car risque de bug ou de piratage

```
1 printf(chaine);
```

# Affichage formaté avec printf

(III)

Si pirate peut fournir chaîne, il peut mettre des % dedans... ☺ Trou de sécurité classique

- ∃ Nombreuses autres versions vers d'autres fichiers (`fprintf()`) ou vers chaînes de caractères (`sprintf()`)

- Extension GNU

- `int asprintf(char **PTR, const char *TEMPLATE, ...)` Formate la sortie dans PTR après l'avoir alloué à la bonne taille. Simplifie bien les choses et supprime des débordements de tampon

- Possible d'étendre les formats !

```
man printf et surtout info libc
```

# Lecture formatée avec scanf

(I)

- Formats de type `printf` mais pour écrire dans arguments
-  Comme `scanf()` doit écrire dans arguments, passer leur adresse !

```
1 void
2 readarray_(double_*array , int n)
3 {
4 int i;
5 for (i=0; i<n; i++)
6 if (scanf ("%lf", &(array[i])) != 1)
7 invalid_input_error();
8 }
```

- Permet de ne lire que si le format correspond. *Renvoie nombre d'éléments lus*
- Extrêmement pratique pour faire de l'analyse syntaxique simple (avant d'utiliser générateurs d'analyseurs à la `flex/bison`)
- Version permettant de lire dans des fichiers, depuis des chaînes...

# goto non locaux

(I)

- Un fonction enregistre l'état du programme dans une structure de donnée
- Une autre permet de restaurer cet état ↗ joue le rôle d'un goto non local
- ⚡ aux nombreuses contraintes d'utilisation
- Pratique pour implémenter des exceptions, récupérer les erreurs dans les programmes...

```
1 #include <stdio.h>
2 #include <malloc.h>
3 #include <setjmp.h>
4 /* Stocke l'état du processeur */
5 jmp_buf buff;
6
7 void throw_here(){
8 /* Reviens dans l'état sauvegardé en renvoyant 1 au setjmp
9 donc saute dans le bloc suivant le setjmp() */
10 longjmp(buff, 1);
```

# goto non locaux

(II)

```
}

12 int main()
14 /* La subtilité est dans les volatiles nécessaires
15 pour prévenir le compilateur qu'il va se passer
16 des choses bizarres et donc que les variables mises
17 en registre doivent être synchronisées souvent
18 avec la mémoire, sinon plantage en -O3... */
19 char * volatile t=NULL; /* des données... */
20 char ** volatile ptr_tab=NULL;

22 /* Sauvegarde dans buff l 'état courant : */
23 if (setjmp(buff))
24 // Affiche des infos
25 fprintf(stderr, "pointer_t: %p\n", t);
26 fprintf(stderr, "pointer_ptr_tab: %p\n", ptr_tab);
27 fflush(stderr);
28 free(ptr_tab[1]);
29 free(ptr_tab);
30 return 10;
```

# goto non locaux

(III)

```
 }/* Ce bloc simule un catch () à la Java... */
32
 /* Ici commence l'équivalent du bloc du try à la Java */
34
 t=(char*)malloc(100);
36 fprintf(stderr,"pointer_t : %p\n",t);

38 ptr_tab=(char**)malloc(sizeof(char*)*2);
39 fprintf(stderr,"pointer_ptr_tab : %p\n",ptr_tab);
40
 ptr_tab[1]=t;
42
 /* Lance une exception à la Java */
44 throw_here();

46 // On ne devrait jamais arriver jusqu'ici ...
47 return 0;
48 }
```



# Le plan

- 1 Introduction
  - Bibliographie
  - Petite histoire
- 2 Langage
  - Généralités
- 3 Un ordinateur ? Mais qu'est-ce ?
  - Les bases de la numérologie
  - Calcul binaire
- 4 Déclarations
  - Généralités
  - Types des objets
  - Portée
- 5 Expressions
  - Sémantique opérateurs
  - Coercition de type
- 6 Instructions
- 7 Préprocesseur
- 8 Bibliothèques
- 9 Modèle mémoire & allocation
- 10 Entrées-sorties
- 11 GNOME
- 12 Éditeur exemple d'Emacs
  - Tuning de son clavier
- 13 Compilation
- 14 Modularisation
- 15 Mise au point
  - Dévermineur (debugger)
  - Analyse mémoire
- 16 Extensions
  - Du C pour le graphisme
  - C++
- 17 Délires
  - Le bêtisier
  - Concours de programmes incompréhensibles
- 18 Conclusion
- 19 Index
- 20 Table des matières

# GNOME

(I)

- Système de fenêtrage & applications bien intégrés
- Pour simplifier développement, reposent sur nombreuses bibliothèques regroupées dans <http://www.gtk.org> (origine : GIMP Toolkit (GNU Image Manipulation Program))
  - ▶ Multi-langages de programmation  
<http://www.gtk.org/bindings.html>
  - ▶ Multi-OS et multi-système graphique : X11/Unix, Windows, *Framebuffer* (écran brut)
  - ▶ Glib : bas niveau : gestion structures de données en C, portabilité, processus légers, gestion d'événements, objets
  - ▶ GDK & GDKPixBuf : affichages graphiques
  - ▶ GTK : objets et gadgets graphiques : menus, boutons...
  - ▶ Pango : affichage des textes en graphique
  - ▶ ATK : accessibilité
  - ▶ GObject : couche objet indépendante du langage (utile en C !)

Glade : constructeur interactif d'interface générant du XML interprété par bibliothèque <http://glade.gnome.org>

# Glib

(I)

- Sous Linux/Debian

```
1 apt-get install libglib2.0-0 libglib2.0-doc libglib2.0-0-dbg \
2 libglib2.0-data libglib2.0-dev
```



# Le plan

- 1 Introduction
  - Bibliographie
  - Petite histoire
- 2 Langage
  - Généralités
- 3 Un ordinateur ? Mais qu'est-ce ?
  - Les bases de la numérologie
  - Calcul binaire
- 4 Déclarations
  - Généralités
  - Types des objets
  - Portée
- 5 Expressions
  - Sémantique opérateurs
  - Coercition de type
- 6 Instructions
- 7 Préprocesseur
- 8 Bibliothèques
- 9 Éditeur exemple d'Emacs
  - Tuning de son clavier
- 10 Compilation
- 11 Modularisation
- 12 Mise au point
  - Dévermineur (debugger)
  - Analyse mémoire
- 13 Extensions
  - Du C pour le graphisme
  - C++
- 14 Délires
  - Le bêtisier
  - Concours de programmes incompréhensibles
- 15 Conclusion
- 16 Index
- 17 Table des matières

# Éditeur de texte

(I)

- Besoin d'un éditeur de texte pour taper les sources du programme
- Augmentation de la productivité
  - ▶ Modes spécialisés pour langages (navigateurs de classes, coloration syntaxique, remise en forme...)
  - ▶ Mode compilation/correction d'erreur
  - ▶ Mode débogueur
- Essayer d'avoir un système portable

## Axiome

Tout élève devrait maîtriser au moins un éditeur de texte avant ce cours...

Eclipse, Kate, vim, Emacs...



# Emacs : LE éditeur

(I)

- Développé depuis années 1970 par Richard M. Stallman (sur TOPS20...)
- Véritable système d'exploitation portable
  - ▶ Gère processus
  - ▶ Gère communication
  - ▶ Programmé en C mais programmable en Emacs Lisp, couche orientée objet
  - ▶ Pas la peine d'apprendre *yet another language*
  - ▶ Peut aussi à servir à éditer des textes
- Beaucoup d'application existent
  - ▶ Correcteur orthographique (`M-x flyspell-mode` ou menu Tools/Spell Checking)
  - ▶ Gestion de courriel et news (GNUS)
  - ▶ Navigateur WWW et éditeur SGML/HTML/WiKi
  - ▶ Serveur WWW
  - ▶ Clients IRC



# Emacs : LE éditeur

(II)

- ▶ Tableur
  - ▶ Mode LaTeX quasi-WYSIWYG
  - ▶ Connexions à distance
  - ▶ Shell de commande
  - ▶ Lecture de documentation et manuels
  - ▶ Comparaison et fusion interactives de textes et répertoires (menus Tools/Compare et Tools/Compare)
  - ▶ Calendrier et agenda
  - ▶ Calculatrice
  - ▶ Logiciel de dessin ASCII-art ☺
  - ▶ Jeux (menu Tools/Games) et documentation
  - ▶ Psychanalyste ( $M-x$  doctor) ☺
  - ▶ Traduction en morse ( $M-x$  morse-region)
  - ▶ Économiseurs d'écran ( $M-x$  zone)
  - ▶ Gestion de sessions, de fenêtres
- Fonctionne sur tous systèmes d'exploitations communs

# Emacs : LE éditeur

(III)

- Multiencodage des caractères (menu Options/MULE Multilingual Environment)
- 2 versions différentes
  - ▶ GNU/Emacs
    - Plus avancé en programmation Lisp
    - Mode texte possible (dans terminal, y compris émulation des menus !) en plus du mode graphique
    - Religion de RMS
  - ▶ XEmacs
    - Plus avancé en graphisme
    - Configuration par défaut plus sympathique
- Compromis à trouver par chacun entre un outil qui fait tout et plein d'outils qui ne font rien... ☺
- Plus qu'un éditeur, un état d'esprit ☺



# Un éditeur abordable

(I)

- Au départ était le ~~verb~~ texte...
- Un des premiers éditeur plein écran interactif (197x)...
- ... et un des derniers à être passé au mode graphique (X11) ☺
- Mode texte + nombreuses fonctionnalités ↗ nombreuses commandes sujet de moqueries par adorateurs de sous-éditeurs ☺
- Mode graphique fenêtré : plupart des choses avec menus, touches de fonctions, *tooltips*... ↗ Simple pour commencer!
- Pas de mode commande/mode insertion troublant comme dans *vi*
- Auto-documenté : C-h (*help*)
- Couper-coller configurable en C-x/C-c/C-v pour inconditionnels Windows (menu Options/CUA) (*Common User Access*)
  - ▶  car ces séquences sont déjà utilisées par Emacs standard...
  - ▶ Est-ce bien nécessaire avec mode couper/coller souris (3 boutons) d'Emacs ?

# Concepts de base

(I)

## Importants à connaître

- *Frame* : fenêtre au sens du gestionnaire de fenêtre graphique (bords, décorations...) ou terminal virtuel
  - ▶ Une *frame* contient plusieurs *windows*
  - ▶ Des *frames* peuvent être affichées sur plusieurs écrans ! (menu File/New frame on Display...)
- *Window* : fenêtre au sens d'Emacs
  - ▶ Contenu dans une *frame*
  - ▶ Contient un *buffer*
- *Buffer* : contient du texte généralement associé à un fichier et éditable
  - ▶ Peut être affiché dans une ou plusieurs *windows*
- *Minibuffer* : en général dans la *windows* du bas, sert à répondre à des questions d'Emacs ou à afficher des messages

# Kit de survie

(I)

- C-g (Control G) : interrompt Emacs quand on est perdu (répondre à question incompréhensible... ☺)
- M-x viper-mode (Méta X) passe en mode vi pour les inconditionnels (c-z change entre mode vi et emacs !

*The newest Emacs VI-emulation mode. (also, A VI Plan for Emacs Rescue or the VI PERil.)*

- ☺
- Émulations pour de nombreux autres éditeurs
- Menu Help ☺
- ⚡ Souvent configuration X11 PC définissent mal Méta et il y a confusion touches Méta et Alt ☺



# Kit de survie

(II)

- Usage de la souris très pratique
  - ▶ Bouton gauche souris : sélectionne
  - ▶ Bouton droit souris : fin de la sélection
  - ▶ Double clic bouton droit souris : fin de la sélection et coupe
  - ▶ Bouton du milieu : insère

# Emacs est lourd

(I)

- Énormément de possibilité
- Un vrai système d'exploitation... ↗ temps de *boot* ☺
- Reboote-t-on une machine pour démarrer chaque application ?  
Non... ☺
- Lancer 1 emacs
- Ouvrir autant de *windows* ou *frames* que nécessaire ☺
- Édition client-serveur
  - ▶ Un Emacs attend des demandes d'éditions
    - Rajouter dans son `~/.emacs`
    - 1    (`server-start`)
  - ▶ Une demande d'édition est envoyée par commande `emacsclient`
  - ▶ Quand édition terminée avec Emacs, taper `C-x #` qui débloque `emacsclient`

# Emacs est lourd

(II)

- ▶ Changer éditeur par défaut Unix dans son `~/.bashrc`

```
1 #_Edit_with_emacs_without_booting_a_new_Emacs_if_there_is_one...:-)
2 alias →e=emacsclient —alternate-editor emacs
3 #_Set_the_default_editor
4 export EDITOR="emacsclient —alternate-editor emacs +%d %s"
```

- ▶ Utiliser mode `mozex` de Mozilla par exemple pour sous-traiter formulaires WWW à `emacsclient`

# Vers une analyse grammaticale des commandes Emacs

- Parfois M- fait comme C- mais plus fort ou inversé
  - ▶ C-v : scroll-up
  - ▶ M-v : scroll-down
- C-u inverse ou modifie une commande en passant un argument
- C-x 4 fait quelque chose dans une nouvelle *windows*
- C-x 5 fait quelque chose dans une nouvelle *frame*
- M-1 M-2 M-3 *truc* : applique 123 fois *truc*
- C-c commandes spécifiques au mode courant

# Compilation

(I)

- Menu Tools/Compile...
- Ouvre fenêtre pour voir erreurs de compilation
- Clic avec bouton milieu sur messages d'erreur pour sauter dans fichier correspondant
- Si compilation via connexion distante, récupération du fichier sélectionné !

# Correction erreurs de syntaxe à la volée

(I)

- Inspiré du mode de correction syntaxique M-x `flyspell-mode`
- M-x `flymake-mode`
- Idée : lance constamment des compilations et analyse messages d'erreurs en temps réel ! ☺
- Affiche erreur en rose et message en tooltip
- Nécessite de rajouter une règle de compilation dans son `Makefile du style`

```
1 check-syntax:
2 ----->gcc -o nul -S ${CHK_SOURCES}
```

- Pour activer automatiquement, mettre dans son `~/.emacs`

```
1 (add-hook 'find-file-hooks 'flymake-find-file-hook)
```

# Débogueurs dans Emacs

(I)

- GUD (*Grand Unified Debugger*)
- Offre interface uniforme depuis Emacs à plein de débogueurs pour différents langages
  - ▶ GDB (`M-x gdb`), DBX, SDB, XDB
  - ▶ Perl (`M-x perldb`)
  - ▶ PDB (Python) (`M-x pdb`)
  - ▶ bash debugger (`M-x bashdb`)
  - ▶ JDB (Java)
- Exécution suivie dans buffer source d'Emacs par =>
- Possible de causer directement au débogueur dans buffer GUD
- `M-x gud-tooltip-mode` pour rajouter affichage valeur de variables sous forme de tooltip
- Mode multi-buffer affichant variables, pile, assembleur...

# Tags

(I)

- Entités d'un programme définies dans de nombreux fichiers
- Difficile pour programmeur de trouver définition d'une entité
- Pour s'y retrouver, constitution d'un fichier d'index TAGS
- Outil etags d'indexation compatible avec nombreux langages
- M-. permet de trouver première définition
- Nombreuses autres fonctionnalités
- Rajouter dans Makefile

```
tags:
----->etags ${TOUS_MES_SOURCES}
```

- M-x visit-tags-table permet de choisir une table de tags
- Variable tags-table-list pour utiliser plusieurs tables de tags
- Existe aussi pour vi (ctags)



# Speedbar

(I)

- Rajoute fenêtre affichant informations supplémentaires en fonction mode courant
  - ▶ Affiche tags
  - ▶ Affiche versions
  - ▶ Affiche fichiers
  - ▶ Affiche info
- M-x speedbar

# Édition de fichiers binaires

(I)

- M-x hexl-mode pour entrer, C-c C-c pour sortir
- Affiche à la fois caractères normaux et hexadécimal
- Commandes pour se déplacer à adresse précise
- Commandes pour insérer en décimal, octal, hexadécimal



# Nécessité d'un système de gestion de version (I)

- Tout n'est pas correct du premier coup
- Besoin d'expérimenter plein de choses
- Quelque chose de faux aujourd'hui peut être utile demain
- Vie d'un logiciel ou d'un document compliqué : développement, mise au point, nombreuses versions en circulation, corrections sur de vieilles versions...
- Travail souvent à plusieurs
- Mémoire qui flanche (« pourquoi ai-je modifié ce /etc/apache/httpd.conf ? »)

~ Besoin de garder des traces du passé !

```
1 #ifndef lint
2 static char vcid[] = "%W%";
#endif /* lint */
```

# Nécessité d'un système de gestion de version (II)

Exemple : chaîne de caractères %W% remplacée information sur le fichier par outil de gestion de version et retrouvée par commande what : permet d'avoir version sur chaque unité de compilation (bibliothèques utilisées...)

# Gestion de version avec Emacs

(I)

- Extrêmement pratique d'avoir gestion de version dans éditeur
  - ▶ Récupère une version
  - ▶ Soumet une nouvelle version
  - ▶ Compare différentes versions avec différentes couleurs
  - ▶ Fusionne différentes versions
  - ▶ Édite fichiers pas à jour
- Mode générique de contrôle de version indépendant de l'outil utilisé !
- ↗ Évite d'avoir à trop s'investir dans un outil spécifique
- Menu Tools/Version Control (apprentissage des raccourcis avec le temps...) dont
  - ▶ Register rajoute fichier dans système de gestion de version (choisi par défaut ou le même que d'autres fichiers dans répertoire)
  - ▶ Check In/Out entre une nouvelle version (qui devient non écrivable) ou sort dernière version (écrivable pour édition)



# Gestion de version avec Emacs

(II)

- ▶ Compare with Base Version affiche différences avec version courante
- ▶ Show History affiche historique du fichier
- ▶ VC Directory Listing affiche fichiers en cours de modification et sous contrôle de versions
- ▶ Insert Header rajoute un commentaire approprié propre au type de fichier édité avec information sur la version. Exemple en C

```
/* Id */
```

qui après entrée dans RCS devient

```
1 /*—————$Id: trans.tex,v 1.25 2009/04/20 12:31:39 keryell
```

pour voir version dans le fichier sans outil de gestion de version sous le coude

- ▶ Update ChangeLog crée/met à jour un fichier ChangeLog avec l'historique du fichier dedans. Pour voir historique dans fichier sans besoin outil de gestion de version



# Gestion de version en local

(I)

- SCCS et CSSC : canal historique
- RCS
  - ▶ crée des fichiers `,v` qui contiennent historique des versions
  - ▶ Cachables dans répertoires RCS
  - ▶ Si partage des fichiers en local à plusieurs, bien gérer les droits (en écriture pour groupe ou ACL)
  - ▶ Commandes de base
    - `ci` pour faire un *check-in* : mémorise une nouvelle version
    - `co` pour faire un *check-out* : récupère une version donersionsnée ou dernière version pour éditer
- Subversion peut aussi tourner en local sur machine



# Gestion de version à distance

(I)

- Souvent si développement collaboratif, serveur de gestion de versions distant
- CVS
  - ▶ Historique
  - ▶ Pas de transaction atomique sur plusieurs fichiers
  - ▶ Pas de modification des répertoires possibles
- Subversion

« *If C gives you enough rope to hang yourself, think of Subversion as a sort of rope storage facility.* »  
*Brian W. FITZPATRICK (tiré du livre officiel de Subversion)*

- ▶ Transactions atomiques
- ▶ Modification globale des répertoires possibles
- ▶ Facilités pour travailler en déconnecté



# Gestion de version à distance

(II)

- ∃ Nombreux autres systèmes libres
- ∃ Autres systèmes commerciaux

# Gestion de version avec Emacs pour CVS et SVN (/)

- Modes spécifiques en plus des fonctions de base de gestion de version d'Emacs (menu Tools/Version Control)
- Si éditions simultanées contradictoires : besoin de résoudre des conflits ↗ mode résolution de conflit basé sur mode Emacs Merge
- Pour CVS : mode PCL-CVS
  - ▶ Rajoute menu
  - ▶ Affichage hiérarchie de fichiers avec information sur version
  - ▶ Fonction de résolution de conflits
- Pour SVN (SubVersion) : mode SVN Status
  - ▶ Interface du style de la précédente

# TRAMP

(I)

- Besoin d'éditer des fichiers sur machines distantes
- ↗ TRAMP : *Transparent Remote (file) Access, Multiple Protocol*
- Sait utiliser ssh, rsh, rlogin, telnet, ftp, rcp, scp, rsync, smbclient
- Permet aussi d'éditer fichiers avec d'autres droits (su et sudo) : pratique pour administration système
- Permet de passer par plusieurs méthodes et machines
- Exemples

/MACHINE:LOCALNAME

/USER@MACHINE:/PATH/TO.FILE

/multi:rsh:out@gate:telnet:kai@real.host:/path/to.file

- Gère les versions à distance



# Modes spécifiques à des langages

(I)

- De nombreux modes disponibles pour éditer divers langages et formats de données
- Mode C (`info CC-mode`)
  - ▶ Gère nombreux langages à syntaxe à la C (C++, Objective-C, Java, CORBA IDL, AWK...)
  - ▶ Déplacements dans les structures syntaxiques
  - ▶ Indentation automatique et configurable (de manière interactive) avec nombreux styles prédéfinis (gnu, k&r, bsd, stroustrup...)
  - ▶ Caractères « électriques » : réindentation dès qu'on tape des {}, ;, (, ), etc
  - ▶ Retour chariot automatique, effacement glouton des espaces
  - ▶ Peut afficher région après préprocesseur
  - ▶ Coloration syntaxique configurable

# Le plan

- 1 Introduction
  - Bibliographie
  - Petite histoire
- 2 Langage
  - Généralités
- 3 Un ordinateur ? Mais qu'est-ce ?
  - Les bases de la numérologie
  - Calcul binaire
- 4 Déclarations
  - Généralités
  - Types des objets
  - Portée
- 5 Expressions
  - Sémantique opérateurs
  - Coercition de type
- 6 Instructions
- 7 Préprocesseur
- 8 Bibliothèques
- 9 Éditeur exemple d'Emacs
  - Tuning de son clavier
- 10 Compilation
- 11 Modularisation
- 12 Mise au point
  - Dévermineur (debugger)
  - Analyse mémoire
- 13 Extensions
  - Du C pour le graphisme
  - C++
- 14 Délires
  - Le bêtisier
  - Concours de programmes incompréhensibles
- 15 Conclusion
- 16 Index
- 17 Table des matières

# Retour aux sources : clavier QWERTY

(I)

- *Use the source, Luke*
- Clavier français (AZERTY) pas du tout adapté à la programmation  
~`'',!@#\$%^&\*()\_-+={}[]|\\$\nTaper des chiffres : pas mieux...
- Contorsions horribles des doigts
- Sous Windows c'est pire \\\\\\ ☺
- perl encore pire
- Le langage C (et l'informatique généralement) a été conçu par des gens qui n'avaient pas de clavier français ☺
- ↗ Achetez des claviers US (pas GB ! ☺)
- Encore mieux qu'un clavier de PC US : un clavier de Sun US (touches Copy/Paste/Cut/...)



# Revamper un clavier AZERTY en QWERTY (I)

- Encore plus *hype*!
- Mettez des autocollants sur les touches qui changent entre les 2
- Un clavier AZERTY a une touche de plus qu'un clavier QWERTY
- Utilisation possible de cette 105<sup>ème</sup> touche pour autre chose
- Exemple : remplacer </> par Undo/Redo de Sun (pour Emacs, OpenOffice...)

Fichier /etc/X11/Xmodmap :

```
! Put Undo/Redo on the <> key:
keycode 0x5E = Undo Redo
```

Si cela ne marche pas tout seul en fonction de son gestionnaire de fenêtre :

```
xmodmap /etc/X11/Xmodmap
```



# Des accents !

(I)

- Belle documentation et commentaires en français
- Incompatible avec les claviers QWERTY ?
- Mais comment faire en AZERTY des ÉéÀàÇç ?
- Comment taper toutes les langues du monde « simplement » ?
- Concept de la touche Compose des Sun pour composer des caractères complexes avec Compose+Accent+Lettre ou autre
- Revamper la touche Contrôle de droite par exemple avec dans son /etc/X11/Xmodmap :

! Compose complex characters with Control\_R:

remove Control = Control\_R

keysym Control\_R = Multi\_key

ou encore mieux le compose:rctrl dans transparent suivant



# Touches Meta et Alt

(I)

- Nombreuses touches permettant de modifier le comportement du gestionnaire de fenêtre ou d'un éditeur
- Shift, Control, Alt, Meta, Super, Hyper...
- Sources de blagues racistes anti-Emacsien(ne)s ☺
- Meta et Alt sont les principales en plus de la touche majuscule
- Malheureusement sous PC X11 grosse confusion  
(programmeurs trop jeunes ? ☺)

Besoin de rajouter dans son /etc/X11/xorg.conf ou  
/etc/X11/XF86Config-4



# Touches Meta et Alt

(II)

```
Section "InputDevice"
Identifier "Keyboard0"
Driver "kbd"
Option "XkbRules" "xorg"
Option "XkbModel" "pc105"
Option "XkbLayout" "us"
/usr/share/X11/xkb/rules/base.lst:
altgr-intl us: International (AltGr dead keys)
Option "XkbVariant" "altgr-intl"
Use the RightCtrl key as a compose key to build complex
characters, keep LeftAlt as Alt like traditional X11 unix and
use LeftWindows key as Meta. Put Euro on AltGr E. For fun try
exotic characters on the keypad too:
Option "XkbOptions" "compose:rctrl,altwin:left_meta_win,eurosign:e,keypad:oss"
EndSection
```



# Avec l'expérience

(I)

- Chacun se construit sur le long terme son environnement
- Étudier les raccourcis de son gestionnaire de fenêtre
  - ▶ Choisir éventuellement un autre
  - ▶ Adapter (touche Menu)
- Étudier les raccourcis de son gestionnaire de éditeur
  - ▶ Touche Impression pour la complétion automatique Hippie d'Emacs



# Le plan

- 1 Introduction
  - Bibliographie
  - Petite histoire
- 2 Langage
  - Généralités
- 3 Un ordinateur ? Mais qu'est-ce ?
  - Les bases de la numérologie
  - Calcul binaire
- 4 Déclarations
  - Généralités
  - Types des objets
  - Portée
- 5 Expressions
  - Sémantique opérateurs
  - Coercition de type
- 6 Instructions
- 7 Préprocesseur
- 8 Bibliothèques
- 9 Modèle mémoire & allocation
- 10 Entrées-sorties
- 11 GNOME
- 12 Éditeur exemple d'Emacs
- 13 Tuning de son clavier
- 14 Compilation
- 15 Modularisation
- 16 Mise au point
- 17 Dévermineur (debugger)
- 18 Analyse mémoire
- 19 Extensions
- 20 Du C pour le graphisme
- 21 C++
- 22 Délires
- 23 Le bêtisier
- 24 Concours de programmes incompréhensibles
- 25 Conclusion
- 26 Index
- 27 Table des matières

# Principe

(I)

- Ordinateur capable d'exécuter que instructions machines

```
1 uuuuuldur2, u456(r5)
 uuuuuaddur1, r2, r3
3 uuuuujmpu0x1234
```

- Besoin de traducteur langage haut niveau vers instructions machine  $\equiv$  compilation
  - ▶ Allocation des objets dans mémoire
  - ▶ Génération séquences d'instructions équivalentes au programme C

# Compilateur gcc

(I)

- Compilateur portable de langages style C (C++, Objective C) de la communauté GNU
- Plus accent sur portabilité que sur performances
- Énormément d'options : voir `info gcc`
- Compilation directe (avec édition de lien)

```
1 gcc toto.c -o toto
```

- Simple compilation

```
1 gcc -c toto.c
```

génère un **toto.o**

- Édition de liens

```
1 gcc toto.o titi.o -o toto
```

- Quelques options très utiles

# Compilateur gcc

(II)

- ▶ -g compile avec informations pour debogueur (moins optimisé en général)
- ▶ -Wall avertit de nombreux problèmes
- ▶ -S génère un fichier d'assemblage .s
- ▶ -E génère source après passage dans préprocesseur
- ▶ -O $n$  optimise au niveau  $n$
- ▶ -std=c99 choisit le dialecte

# Makefile

(I)

## Système d'aide à la construction de programmes

- Définit ce qui doit être fait
- Définit comment cela doit être fait
- Construction paresseuse des fichiers indispensable pour gros projets : ne compile que ce qui n'est pas à jour
- Compare date de cible avec dates de sources ↗ si utilisation d'ordinateurs dont horloges mal synchronisées... ↗ utilisation protocoles synchronisation style NTP
- Moteur *make* et fichier de configuration par défaut `makefile` ou `Makefile`
- Première cible trouvée exécutée en premier ↗ `Makefile` minimal pour compiler `toto.c`

```
1 all: toto
```

# Makefile

(II)

- Règles de type

```
1 cible : sources
 -----> action1
3 -----> action2
 -----> ...
```



Actions commencent par une tabulation !

- Règles implicites qui simplifient la tâche

- ▶ Pour compiler un programme C *toto.c* avec `make toto` sera implicitement utilisées les règles

```
1 $(CC) -c $(CPPFLAGS) $(CFLAGS) toto.c
2 $(CC) $(LDFLAGS) toto.o $(LOADLIBES) $(LDLIBS) -o toto
```

- ▶ Pour une édition de lien avec une règle simple

```
1 x : y.o z.o
```

sont compilés automatiquement puis liés les *x.c*, *y.c* et *z.c*

# Makefile

(III)

- Variables pour simplifier/paramétriser *makefiles*

```
1 CFLAGS=-O4 -Wall -g -std=c99
```

Possible de passer en paramètre lors de l'appel

```
1 make 'CFLAGS=-O4 -Wall -g -std=c99' toto
```

voire au niveau variable du shell

```
1 export CFLAGS=' -O4 -Wall -g -std=c99 '
```

Voir info make

# Le plan

- 1 Introduction
  - Bibliographie
  - Petite histoire
- 2 Langage
  - Généralités
- 3 Un ordinateur ? Mais qu'est-ce ?
  - Les bases de la numérologie
  - Calcul binaire
- 4 Déclarations
  - Généralités
  - Types des objets
  - Portée
- 5 Expressions
  - Sémantique opérateurs
  - Coercition de type
- 6 Instructions
- 7 Préprocesseur
- 8 Bibliothèques
- 9 Modèle mémoire & allocation
- 10 Entrées-sorties
- 11 GNOME
- 12 Éditeur exemple d'Emacs
  - Tuning de son clavier
- 13 Compilation
- 14 Modularisation
  - Mise au point
  - Dévermineur (debugger)
  - Analyse mémoire
- 15 Extensions
  - Du C pour le graphisme
  - C++
- 16 Délires
  - Le bêtisier
  - Concours de programmes incompréhensibles
- 17 Conclusion
- Index
- Table des matières

# Modularisation

(I)

- Besoin de gérer la complexité
  - ▶ Lisibilité
  - ▶ Mise au point
  - ▶ Maintenabilité
  - ▶ Réutilisabilité
  - ▶ Portabilité
  - ▶ Extensibilité
- ↗ Diviser pour régner
- ↗ Séparer interface d'implémentation

# Fonctions

(I)

- Factoriser le plus de code dans des fonctions
- Regrouper fonctions reliées par thématique dans mêmes fichiers
- Mettre toutes fonctions internes (non interfaces) en **static**



# Fichiers

(I)

- Unité de compilation en C
- Base de la compilation plus rapide avec `make`
- Base de la visibilité en C car pas de `private` (**static** ou pas)
- Mettre en-têtes de fonctions et variables dans 2 fichiers .h au moins
  - ▶ Fichier définissant interfaces publiques (non **static**) avec fonctions, variables mais aussi types et macros (style `projet.h`)
  - ▶ Fichier définissant objets pour compiler les module avec définition des objets internes : fonctions, variables, types et macros (style `projet-local.h`)
- Génération automatique avec utilisation d'outils comme `cproto` ou `protoize`

# Utiliser des symboles

(I)

- Éviter de mettre des constantes explicites dans programmes

```
1 double v[100][100];
```

- ▶ Programme difficile à comprendre
- ▶ Difficile à modifier, peu adaptable

- Utiliser des constantes factorisées

```
1 enum{TAILLE_VECTEUR=100};
2 #define NOM_PROJET "gros_projet"
3 const double d=3.2;
4 const char *message="ne bougez plus!"
5 double v[TAILLE_VECTEUR][TAILLE_VECTEUR];
```

Éviter le préprocesseur pour ça car pas objet de première classe

# Le plan

- 1 Introduction
  - Bibliographie
  - Petite histoire
- 2 Langage
  - Généralités
- 3 Un ordinateur ? Mais qu'est-ce ?
  - Les bases de la numérologie
  - Calcul binaire
- 4 Déclarations
  - Généralités
  - Types des objets
  - Portée
- 5 Expressions
  - Sémantique opérateurs
  - Coercition de type
- 6 Instructions
- 7 Préprocesseur
- 8 Bibliothèques
- 9 Modèle mémoire & allocation
- 10 Entrées-sorties
- 11 GNOME
- 12 Éditeur exemple d'Emacs
  - Tuning de son clavier
- 13 Compilation
- 14 Modularisation
- 15 Mise au point
  - Dévermineur (debugger)
  - Analyse mémoire
- 16 Extensions
  - Du C pour le graphisme
  - C++
- 17 Délires
  - Le bêtisier
  - Concours de programmes incompréhensibles
- 18 Conclusion
- 19 Index
- 20 Table des matières

# Le plan

- 1 Introduction
  - Bibliographie
  - Petite histoire
- 2 Langage
  - Généralités
- 3 Un ordinateur ? Mais qu'est-ce ?
  - Les bases de la numérologie
  - Calcul binaire
- 4 Déclarations
  - Généralités
  - Types des objets
  - Portée
- 5 Expressions
  - Sémantique opérateurs
  - Coercition de type
- 6 Instructions
- 7 Préprocesseur
- 8 Bibliothèques
- 9 Éditeur exemple d'Emacs
  - Tuning de son clavier
- 10 Compilation
- 11 Modularisation
- 12 Mise au point
  - Dévermineur (debugger)
  - Analyse mémoire
- 13 Extensions
  - Du C pour le graphisme
  - C++
- 14 Délires
  - Le bêtisier
  - Concours de programmes incompréhensibles
- 15 Conclusion
- 16 Index
- 17 Table des matières

# Débogueur

(I)

- Permet

- ▶ Analyse *post mortem* d'un programme à partir du fichier image mémoire (programme, variables...) core
- ▶ Exécution pas à pas
- ▶ Désassemblage et affichage de variables, mémoire
- ▶ Modification programme et variables
- ▶ Points d'arrêts (*break-points*) à certains endroits du programme (ligne, fonction, adresse...)
- ▶ Arrêt sur conditions mémoires (*watch-points*)
  - Si case mémoire ou variable modifiée, vérifie telle condition...
  - Très pratique pour déverminer problème sordide d'écrasage de case mémoire
  -   Si pas de support matériel, très très lent (exécution pas à pas et test de la condition... ☺)

# Débogueur

(II)

- Nécessite d'avoir dans exécutable (ou ailleurs dans cas de système d'exploitation) table des symboles et informations de débogage (numéros de ligne, noms fichiers sources...) (*stab*) : rajoutés à la compilation avec option `-g`  
~  Programme plus gros, moins optimisé
- Débogage du noyau du système d'exploitation lui-même ? Plus subtil... ☺
  - ▶ Utiliser émulateur de machine, machine virtuelle...
  - ▶ Modifier noyau pour prendre en compte mise au point
    - Mettre une partie du débogueur dans noyau
    - Faire tourner interface de débogage sur autre machine
    - Amorcer machine sur débogueur noyau au lieu de noyau

<http://heather.cs.ucdavis.edu/~matloff/UnixAndC/CLanguage/Debug.html>  
« *Guide to Faster, Less Frustrating Debugging* »

# Débogueur GNU gdb

(I)

*gdb [options] programme core*

- Sait gérer symboliquement nombreux langages
- Options très utiles
  - ▶ *-p process-id* : débogage processus déjà en train de tourner
  - ▶ *-f* : débogage aussi des processus créé par le programme
- Instructions de base
  - ▶ Toutes les commandes sont abréviables
  - ▶ Complétion automatique sur commandes et symboles du programme
  - ▶ *run arguments* : lance programme
  - ▶ *help thème* et *apropos quelque-chose* pour la documentation en ligne
  - ▶ *info* sur aspects du programmes
  - ▶ *show* sur aspects de gdb
  - ▶ *set* permet de modifier comportement et variables du programme

# Débogueur GNU gdb

(II)

- ▶ break pose un point d'arrêt  
Possible de faire point d'arrêt conditionnel mais  performances...
- ▶ watch pose une surveillance de variable ou mémoire   performances...
- ▶ trace met des points d'échantillonnage de variables pour analyse plus tard des valeurs avec tfind, tdump (programmes temps réel...)
- ▶ clear et delete pour supprimer *break-points* et *watch-points*
- ▶ continue l'exécution
- ▶ step avance d'un ou plusieurs pas
- ▶ next comme step mais sans suivre appels de fonctions
- ▶ stepti et nextti : idem mais au niveau instruction machine
- ▶ backtrace ou bt affiche pile des appels de fonctions, option full pour afficher variables locales en plus
- ▶ frame sélectionne un niveau de pile d'appels
- ▶ up et down pour se déplacer dans cette pile
- ▶ list affiche le source du programme

# Débogueur GNU gdb

(III)

- ▶ `print` affiche une variable ou zone mémoire. Nombreux modes disponibles
- ▶ `display` affiche après chaque commande une variable
- ∃ Nombreuses interfaces graphiques ou pas : Eclipse, ddd, Emacs...

Comme d'habitude : `man gdb` et surtout `info gdb`

# Débogueur ddd

(I)

- *Data Display Debugger*
- Surcouche multi-fenêtre à plusieurs débogueur dont gdb
- Affichage des variables de manière graphiques (structures...)



# Le plan

- 1 Introduction
  - Bibliographie
  - Petite histoire
- 2 Langage
  - Généralités
- 3 Un ordinateur ? Mais qu'est-ce ?
  - Les bases de la numérologie
  - Calcul binaire
- 4 Déclarations
  - Généralités
  - Types des objets
  - Portée
- 5 Expressions
  - Sémantique opérateurs
  - Coercition de type
- 6 Instructions
- 7 Préprocesseur
- 8 Bibliothèques
- 9 Éditeur exemple d'Emacs
  - Tuning de son clavier
- 10 Compilation
- 11 Modularisation
- 12 Mise au point
  - Dévermineur (debugger)
  - Analyse mémoire
- 13 Extensions
  - Du C pour le graphisme
  - C++
- 14 Délires
  - Le bêtisier
  - Concours de programmes incompréhensibles
- 15 Conclusion
- 16 Index
- 17 Table des matières

# Mise au point allocation mémoire

(I)

## LA difficulté du C

Assurer mémoire correctement utilisée (allocation, pointeurs...)

- Outil commercial avec interface graphique Purify  
<http://www-306.ibm.com/software/awdtools/purify>
  - ▶ Instrumentation du code lors de l'édition des liens
  - ▶ Détecte
    - Débordements de tableaux et variables, y compris statiques et automatiques (dans pile)
    - Fuites mémoires
    - Lecture de mémoire non allouée
    - Erreur d'allocations/désallocation
  - ▶ Interface graphique pour tracer l'allocation/désallocation de toute case mémoire
- Valgrind <http://en.wikipedia.org/wiki/Valgrind>  
<http://valgrind.org>

# Mise au point allocation mémoire

(II)

- ▶ Déetecte
  - Problèmes d'allocation mémoire
  - Fuite mémoire
  -  Ne détecte pas (encore) débordement variables statiques ou automatiques
  - Précision au bit près
  - Autorise des copies de mémoire non-initialisée tant que pas utiliser dans une expression

- ▶ Différents modules disponibles

MemCheck Déverminage classique de la mémoire

Helgrind Détection d'incohérences dans programmes multi-thread

Massif Analyse comportement du tas

Cachegrind Analyse de comportement de cache pour optimisation. Interface graphique KCacheGrind

```
valgrind --leak-check=yes myprog arg1 arg2
```

- ▶ Quelques options de hacker :



# Mise au point allocation mémoire (III)

```
--malloc-fill=a55a --free-fill=e11e --freelist-vol=1000000000
--track-origins=yes --leak-resolution=high
--num-callers=100 --leak-check=full
```

- ▶ Voir documentation, section *MEMCHECK OPTIONS*.
- ▶ Fait tourner programme dans machine virtuelle et instrumente programme (rajoute aux accès à la mémoire du contrôle ou autre)
- GCC avec *mudflap*
  - ▶ Instrumentation des accès sensibles (pointeurs, tableaux, chaînes de caractères, tas...)
  - ▶ Sous-traitance à bibliothèque *libmudflap*
  - ▶ Vérifie absence d'erreur de mémoire (accès, débordement...) *lors de l'exécution*
  - ▶ Compilation avec option *-fmudflap*
  - ▶ Fonctionnement à l'exécution contrôlé par variable d'environnement *MUDFLAP\_OPTIONS*
- efence - Electric Fence Malloc Debugger
  - ▶ Option *-lefence* à l'édition de lien

# Mise au point allocation mémoire

(IV)

- ▶ Modifie `malloc()` & co
- ▶ Déetecte débordements en lecture et écriture en (dés)alloquant des pages mémoires autour de la zone mémoire ↗ très gourmand en mémoire !
- ▶ Utilisé par Pixar depuis 1987 ☺

Permet de détecter des erreurs non fatales, contrairement aux dévermineurs... avant qu'elles ne le deviennent !



# Le plan

- 1 Introduction
  - Bibliographie
  - Petite histoire
- 2 Langage
  - Généralités
- 3 Un ordinateur ? Mais qu'est-ce ?
  - Les bases de la numérologie
  - Calcul binaire
- 4 Déclarations
  - Généralités
  - Types des objets
  - Portée
- 5 Expressions
  - Sémantique opérateurs
  - Coercition de type
- 6 Instructions
- 7 Préprocesseur
- 8 Bibliothèques
- 9 Modèle mémoire & allocation
- 10 Entrées-sorties
- 11 GNOME
- 12 Éditeur exemple d'Emacs
  - Tuning de son clavier
- 13 Compilation
- 14 Modularisation
- 15 Mise au point
- 16 Dévermineur (debugger)
- 17 Analyse mémoire
- 18 Extensions
  - Du C pour le graphisme
  - C++
- 19 Délires
  - Le bêtisier
  - Concours de programmes incompréhensibles
- 20 Conclusion
- 21 Index
- 22 Table des matières

# Le plan

- 1 Introduction
  - Bibliographie
  - Petite histoire
- 2 Langage
  - Généralités
- 3 Un ordinateur ? Mais qu'est-ce ?
  - Les bases de la numérologie
  - Calcul binaire
- 4 Déclarations
  - Généralités
  - Types des objets
  - Portée
- 5 Expressions
  - Sémantique opérateurs
  - Coercition de type
- 6 Instructions
- 7 Préprocesseur
- 8 Bibliothèques
- 9 Éditeur exemple d'Emacs
  - Tuning de son clavier
- 10 Compilation
- 11 Modularisation
- 12 Mise au point
  - Dévermineur (debugger)
  - Analyse mémoire
- 13 Extensions
  - Du C pour le graphisme
  - C++
- 14 Délires
  - Le bêtisier
  - Concours de programmes incompréhensibles
- 15 Conclusion
- 16 Index
- 17 Table des matières

# CUDA

(I)

<http://developer.nvidia.com>

- Gros besoin de réalisme dans les jeux et applications multimédia
- Cartes graphiques de plus en plus performantes avec beaucoup de mémoire
- Domaine de recherche « fun »
- Besoin de développer de nouvelles versions & application
  - ▶ En interne à nVidia pour bibliothèques et fonctionnalités
    - Rendu de cheveux/poils et coiffeur virtuel  
<http://www.joealter.com>
    - Translucidité
  - ▶ En externe car velléités d'exploiter cette puissance de calcul
- Compilateur
- Environnement d'exécution (*runtime*)
  - ▶ Mesures de performance
- Pour Windows, Linux x86 32 & 64 bits, MacOS X
- Parallélisation sur 2 cartes avec SLI (Scalable Link Interface)

# Nouveaux formats de données

(I)

- Beaucoup de mémoire, mais jamais assez ☺
- Débit importants entre CPU et GPU (PCI X16...) mais jamais assez
- ↗ Types plus petits adaptés à la taille de chaque problèmes
  - ▶ Type `half` flottant : 10 bits de mantisse et 5 bits d'exposant



# Le plan

- 1 Introduction
  - Bibliographie
  - Petite histoire
- 2 Langage
  - Généralités
- 3 Un ordinateur ? Mais qu'est-ce ?
  - Les bases de la numérologie
  - Calcul binaire
- 4 Déclarations
  - Généralités
  - Types des objets
  - Portée
- 5 Expressions
  - Sémantique opérateurs
  - Coercition de type
- 6 Instructions
- 7 Préprocesseur
- 8 Bibliothèques
- 9 Éditeur exemple d'Emacs
  - Tuning de son clavier
- 10 Compilation
- 11 Modularisation
- 12 Mise au point
  - Dévermineur (debugger)
  - Analyse mémoire
- 13 Extensions
  - Du C pour le graphisme
  - C++
- 14 Délires
  - Le bêtisier
  - Concours de programmes incompréhensibles
- 15 Conclusion
- 16 Index
- 17 Table des matières

# C++

... en 1 transparent ☺

- Langage orienté objet basé sur C
  - ▶ Rajoute fonctions dans structures ↵ **classes**
  - ▶ Polymorphisme
  - ▶ Surcharge d'opérateurs du langage (+ - , () []...)
  - ▶ Exceptions
  - ▶ Patrons de types génériques
  - ▶ Espaces de noms
  - ▶ Grosse bibliothèque standard (STL, Boost)
- Encore bien plus subtil que le C...
- En attendant son apprentissage... ∃ groupe de musique C++ dont premier disque est « Orienté Objet »  
<http://www.orientee-objet.com/>,  
<http://www.myspace.com/charlottecplusplus> prochain concert gratuit 13/05/2009, 20h, L'International, Paris ☺

# Le plan

- 1 Introduction
  - Bibliographie
  - Petite histoire
- 2 Langage
  - Généralités
- 3 Un ordinateur ? Mais qu'est-ce ?
  - Les bases de la numérologie
  - Calcul binaire
- 4 Déclarations
  - Généralités
  - Types des objets
  - Portée
- 5 Expressions
  - Sémantique opérateurs
  - Coercition de type
- 6 Instructions
- 7 Préprocesseur
- 8 Bibliothèques
- 9 Modèle mémoire & allocation
- 10 Entrées-sorties
- 11 GNOME
- 12 Éditeur exemple d'Emacs
  - Tuning de son clavier
- 13 Compilation
- 14 Modularisation
- 15 Mise au point
- 16 Dévermineur (debugger)
- 17 Analyse mémoire
- 18 Extensions
  - Du C pour le graphisme
  - C++
- 19 Délires
  - Le bêtisier
  - Concours de programmes incompréhensibles
- 20 Conclusion
- 21 Index
- 22 Table des matières

# Le plan

- 1 Introduction
  - Bibliographie
  - Petite histoire
- 2 Langage
  - Généralités
- 3 Un ordinateur ? Mais qu'est-ce ?
  - Les bases de la numérologie
  - Calcul binaire
- 4 Déclarations
  - Généralités
  - Types des objets
  - Portée
- 5 Expressions
  - Sémantique opérateurs
  - Coercition de type
- 6 Instructions
- 7 Préprocesseur
- 8 Bibliothèques
- 9 Éditeur exemple d'Emacs
  - Tuning de son clavier
- 10 Compilation
- 11 Modularisation
- 12 Mise au point
  - Dévermineur (debugger)
  - Analyse mémoire
- 13 Extensions
  - Du C pour le graphisme
  - C++
- 14 Délires
  - Le bêtisier
  - Concours de programmes incompréhensibles
- 15 Conclusion
- 16 Index
- 17 Table des matières

# L'octal qui passe à la trappe

(I)

```
1 uustrcpy(msg.mtext, "coucou");
uu id=umsgget(k,IPC_CREAT|660);
```

au lieu de

```
1 uustrncpy(msg.mtext, "coucou", sizeof(msg.mtext));
2 uu id=umsgget(k,IPC_CREAT|0660);
```

# Choses à (ne pas) faire

(I)

<http://freeworld.thc.org/root/phun/unmaintain.html>

- « How To Write Unmaintainable Code — Ensure a job for life ☺ », Roedy GREEN, Canadian Mind Products
- *« An early version of this article appeared in Java Developers' Journal (volume 2 issue 6). I also spoke on this topic in 1997 November at the Colorado Summit Conference. It has been gradually growing ever since. »*



# Le plan

- 1 Introduction
  - Bibliographie
  - Petite histoire
- 2 Langage
  - Généralités
- 3 Un ordinateur ? Mais qu'est-ce ?
  - Les bases de la numérologie
  - Calcul binaire
- 4 Déclarations
  - Généralités
  - Types des objets
  - Portée
- 5 Expressions
  - Sémantique opérateurs
  - Coercition de type
- 6 Instructions
- 7 Préprocesseur
- 8 Bibliothèques
- 9 Éditeur exemple d'Emacs
  - Tuning de son clavier
- 10 Compilation
- 11 Modularisation
- 12 Mise au point
  - Dévermineur (debugger)
  - Analyse mémoire
- 13 Extensions
  - Du C pour le graphisme
  - C++
- 14 Délires
  - Le bêtisier
  - Concours de programmes incompréhensibles
- 15 Conclusion
- 16 Index
- 17 Table des matières

# Qu'imprime-ce ?

## Quizz

```
1 main(){printf (&unix["\021%six\012\0"],
2 unix["have"]+"fun"-0x60);}
```

- `unix` est une variable faiblement définie valant 1
- `&unix["\021%six\012\0"]`  $\longleftrightarrow$  `x*(unix["\021%six\012\0"])`  $\longleftrightarrow$  `"%six\012\0"`
- `(unix)["have"]+"fun"-0x60`  $\longleftrightarrow$  `"have"[1]+"fun"-0x60`  $\longleftrightarrow$  `"fun"+(a^2-0x60)`  $\longleftrightarrow$  `"fun"+1`  $\longleftrightarrow$  `"un"`

# Qu'imprime-ce ?

## Quizz

```
1 main(){printf (&unix["\021%six\012\0"],
2 (unix)["have"]+ "fun" -0x60);}
```

- **unix** est une variable faiblement définie valant 1
  - `&unix["\021%six\012\0"]`  $\iff$  `&(*(unix + "\021%six\012\0"))`  $\iff$  `"%six\012\0"`
  - `(unix)[ "have" ]+ "fun" -0x60`  $\iff$  `"have" [1] + "fun" -0x60`  $\iff$  `"fun" + ('a' - 0x60)`  $\iff$  `"fun" + 1`  $\iff$  `"un"`

# Qu'imprime-ce ?

## Quizz

```
1 main(){printf (&unix["\021%six\012\0"],
2 (unix)["have"]+"fun"-0x60);}
```

- `unix` est une variable faiblement définie valant 1
- `&unix["\021%six\012\0"]`  $\iff$  `&(*(unix+"\021%six\012\0"))`  $\iff$  `"%six\012\0"`
- `(unix)[ "have"]+"fun"-0x60`  $\iff$  `"have" [1]+"fun"-0x60`  $\iff$  `"fun"+("a"-0x60)`  $\iff$  `"fun"+1`  $\iff$  `"un"`

# Qu'imprime-ce ?

## Quizz

```
1 main(){printf (&unix["\021%six\012\0"],
2 (unix)["have"]+ "fun" -0x60);}
```

- `unix` est une variable faiblement définie valant 1
- `&unix["\021%six\012\0"]`  $\iff$  `&(*(unix+"\021%six\012\0"))`  $\iff$  `"%six\012\0"`
- `(unix)[ "have" ]+ "fun" -0x60`  $\iff$  `"have" [1]+ "fun" -0x60`  $\iff$  `"fun" + ('a' -0x60)`  $\iff$  `"fun" + 1`  $\iff$  `"un"`

# Concours

- The International Obfuscated C Code Contest

<http://www.de.ioccc.org/years.html>

- ▶ Un programme de génération de dessins stéréoscopiques dont le source est un... dessin stéréoscopique  
<http://www.de.ioccc.org/2001/herrmann2.c>
- ▶ Poésie sur relation amoureuse  
<http://www.ioccc.org/1990/westley.c>
- ▶ Simulateur de vol en forme d'avion  
<http://www.ioccc.org/1998/banks.c>

- Concours sur le thème du bug

<http://worsethanfailure.com/Articles/The-Worse-Than-Failure-Program>



# Retour aux sources de l'informatique (I)

Vous avez désormais des connaissances suffisantes pour :

<http://computing-dictionary.thefreedictionary.com/Use+the+Source+Luke>



# Le plan

- 1 Introduction
  - Bibliographie
  - Petite histoire
- 2 Langage
  - Généralités
- 3 Un ordinateur ? Mais qu'est-ce ?
  - Les bases de la numérologie
  - Calcul binaire
- 4 Déclarations
  - Généralités
  - Types des objets
  - Portée
- 5 Expressions
  - Sémantique opérateurs
  - Coercition de type
- 6 Instructions
- 7 Préprocesseur
- 8 Bibliothèques
- 9 Modèle mémoire & allocation
- 10 Entrées-sorties
- 11 GNOME
- 12 Éditeur exemple d'Emacs
  - Tuning de son clavier
- 13 Compilation
- 14 Modularisation
- 15 Mise au point
  - Dévermineur (debugger)
  - Analyse mémoire
- 16 Extensions
  - Du C pour le graphisme
  - C++
- 17 Délires
  - Le bêtisier
  - Concours de programmes incompréhensibles
- 18 Conclusion
- 19 Index
- 20 Table des matières

# Conclusion

(I)

- Langage puissant & spectre large horizontalement et verticalement
- Langage très (trop ?) subtil
- Déclarations compliquées
- Arithmétique binaire puissante pour optimiser et niveau matériel/électronique
- Pointeurs & allocation mémoire subtiles
- Sert de base à de nombreux autres langages
- ↗ Il faut pratiquer !
- Suite : programmation système et réseau en 3A SLR/IT & Module IAHP du Master Recherche

# Le plan

- 1 Introduction
  - Bibliographie
  - Petite histoire
- 2 Langage
  - Généralités
- 3 Un ordinateur ? Mais qu'est-ce ?
  - Les bases de la numérologie
  - Calcul binaire
- 4 Déclarations
  - Généralités
  - Types des objets
  - Portée
- 5 Expressions
  - Sémantique opérateurs
  - Coercition de type
- 6 Instructions
- 7 Préprocesseur
- 8 Bibliothèques
- 9 Modèle mémoire & allocation
- 10 Entrées-sorties
- 11 GNOME
- 12 Éditeur exemple d'Emacs
  - Tuning de son clavier
- 13 Compilation
- 14 Modularisation
- 15 Mise au point
  - Dévermineur (debugger)
  - Analyse mémoire
- 16 Extensions
  - Du C pour le graphisme
  - C++
- 17 Délires
  - Le bêtisier
  - Concours de programmes incompréhensibles
- 18 Conclusion
- 19 Index
- 20 Table des matières

|                             |    |                              |                               |
|-----------------------------|----|------------------------------|-------------------------------|
| #define                     |    |                              | négation                      |
| définition                  |    | ou logique booléen           | opérateur, 203                |
| préprocesseur, 251          |    | opérateur, 218               | opérateur                     |
| macroconstante              |    | ou logique booléen           | négation, 203                 |
| préprocesseur, 251          |    | opérateur                    | soustraction, 203             |
| macrofonction               | () | appel fonction               | soustraction                  |
| préprocesseur, 251          |    | opérateur, 202               | opérateur, 203                |
| préprocesseur               | *  | opérateur                    | décrémentation                |
| définition, 251             |    | appel fonction, 202          | opération                     |
| macroconstante, 251         |    | --                           | décrémentation, 222           |
| macrofonction, 251          | *  | déréférencement de pointeur  | déréférencement de champ      |
| #define                     |    | opérateur, 230               | opérateur, 197                |
| constante                   |    | multiplication               | opérateur                     |
| préprocesseur, 110, 164     |    | opérateur, 203               | déréférencement de champ, 197 |
| préprocesseur               |    | opérateur                    | opérateur                     |
| constante, 110, 164         |    | déréférencement de pointeur, | déréférencement de champ,     |
| #if                         |    | 230                          | 197                           |
| compilation conditionnelle  |    | multiplication, 203          |                               |
| préprocesseur, 256          |    |                              | champ                         |
| préprocesseur               | +  | addition                     | opérateur, 196                |
| compilation conditionnelle, |    | opérateur, 203               | opérateur                     |
| 256                         |    | addition, 203                | champ, 196                    |
| #include                    |    | opérateur                    | division                      |
| inclusionfichier            |    | incrémentation               | opérateur, 203                |
| préprocesseur, 250          | ++ | opérateur, 222               | opérateur                     |
| préprocesseur               |    | incrémentation               | division, 203                 |
| inclusionfichier, 250       |    | opération                    |                               |
| &                           |    | incrémentation, 222          | inférieur                     |
| et logique                  | ,  | liste d'expressions          | opérateur, 217                |
| opérateur, 204              |    | opérateur, 234               | opérateur                     |
| adresse d'un objet          |    | opérateur                    | inférieur, 217                |
| opérateur, 230              |    | liste d'expressions, 234     | <<                            |
| opérateur                   |    | opération                    | décalage à gauche             |
| et logique, 204             | ,  | opérateur, 234               | opérateur, 205                |
| adresse d'un objet, 230     |    | opérateur                    | opérateur                     |
| &&                          |    | négation                     | décalage à gauche, 205        |
| et logique booléen          |    | opérateur, 77                |                               |
| opérateur, 217              |    | opérateur                    | =                             |
| opérateur                   |    | négation, 77                 | <=                            |
| et logique booléen, 217     | -  |                              | inférieur ou égal             |

|                                          |                                    |                           |
|------------------------------------------|------------------------------------|---------------------------|
| opérateur, 217                           | indice                             | complément restreint      |
| opérateur inférieur ou égal, 217         | opérateur, 198                     | opérateur, 205            |
| =                                        | opérateur indice, 198              | complément restreint      |
| affection opérateur, 45                  | modulo entier                      | opérateur, 77             |
| opérateur affectation, 45                | opérateur, 203                     | négation                  |
| !=                                       | opérateur modulo entier, 203       | opérateur, 205, 217       |
| opérateur test de différence, 217        | et logique &                       | complément restreint, 205 |
| test de différence opérateur, 217        | opérateur, 204                     | complément restreint, 77  |
| ==                                       | opérateur &, 204                   | négation, 205, 217        |
| opérateur test d'égalité, 217            | ou logique exclusif ^              |                           |
| test d'égalité opérateur, 217            | opérateur, 204                     | caractère codage, 73      |
| >                                        | opérateur ^, 204                   | codage caractère, 73      |
| opérateur supérieur, 217                 | et logique booléen &&              | entier int16_t, 105       |
| supérieur opérateur, 217                 | opérateur, 217                     | uint16_t, 105             |
| >=                                       | opérateur &&, 217                  | int16_t entier, 105       |
| opérateur supérieur ou égal, 217         | ou logique booléen                 | uint16_t entier, 105      |
| supérieur ou égal opérateur, 217         | opérateur, 218                     | 32 bits                   |
| >>                                       | opérateur   , 218                  | entier int32_t, 105       |
| décalage à droite opérateur, 205         | ou logique exclusif opérateur, 204 | uint32_t entier, 105      |
| opérateur décalage à droite, 205         | opérateur ou logique exclusif, 204 | int32_t entier, 105       |
| ? :                                      | !                                  | uint32_t entier, 105      |
| expression conditionnelle opérateur, 220 | négation logique opérateur, 217    | 64 bits                   |
| opérateur expression conditionnelle, 220 | opérateur négation logique, 217    | entier int64_t, 105       |
|                                          | ~                                  | uint64_t entier, 105      |
|                                          | !                                  | int64_t entier, 105       |
|                                          | négation logique opérateur, 217    | uint64_t entier, 105      |
|                                          | opérateur négation logique, 217    | 8 bits                    |
|                                          | ~                                  | entier                    |

int8\_t, 105  
 uint8\_t, 105  
 int8\_t  
     entier, 105  
 uint8\_t  
     entier, 105  
  
**addition**  
     +  
         opérateur, 203  
**opérateur**  
     +, 203  
**adresse d'un objet**  
     &  
         opérateur, 230  
**opérateur**  
     &, 230  
**affectation**  
     =  
         opérateur, 45  
**arithmétique**  
     opération, 221  
**effet de bord**  
     opération, 221  
**opérateur**  
     =, 45  
**opération**  
     arithmétique, 221  
     effet de bord, 221  
**variable**, 45  
**affichage**  
     chaîne de caractères  
         puts()puts(), 275  
     puts()puts()  
         chaîne de caractères, 275  
**agrégat**  
     expression  
         élémentaire, 188  
     élémentaire  
         expression, 188  
  
**agrégé**  
     type, 82  
  
**aiguillage**  
     break  
         instruction, 241, 245  
     case  
         instruction, 241  
     default  
         instruction, 241  
     instruction  
         break, 241, 245  
         case, 241  
         default, 241  
         switch(), 241  
     switch()  
         instruction, 241  
  
**alias**  
     attribut  
         restrict, 84, 170  
     restrict  
         attribut, 84, 170  
  
**aliasing**  
     mémoire  
         pointeur, 166, 170  
     pointeur  
         mémoire, 166, 170  
  
**allocation**  
     calloc()  
         mémoire, 267  
     dynamique  
         mémoire, 267  
     malloc()  
         mémoire, 267  
     malloc(int size)  
         mémoire, 163  
  
**matrice**  
     taille variable, 199  
  
**mémoire**  
     calloc(), 267  
     dynamique, 267  
  
**pointeur**  
     type, 267  
     malloc(int size), 163  
  
**pointeur**  
     type, 127  
**taille variable**  
     matrice, 199  
**type**  
     pointeur, 127  
  
**appel fonction**  
     ()  
         opérateur, 202  
**opérateur**  
     (), 202  
  
**arithmétique**  
     affectation  
         opération, 221  
**binaire**  
     multispin-coding, 210  
     optimisation, 207, 210  
     opérateur, 204  
  
**fixe**  
     virgule, 117  
**flottant**  
     non associativité, 116  
     somme, 118  
     type, 83, 111, 116  
  
**integral**  
     type, 83  
**multispin-coding**  
     binaire, 210  
  
**non associativité**  
     flottant, 116  
**optimisation**  
     binaire, 207, 210  
  
**opérateur**  
     binaire, 204  
  
**opération**  
     affectation, 221  
  
**pointeur**  
     type, 232

|                        |         |                          |                        |
|------------------------|---------|--------------------------|------------------------|
| réel                   |         | entier, 101              | représentation         |
| type, 116              |         | rangement                | nombre, 65             |
| somme                  |         | auto, 84, 162            | base 2048              |
| flottant, 118          |         | register, 84, 160        | nombre                 |
| type, 82               |         | static, 84               | représentation, 72     |
| flottant, 83, 111, 116 |         | register                 | représentation         |
| intégral, 83           |         | rangement, 84, 160       | nombre, 72             |
| pointeur, 232          |         | restrict                 | base 256               |
| réel, 116              |         | alias, 84, 170           | nombre                 |
| virgule                |         | short                    | représentation, 71     |
| fixe, 117              |         | entier, 101              | représentation         |
| ASCII                  |         | signed                   | nombre, 71             |
| caractère              |         | entier, 101              | base 65536             |
| codage, 91             |         | static                   | nombre                 |
| codage                 |         | rangement, 84            | représentation, 73     |
| caractère, 91          |         | unsigned                 | représentation         |
| assembleur, 36         |         | entier, 101              | nombre, 73             |
| associativité          |         | volatile                 | baud                   |
| opérateur              |         | volatilité, 84, 161, 165 | nombre                 |
| priorité, 191          |         | volatilité               | représentation, 64     |
| priorité               |         | volatile, 84, 161, 165   | représentation         |
| opérateur, 191         | auto    | attribut                 | nombre, 64             |
| attribut               |         | rangement, 84, 162       | bibliothèque           |
| alias                  |         | rangement                | libc, 11               |
| restrict, 84, 170      |         | attribut, 84, 162        | standard, 260          |
| auto                   |         | attribut                 | binaire                |
| rangement, 84, 162     |         | rangement, 84, 162       | arithmétique           |
| const                  | base 8  | rangement                | multispin-coding, 210  |
| constante, 84, 164     |         | attribut                 | optimisation, 207, 210 |
| constante              |         | rangement                | opérateur, 204         |
| const, 84, 164         |         | attribut                 | constante              |
| entier                 |         | rangement                | expression, 183        |
| long, 101              | base 16 | attribut                 | expression             |
| long long, 101         |         | rangement                | constante, 183         |
| short, 101             |         | attribut                 | logique                |
| signed, 101            |         | rangement                | opérateur, 217         |
| unsigned, 101          |         | attribut                 | multispin-coding       |
| long                   | base 2  | rangement                | arithmétique, 210      |
| entier, 101            |         | attribut                 | nombre                 |
| long long              |         | rangement                | représentation, 65     |

|                        |                       |                 |
|------------------------|-----------------------|-----------------|
| optimisation           | instruction, 50, 243  | char            |
| arithmétique, 207, 210 |                       | type, 99        |
| opérateur              | for                   | CHAR_BIT        |
| arithmétique, 204      | instruction, 51, 244  | taille, 99      |
| logique, 217           | instruction           | chaîne          |
| représentation         | break, 242, 245       | large, 137      |
| nombre, 64, 65         | continue, 242, 245    | type, 134       |
| système                | do, 50, 243           | codage          |
| unité, 65              | for, 51, 244          | 16 bits, 73     |
| unité                  | while, 50, 242        | ASCII, 91       |
| système, 65            | instruction, 50, 242  | ISO-10646, 98   |
| bit                    | branchement           | ISO-646, 91     |
| nombre                 | goto                  | ISO-8859-1, 97  |
| représentation, 64     | instruction, 245      | ISO-8859-15, 97 |
| représentation         | instruction           | latin-1, 97     |
| nombre, 64             | goto, 245             | latin-9, 97     |
| structure              | break                 | UNICODE, 98     |
| type, 144              | aiguillage            | constante       |
| type                   | instruction, 241, 245 | expression, 184 |
| structure, 144         | boucle                | expression      |
| bool                   | instruction, 242, 245 | constante, 184  |
| booléen                | instruction           | intégral        |
| entier, 108            | aiguillage, 241, 245  | type, 83, 90    |
| entier                 | boucle, 242, 245      | ISO-10646       |
| booléen, 108           |                       | codage, 98      |
| booléen                | C++                   | ISO-646         |
| bool                   | langage               | codage, 91      |
| entier, 108            | objet, 351            | ISO-8859-1      |
| entier                 | objet                 | codage, 97      |
| bool, 108              | langage, 351          | ISO-8859-15     |
| intégral               | calloc()              | codage, 97      |
| type, 83, 108          | allocation            | large           |
| type                   | mémoire, 267          | chaîne, 137     |
| intégral, 83, 108      | mémoire               | type, 98, 107   |
| boucle                 | allocation, 267       | wchar_t, 107    |
| break                  | caractère             | latin-1         |
| instruction, 242, 245  | 16 bits               | codage, 97      |
| continue               | codage, 73            | latin-9         |
| instruction, 242, 245  | ASCII                 | codage, 97      |
| do                     | codage, 91            |                 |

|                           |                           |                           |
|---------------------------|---------------------------|---------------------------|
| CHAR_BIT, 99              | valeur, 99                | ASCII                     |
| type                      | valeur                    | caractère, 91             |
| <b>char</b> , 99          | limite, 99                | caractère                 |
| chaîne, 134               | CHAR_BIT                  | 16 bits, 73               |
| intégral, 83, 90          | caractère                 | ASCII, 91                 |
| large, 98, 107            | taille, 99                | ISO-10646, 98             |
| vecteur, 134              | taille                    | ISO-646, 91               |
| UNICODE                   | caractère, 99             | ISO-8859-1, 97            |
| codage, 98                | chaîne                    | ISO-8859-15, 97           |
| vecteur                   | caractère                 | latin-1, 97               |
| type, 134                 | large, 137                | latin-9, 97               |
| wchar_t                   | type, 134                 | UNICODE, 98               |
| large, 107                | large                     | ISO-10646                 |
| case                      | caractère, 137            | caractère, 98             |
| aiguillage                | type                      | ISO-646                   |
| instruction, 241          | caractère, 134            | caractère, 91             |
| instruction               | chaîne de caractères      | ISO-8859-1                |
| aiguillage, 241           | affichage                 | caractère, 97             |
| cast                      | puts()puts(), 275         | ISO-8859-15               |
| conversion explicite      | puts()puts()              | caractère, 97             |
| type, 228                 | affichage, 275            | latin-1                   |
| type                      | chaîne de caractères      | caractère, 97             |
| conversion explicite, 228 | clonage                   | latin-9                   |
| champ                     | _strdup(), 271            | caractère, 97             |
| .                         | longueur                  | UNICODE                   |
| opérateur, 196            | _strlen(), 269            | caractère, 98             |
| opérateur                 | _strdup()                 | coercition                |
| ., 196                    | clonage, 271              | conversion explicite      |
| champ de bits             | _strlen()                 | type                      |
| structure                 | longueur, 269             | conversion explicite, 228 |
| type, 144                 | clavier                   | commentaires, 35          |
| type                      | configuration, 315        | comparaison               |
| structure, 144            | clonage                   | opérateur, 217            |
| <b>char</b>               | chaîne de caractères      | compilateur, 36           |
| caractère                 | _strdup(), 271            | GCC, 323                  |
| type, 99                  | _strdup()                 | gcc, 11                   |
| type                      | chaîne de caractères, 271 | compilation, 322          |
| caractère, 99             | codage                    | make, 325                 |
| char                      | 16 bits                   | méthode, 325              |
| limite                    | caractère, 73             | makefile, 325             |

|                            |                         |                       |
|----------------------------|-------------------------|-----------------------|
| compilation conditionnelle | $\sim$ , 77             | expression            |
| #if                        | complément vrai         | binnaire, 183         |
| préprocesseur, 256         | nombre                  | caractère, 184        |
| préprocesseur              | négatif, 76             | complexe, 184         |
| #if, 256                   | négatif                 | décimale, 183         |
| complexe                   | nombre, 76              | entière, 183          |
| constante                  | complément à 2          | flottante, 184        |
| expression, 184            | nombre                  | hexadécimal, 183      |
| double complex             | négatif, 77             | octale, 183           |
| flottant, 125              | négatif                 | flottante             |
| expression                 | nombre, 77              | expression, 184       |
| constante, 184             | composition             | hexadécimal           |
| float complex              | déclaration             | expression, 183       |
| flottant, 125              | opérateurs, 151         | octale                |
| flottant                   | opérateurs              | expression, 183       |
| double complex, 125        | déclaration, 151        | préprocesseur         |
| float complex, 125         | const                   | #define, 110, 164     |
| long double complex, 125   | attribut                | construction          |
| type, 83, 125              | constante, 84, 164      | expression, 190       |
| long double complex        | constante               | continue              |
| flottant, 125              | attribut, 84, 164       | boucle                |
| type                       | constante               | instruction, 242, 245 |
| flottant, 83, 125          | #define                 | boucle, 242, 245      |
| complément restreint       | préprocesseur, 110, 164 | contrôle de flot, 48  |
| -                          | attribut                | conventions           |
| opérateur, 205             | const, 84, 164          | typographie, 25       |
| nombre                     | binaire                 | conversion            |
| négatif, 205               | expression, 183         | type, 193             |
| négatif                    | caractère               | conversion explicite  |
| nombre, 205                | expression, 184         | coercition            |
| opérateur                  | complexe                | type, 228             |
| ~, 205                     | expression, 184         | coercition, 228       |
| complément restreint       | const                   | conversion explicite  |
| -                          | attribut, 84, 164       | cast                  |
| opérateur, 77              | décimale                | type, 228             |
| nombre                     | expression, 183         | type                  |
| négatif, 75                | entière                 | cast, 228             |
| négatif                    | expression, 183         | <court>               |
| nombre, 75                 |                         | <copy>                |
| opérateur                  |                         | <print>               |

|                      |                           |                             |
|----------------------|---------------------------|-----------------------------|
| entier               | allocation                | initialisation              |
| short int, 101       | mémoire, 267              | variable, 43, 175           |
| short int            | mémoire                   | objet                       |
| entier, 101          | allocation, 267           | place, 159                  |
| ddd                  | débogage                  | opérateurs                  |
| débogueur, 340       | débogueur                 | composition, 151            |
| default              | mise au point, 335        | place                       |
| aiguillage           | mise au point             | objet, 159                  |
| instruction, 241     | débogueur, 335            | portée, 155                 |
| instruction          | débogueur                 | variable, 43, 84            |
| aiguillage, 241      | DDD, 340                  | initialisation, 43, 175     |
| division             | débogage                  | décrémentation              |
| /                    | mise au point, 335        | --                          |
| opérateur, 203       | GDB, 337                  | opération, 222              |
| opérateur            | gdb, 12                   | opération                   |
| /, 203               | mise au point             | --, 222                     |
| do                   | début                     | définition                  |
| boucle               | main()                    | #define                     |
| instruction, 50, 243 | programme, 32–34, 36, 198 | préprocesseur, 251          |
| instruction          | programme                 | préprocesseur               |
| boucle, 50, 243      | main(), 32–34, 36, 198    | #define, 251                |
| double               | décalage à droite         | dénormalisé                 |
| double               | >>                        | flottant                    |
| flottant, 111        | opérateur, 205            | nombre, 121                 |
| flottant             | opérateur                 | nombre                      |
| double, 111          | >, 205                    | flottant, 121               |
| double               | décalage à gauche         | déréférencement de pointeur |
| double               | <<                        | *                           |
| flottant, 111        | opérateur, 205            | opérateur, 230              |
| flottant             | opérateur                 | opérateur                   |
| double, 111          | <, 205                    | *, 230                      |
| double complex       | décimale                  | déréférencement de champ    |
| complexe             | constante                 | ->                          |
| flottant, 125        | expression, 183           | opérateur, 197              |
| flottant             | expression                | opérateur                   |
| complexe, 125        | constante, 183            | ->, 197                     |
| durée de vie         | déclaration               | déverminage                 |
| objet, 160           | composition               | mémoire, 342                |
| dynamique            | opérateurs, 151           | Purify, 342                 |

|                      |                    |                    |
|----------------------|--------------------|--------------------|
| Purify               | unsigned, 101      | nombre, 62         |
| mémoire, 342         | bool               | short              |
| Valgrind             | booléen, 108       | attribut, 101      |
| mémoire, 342         | booléen            | short int          |
|                      | bool, 108          | court, 101         |
| effet de bord, 31    | court              | signed             |
| affectation          | short int, 101     | attribut, 101      |
| opération, 221       | enum               | très long          |
| opération            | énumération, 109   | long long int, 101 |
| affectation, 221     | int                | type               |
| else                 | normal, 101        | intégral, 83, 101  |
|                      | int16_t            | non signé, 101     |
| instruction          | 16 bits, 105       | uint16_t           |
| test, 49, 239        | int32_t            | 16 bits, 105       |
| test                 | 32 bits, 105       | uint32_t           |
|                      | int64_t            | 32 bits, 105       |
| Emacs                | 64 bits, 105       | uint64_t           |
|                      | int8_t             | 64 bits, 105       |
| texte                | 8 bits, 105        | uint8_t            |
| éditeur, 288         | intégral           | 8 bits, 105        |
| éditeur              | type, 83, 101      | unsigned           |
|                      | large              | attribut, 101      |
| texte, 288           | wint_t, 107        | wint_t             |
| emacs                | long               | large, 107         |
|                      | long int, 101      | énumération        |
| éditeur de texte, 12 | long               | enum, 109          |
| entier               | attribut, 101      | entière            |
| 16 bits              | long int           | constante          |
|                      | long, 101          | expression, 183    |
| int16_t, 105         | long long          | expression         |
| uint16_t, 105        | attribut, 101      | constante, 183     |
| 32 bits              | long long int      | entrée             |
|                      | attribut, 101      | format             |
| int32_t, 105         | très long, 101     | scanf(), 279       |
| uint32_t, 105        | nombre             | scanf()            |
| 64 bits              | représentation, 62 | format, 279        |
|                      | non signé          | entrée-sortie      |
| int64_t, 105         | type, 101          | fclose()           |
| uint64_t, 105        | normal             | fermeture, 274     |
| 8 bits               | int, 101           | fermeture          |
|                      | représentation     | fclose()           |
| attribut             |                    | fermeture          |
|                      |                    | fclose()           |
| long, 101            |                    | fermeture          |
| long long, 101       |                    | fclose()           |
| short, 101           |                    | fermeture          |
| signed, 101          |                    | fclose()           |

|                                   |                                           |                                     |
|-----------------------------------|-------------------------------------------|-------------------------------------|
| <code>fclose()</code> , 274       | littérale élémentaire, 183                | virgule arithmétique, 117           |
| <code>fopen()</code>              | <code>lvalue</code> élémentaire, 189      | <code>float</code> flottant         |
| ouverture, 274                    | octale constante, 183                     | petit, 111                          |
| ouverture                         | <code>symbolique</code> élémentaire, 187  | flottant, 111                       |
| <code>fopen()</code> , 274        | <code>type</code> , 173                   | <code>float complex</code> complexe |
| <code>stream</code> , 274         | élémentaire agrégat, 188                  | flottant, 125                       |
| <code>enum</code>                 | L-valeur, 189                             | complexe, 125                       |
| entier                            | littérale, 183                            | flottant                            |
| énumération, 109                  | <code>lvalue</code> , 189                 | arithmétique                        |
| énumération                       | symbolique, 187                           | non associativité, 116              |
| entier, 109                       | expression conditionnelle                 | somme, 118                          |
| <code>expression</code> , 28, 182 | ? :                                       | <code>type</code> , 83, 111, 116    |
| agrégat                           | opérateur, 220                            | <code>complex</code>                |
| élémentaire, 188                  | opérateur                                 | double complex, 125                 |
| <code>binaire</code>              | ? :, 220                                  | float complex, 125                  |
| constante, 183                    | extension                                 | long double complex, 125            |
| caractère                         | nombre précision, 77                      | type, 83, 125                       |
| constante, 184                    | précision nombre, 77                      | <code>double</code>                 |
| complexe                          | <code>fclose()</code>                     | double, 111                         |
| constante, 184                    | entrée-sortie fermeture, 274              | double                              |
| constante                         | fermeture entrée-sortie, 274              | double complex                      |
| binaire, 183                      | fermeture entrée-sortie                   | complexe, 125                       |
| caractère, 184                    | entrée-sortie <code>fclose()</code> , 274 | dénormalisé                         |
| complexe, 184                     | fermeture entrée-sortie, 274              | nombre, 121                         |
| décimale, 183                     | FIBONACCI, 155                            | <code>float</code>                  |
| entière, 183                      | fixe arithmétique                         | petit, 111                          |
| flottante, 184                    | virgule, 117                              | float complex                       |
| hexadécimal, 183                  |                                           | complexe, 125                       |
| octale, 183                       |                                           | long double                         |
| <code>construction</code> , 190   |                                           | étendu, 111                         |
| décimale                          |                                           | long double complex                 |
| constante, 183                    |                                           | complexe, 125                       |
| entière                           |                                           | nombre                              |
| constante, 183                    |                                           |                                     |
| flottante                         |                                           |                                     |
| constante, 184                    |                                           |                                     |
| hexadécimal                       |                                           |                                     |
| constante, 183                    |                                           |                                     |
| L-valeur                          |                                           |                                     |
| élémentaire, 189                  |                                           |                                     |

|                            |                            |                      |
|----------------------------|----------------------------|----------------------|
| normalisé, 120             | scanf(), 279               | représentation       |
| non associativité          | fprintf                    | nombre, 70           |
| arithmétique, 116          | sortie, 276                |                      |
| normalisé                  | scanf()                    | identificateur       |
| nombre, 120                | entrée, 279                | nom                  |
| petit                      | sortie                     | objet, 85            |
| float, 111                 | fprintf, 276               | objet                |
| réel                       | free()                     | nom, 85              |
| type, 83, 111              | libération                 | if                   |
| somme                      | mémoire, 267               | instruction          |
| arithmétique, 118          | mémoire                    | test, 49, 239        |
| type                       | libération, 267            | test                 |
| arithmétique, 83, 111, 116 | GCC                        | instruction, 49, 239 |
| complexe, 83, 125          | compilateur, 323           | inclusionfichier     |
| réel, 83, 111              | gcc                        | #include             |
| étendu                     | compilateur, 11            | préprocesseur, 250   |
| long double, 111           | GDB                        | préprocesseur        |
| flottante                  | débogueur, 337             | #include, 250        |
| constante                  | gdb                        | incomplet            |
| expression, 184            | débogueur, 12              | fonction             |
| expression                 | gettimeofday, 39           | type, 150            |
| constante, 184             | globale                    | pointeur             |
| fonction, 30, 330          | variable, 46               | type, 149            |
| incomplet                  | GNU                        | structure            |
| type, 150                  | système d'exploitation, 23 | type, 149            |
| paramètre, 47              | goto                       | type, 149            |
| type, 82, 147              | branchement                | fonction, 150        |
| incomplet, 150             | instruction, 245           | pointeur, 149        |
| fopen(                     | instruction                | structure, 149       |
| entrée-sortie              | branchement, 245           | incomplets           |
| ouverture, 274             |                            | type, 82             |
| ouverture                  | hexadécimal                | incomplète           |
| entrée-sortie, 274         | constante                  | initialisation       |
| for                        | expression, 183            | variable, 176        |
| boucle                     | expression                 | variable             |
| instruction, 51, 244       | constante, 183             | initialisation, 176  |
| instruction                | nombre                     | incrémentation       |
| boucle, 51, 244            | représentation, 70         | ++                   |
| format                     |                            | opération, 222       |
| entrée                     |                            | opérations           |

|                   |                      |                    |          |              |         |
|-------------------|----------------------|--------------------|----------|--------------|---------|
| ++                | 222                  | continue           | 242, 245 | boucle       | 50, 242 |
| indentation       |                      | do                 | 50, 243  | int          |         |
| programme         |                      | for                | 51, 244  | entier       |         |
| présentation      | 55                   | while              | 50, 242  | normal       | 101     |
| présentation      |                      | branchement        |          | limite       |         |
| programme         | 55                   | goto               | 245      | valeur       | 101     |
| indice            | []                   | break              |          | normal       |         |
|                   |                      | aiguillage         | 241, 245 | entier       | 101     |
|                   | opérateur, 198       | boucle             | 242, 245 | valeur       |         |
| opérateur         | [] , 198             | case               |          | limite       | 101     |
| infixe            | opérateur, 190       | aiguillage         | 241      | int16_t      |         |
| inférieur         | <                    | continue           |          | 16 bits      |         |
|                   | opérateur, 217       | boucle             | 242, 245 | entier       | 105     |
|                   | opérateur            | default            |          | entier       |         |
|                   | <, 217               | aiguillage         | 241      | 16 bits, 105 |         |
| inférieur ou égal | <=                   | do                 |          | int32_t      |         |
|                   | opérateur, 217       | boucle             | 50, 243  | 32 bits      |         |
|                   | opérateur            | else               |          | entier       | 105     |
|                   | <=, 217              | test               | 49, 239  | entier       |         |
| initialisation    |                      | for                |          | 32 bits      | 105     |
|                   | déclaration          | boucle             | 51, 244  | int64_t      |         |
|                   | variable, 43, 175    | goto               |          | 64 bits      |         |
|                   | incomplète           | branchement        | 245      | entier       | 105     |
|                   | variable, 176        | if                 |          | 64 bits, 105 |         |
|                   | variable             | test               | 49, 239  | int8_t       |         |
|                   | déclaration, 43, 175 | programme          |          | 8 bits       |         |
|                   | incomplète, 176      | présentation       |          | entier       | 105     |
| inline            | 39                   | programme          | 52       | entier       |         |
| instruction       | 28, 236              | retour de fonction |          | 8 bits, 105  |         |
|                   | aiguillage           | return             | 245      | intégral     |         |
|                   | break, 241, 245      | return             |          | arithmétique |         |
|                   | case, 241            | retour de fonction | 245      | type         | 83      |
|                   | default, 241         | saut               | 245      | booléen      |         |
|                   | switch(), 241        | switch()           |          | type         | 83, 108 |
| boucle            |                      | aiguillage         | 241      | caractère    |         |
|                   | break, 242, 245      | test               |          | type         | 83, 90  |
|                   |                      | else               | 49, 239  | entier       |         |
|                   |                      | if                 | 49, 239  | type         | 83, 101 |
|                   |                      | while              |          |              |         |

|                      |                    |                     |
|----------------------|--------------------|---------------------|
| arithmétique, 83     | chaîne, 137        | valeur, 101         |
| booléen, 83, 108     | type, 98, 107      | valeur              |
| caractère, 83, 90    | wchar_t, 107       | char, 99            |
| entier, 83, 101      | chaîne             | int, 101            |
| énumération, 83, 109 | caractère, 137     | long int, 101       |
| énumération          | entier             | long long int, 101  |
| type, 83, 109        | wint_t, 107        | short int, 101      |
| ISO-10646            | type               | linéarisation       |
| caractère            | caractère, 98, 107 | type                |
| codage, 98           | wchar_t            | vecteur, 129        |
| codage               | caractère, 107     | vecteur             |
| caractère, 98        | wint_t             | type, 129           |
| ISO-646              | entier, 107        | liste d'expressions |
| caractère            | latin-1            | ,                   |
| codage, 91           | caractère          | opérateur, 234      |
| codage               | codage, 97         | opérateur           |
| caractère, 91        | codage             | ,, 234              |
| ISO-8859-1           | caractère, 97      | littérale           |
| caractère            | latin-9            | expression          |
| codage, 97           | caractère          | élémentaire, 183    |
| codage               | codage, 97         | élémentaire         |
| caractère, 97        | codage             | expression, 183     |
| ISO-8859-15          | caractère, 97      | locale              |
| caractère            | libc               | variable, 46        |
| codage, 97           | bibliothèque, 11   | logique             |
| codage               | libération         | binaire             |
| caractère, 97        | free()             | opérateur, 217      |
| L-valeur             | mémoire            | opérateur           |
| expression           | free(), 267        | binaire, 217        |
| élémentaire, 189     | limite             | long                |
| élémentaire          | char               | entier              |
| expression, 189      | valeur, 99         | long int, 101       |
| langage              | int                | long int            |
| C++                  | valeur, 101        | entier, 101         |
| objet, 351           | long int           | attribut            |
| objet                | valeur, 101        | entier, 101         |
| C++, 351             | long long int      | entier              |
| large                | valeur, 101        | attribut, 101       |
| caractère            | short int          | long int            |

|                           |                           |                            |
|---------------------------|---------------------------|----------------------------|
| entier                    | lvalue                    | taille variable            |
| long, 101                 | expression                | allocation, 199            |
| long                      | élémentaire, 189          | mise au point              |
| entier, 101               | élémentaire               | débogage                   |
| long double               | expression, 189           | débogueur, 335             |
| flottant                  | macroconstante            | débogueur                  |
| étendu                    | #define                   | débogage, 335              |
| flottant, 111             | préprocesseur, 251        | modulo entier              |
| long double complex       | préprocesseur             | %                          |
| complexe                  | #define, 251              | opérateur, 203             |
| flottant, 125             | macrofonction             | opérateur                  |
| flottant                  | #define                   | %, 203                     |
| complexe, 125             | préprocesseur, 251        | modèle                     |
| long int                  | préprocesseur             | mémoire, 266               |
| limite                    | #define, 251              | mommage                    |
| valeur, 101               | main()                    | type                       |
| valeur                    | début                     | typedef, 171               |
| limite, 101               | programme, 32–34, 36, 198 | typedef                    |
| long long                 | programme                 | type, 171                  |
| attribut                  | début, 32–34, 36, 198     | mots-clés, 87              |
| entier, 101               | make, 12                  | MULTICS                    |
| entier                    | compilation, 325          | système d'exploitation, 17 |
| attribut, 101             | make, 37                  | multiplication             |
| long long int             | Makefile, 37              | *                          |
| entier                    | makefile                  | opérateur, 203             |
| très long, 101            | compilation, 325          | opérateur                  |
| limite                    | malloc()                  | *, 203                     |
| valeur, 101               | allocation                | multispin-coding           |
| très long                 | mémoire, 267              | arithmétique               |
| entier, 101               | mémoire                   | binaire, 210               |
| valeur                    | allocation, 267           | binaire                    |
| limite, 101               | malloc(int size)          | arithmétique, 210          |
| longjmp()                 | allocation                | mémoire                    |
| saut non local, 280       | mémoire, 163              | aliasing                   |
| longueur                  | mémoire                   | pointeur, 166, 170         |
| chaîne de caractères      | allocation, 163           | allocation                 |
| strlen(), 269             | matrice                   | calloc(), 267              |
| strlen()                  | allocation                | dynamique, 267             |
| chaîne de caractères, 269 | taille variable, 199      | malloc(), 267              |

|                    |                           |                   |
|--------------------|---------------------------|-------------------|
| calloc()           | représentation, 73        | base 8, 68        |
| allocation, 267    |                           | base 16, 70       |
| dynamique          | baud                      | base 2, 65        |
| allocation, 267    | représentation, 64        | base 2048, 72     |
| déverminage, 342   | binaire                   | base 256, 71      |
| Purify, 342        | représentation, 64, 65    | base 65536, 73    |
| Valgrind, 342      |                           | baud, 64          |
| free()             | bit                       | binaire, 64, 65   |
| libération, 267    | représentation, 64        | bit, 64           |
| libération         | complément restreint      | entier, 62        |
| free(), 267        | négatif, 205              | hexadécimal, 70   |
| malloc()           | complément restreint      | négatif, 75       |
| allocation, 267    | négatif, 75               | octal, 68         |
| malloc(int size)   | complément vrai           | signe             |
| allocation, 163    | négatif, 76               | négatif, 77       |
| modèle, 266        | complément à 2            | non associativité |
| pointeur           | négatif, 77               | arithmétique      |
| aliasing, 166, 170 | dénormalisé               | flottant, 116     |
| Purify             | flottant, 121             | flottant          |
| déverminage, 342   | entier                    | arithmétique, 116 |
| Valgrind           | représentation, 62        | non finie         |
| déverminage, 342   | extension                 | structure         |
| nom                | précision, 77             | type, 139         |
| identificateur     | flottant                  | type              |
| objet, 85          | dénormalisé, 121          | structure, 139    |
| objet              | normalisé, 120            | non signé         |
| identificateur, 85 | hexadécimal               | entier            |
| nombre             | représentation, 70        | type              |
| base 8             | normalisé                 | entier, 101       |
| représentation, 68 | flottant, 120             | int               |
| base 16            | négatif                   | entier, 101       |
| représentation, 70 | complément restreint, 205 | normal            |
| base 2             | complément restreint, 75  | entier            |
| représentation, 65 | complément vrai, 76       | int               |
| base 2048          | complément à 2, 77        | entier, 101       |
| représentation, 72 | représentation, 75        | normalisé         |
| base 256           | signe, 77                 | flottant          |
| représentation, 71 | octal                     | nombre, 120       |
| base 65536         | représentation, 68        | nombre            |
|                    | précision                 | flottant, 120     |
|                    | extension, 77             | nombre, 120       |
|                    | représentation            | flottant, 120     |

|                      |                           |                         |                                  |
|----------------------|---------------------------|-------------------------|----------------------------------|
| négatif              |                           | durée de vie, 160       | déréférencement de pointeur, 230 |
| complément restreint | nombre, 205               | declaration             | multiplication, 203              |
| complément restreint | nombre, 75                | identificateur          | + addition, 203                  |
| complément vrai      | nombre, 76                | langage                 | ,                                |
| complément à 2       | nombre, 77                | nom                     | - liste d'expressions, 234       |
| nombre               | complément restreint, 205 | place                   | - négation, 77                   |
|                      | complément restreint, 75  | déclaration, 159        | - négation, 203                  |
|                      | complément vrai, 76       | octal                   | soustraction, 203                |
|                      | complément à 2, 77        | nombre                  | -> déréférencement de champ, 197 |
| représentation       | représentation, 75        | représentation          | .                                |
|                      | signe, 77                 | nombre, 68              | champ, 196                       |
| représentation       | nombre, 75                | octale                  | / division, 203                  |
| signe                | nombre, 77                | constante               | < inférieur, 217                 |
| négation             |                           | expression              | << décalage à gauche, 205        |
| -                    | opérateur, 77             | octet, 81               | <= inférieur ou égal, 217        |
| -                    | opérateur, 203            | ompilation, 37          | = affectation, 45                |
| -                    | opérateur, 205, 217       | optimisation            | != test de différence, 217       |
| opérateur            | -, 77                     | arithmétique            | == test d'égalité, 217           |
|                      | -, 203                    | binaire, 207, 210       | > supérieur, 217                 |
|                      | -, 205, 217               | binaire                 | >= supérieur ou égal, 217        |
| négation logique     | !                         | opérateur               | >> décalage à droite, 205        |
|                      | opérateur, 217            | &                       | ? :                              |
| opérateur            | !, 217                    | et logique, 204         | expression conditionnelle,       |
|                      |                           | adresse d'un objet, 230 |                                  |
| objet, 81            | C++                       | &&                      |                                  |
|                      | langage, 351              | et logique booléen, 217 |                                  |
|                      |                           |                         |                                  |
|                      |                           | ou logique booléen, 218 |                                  |
|                      |                           | ()                      |                                  |
|                      |                           | appel fonction, 202     |                                  |
|                      |                           | *                       |                                  |

|                                |                                       |                                   |
|--------------------------------|---------------------------------------|-----------------------------------|
| □                              | complément restreint<br>~, 77         | taille objet, 223                 |
| %                              | division<br>/, 203                    | soustraction<br>-, 203            |
| modulo entier, 203             | décalage à droite<br>>>, 205          | supérieur<br>>, 217               |
| et logique<br>&, 204           | décalage à gauche<br><<, 205          | supérieur ou égal<br>>=, 217      |
| ou logique exclusif<br>~, 204  | déréférencement de pointeur<br>*, 230 | taille objet<br>sizeof(), 223     |
| et logique booléen<br>&&, 217  | déréférencement de champ<br>->, 197   | test d'égalité<br>==, 217         |
| ou logique booléen<br>  , 218  | expression conditionnelle<br>?:, 220  | test de différence<br>!=, 217     |
| ~                              | indice<br>[], 198                     | opérateurs                        |
| ou logique exclusif, 204       | infixe, 190                           | composition                       |
| !                              | inférieur<br><, 217                   | déclaration, 151                  |
| négation logique, 217          | inférieur ou égal<br><=, 217          | déclaration                       |
| -                              | liste d'expressions<br>,, 234         | composition, 151                  |
| complément restreint, 205      | logique<br>binaire, 217               | opération                         |
| complément restreint, 77       | modulo entier<br>%, 203               | ++<br>incrémantation, 222         |
| négation, 205, 217             | multiplication<br>*, 203              | --<br>décrémantation, 222         |
| addition<br>+, 203             | négation<br>~, 77                     | affectation                       |
| adresse d'un objet<br>&, 230   | ~, 203                                | arithmétique<br>arithmétique, 221 |
| affectation<br>=, 45           | ~, 205, 217                           | effet de bord, 221                |
| appel fonction<br>( ), 202     | négation logique<br>!, 217            | arithmétique<br>affectation, 221  |
| arithmétique<br>binaire, 204   | postfixe, 190                         | décrémantation<br>--, 222         |
| associativité<br>priorité, 191 | priorité, 191                         | effet de bord<br>affectation, 221 |
| binaire<br>arithmétique, 204   | associativité, 191                    | incrémantation<br>++, 222         |
| comparaison, 217               | préfixe, 190                          | ouverture                         |
| complément restreint<br>~, 205 | sizeof()                              | entrée-sortie<br>fopen(), 274     |

|                      |                            |                        |                                 |
|----------------------|----------------------------|------------------------|---------------------------------|
| paramètre            |                            | opérateur, 190         | compilation conditionnelle, 256 |
| fonction, 47         |                            |                        |                                 |
| passage en paramètre |                            | format                 | #include                        |
| type                 |                            | sortie, 276            | inclusionfichier, 250           |
| vecteur, 131         |                            | sortie                 | compilation conditionnelle      |
| vecteur              |                            | format, 276            | #if, 256                        |
| type, 131            |                            |                        | constante                       |
| petit                |                            | priorité               | #define, 110, 164               |
|                      | float                      | associativité          | définition                      |
|                      | flottant, 111              | opérateur, 191         | #define, 251                    |
|                      | flottant                   | opérateur, 191         | inclusionfichier                |
|                      | float, 111                 | associativité, 191     | #include, 250                   |
| place                |                            | procédure, 30          | macroconstante                  |
|                      | déclaration                | programme, 32          | #define, 251                    |
|                      | objet, 159                 | début                  | macrofonction                   |
|                      | objet                      | main(), 32–34, 36, 198 | #define, 251                    |
|                      | déclaration, 159           | indentation            | présentation                    |
| pointeur             |                            | présentation, 55       | indentation                     |
|                      | aliasing                   | instruction            | programme, 55                   |
|                      | mémoire, 166, 170          | présentation, 52       | instruction                     |
|                      | allocation                 | main()                 | programme, 52                   |
|                      | type, 127                  | début, 32–34, 36, 198  | programme                       |
|                      | arithmétique               | présentation           | indentation, 55                 |
|                      | type, 232                  | indentation, 55        | instruction                     |
|                      | incomplet                  | instruction            | instruction, 52                 |
|                      | type, 149                  |                        | Purify, 342                     |
|                      | mémoire                    | précision              | déverminage                     |
|                      | aliasing, 166, 170         | extension              | mémoire, 342                    |
|                      | type, 82, 126              | nombre, 77             | mémoire                         |
|                      | allocation, 127            | nombre                 | déverminage, 342                |
|                      | arithmétique, 232          | extension, 77          | puts() (puts())                 |
|                      | incomplet, 149             | préfixe                | affichage                       |
|                      | void, 149                  | opérateur, 190         | chaîne de caractères, 275       |
|                      | void                       | préprocesseur, 36, 248 | chaîne de caractères            |
|                      | type, 149                  | #define                | affichage, 275                  |
| portée               |                            | définition, 251        |                                 |
|                      | déclaration, 155           | macroconstante, 251    | rangement                       |
| POSIX                |                            | macrofonction, 251     | attribut                        |
|                      | système d'exploitation, 23 | #define                | auto, 84, 162                   |
| postfixe             |                            | constante, 110, 164    | register, 84, 160               |
|                      |                            | #if                    | static, 84                      |

**auto**  
 attribut, 84, 162  
**register**  
 attribut, 84, 160  
**static**  
 attribut, 84  
**register**  
 attribut  
 rangement, 84, 160  
 rangement  
 attribut, 84, 160  
**représentation**  
 base 8  
 nombre, 68  
 base 16  
 nombre, 70  
 base 2  
 nombre, 65  
 base 2048  
 nombre, 72  
 base 256  
 nombre, 71  
 base 65536  
 nombre, 73  
 baud  
 nombre, 64  
 binaire  
 nombre, 64, 65  
 bit  
 nombre, 64  
 entier  
 nombre, 62  
 hexadécimal  
 nombre, 70  
 nombre  
 base 8, 68  
 base 16, 70  
 base 2, 65  
 base 2048, 72  
 base 256, 71

**restrict**  
 alias  
 attribut, 84, 170  
 attribut  
 alias, 84, 170  
**retour de fonction**  
 instruction  
 return, 245  
 return  
 instruction, 245  
**return**  
 instruction  
 retour de fonction, 245  
 retour de fonction  
 instruction, 245

**saut**  
 récursion, 155  
 récursivité, 155  
**réel**  
 arithmétique  
 type, 116  
 flottant  
 type, 83, 111  
**type**  
 arithmétique, 116  
 flottant, 83, 111

**base** 65536, 73  
**baud**, 64  
**binaire**, 64, 65  
**bit**, 64  
**entier**, 62  
 hexadécimal, 70  
 négatif, 75  
 octal, 68  
 nombre, 75  
**octal**  
 nombre, 68

**négatif**  
**nombre**, 75  
**octal**  
 nombre, 68

**scope**  
 variable, 46

**setjmp()**  
 saut non local, 280

**short**  
 attribut  
 entier, 101  
**short int**  
 court  
 entier  
 attribut, 101  
 entier  
 court, 101  
**short int**  
 limite  
 valeur, 101  
**valeur**  
 limite, 101

**signe**  
 nombre  
 négatif, 77  
 négatif  
 nombre, 77

**signed**  
 attribut  
 entier, 101  
 entier  
 attribut, 101

**sizeof()**

|                           |                        |                      |
|---------------------------|------------------------|----------------------|
| opérateur                 | structure              | GNU, 23              |
| taille objet, 223         | bit                    | MULTICS, 17          |
| taille objet              | type, 144              | POSIX, 23            |
| opérateur, 223            | champ de bits          | UNIX, 18             |
| somme                     | type, 144              |                      |
| arithmétique              | incomplet              | tabulation, 55       |
| flottant, 118             | type, 149              | taille               |
| flottant                  | non finie              | caractère            |
| arithmétique, 118         | type, 139              | CHAR_BIT, 99         |
| sortie                    | type, 82, 138          | CHAR_BIT             |
| format                    | bit, 144               | caractère, 99        |
| fprintf, 276              | champ de bits, 144     | taille objet         |
| fprintf                   | incomplet, 149         | opérateur            |
| format, 276               | non finie, 139         | sizeof(), 223        |
| sous-routine, 30          | supérieur              | sizeof()             |
| soustraction              | >                      | opérateur, 223       |
| -                         | opérateur, 217         | taille variable      |
| opérateur, 203            | opérateur              | type                 |
| opérateur                 | >, 217                 | vecteur, 132         |
| -, 203                    | supérieur ou égal      | vecteur              |
| standard                  | >=                     | type, 132            |
| bibliothèque, 260         | opérateur, 217         | taille variable      |
| type, 146                 | opérateur              | allocation           |
| static                    | >=, 217                | matrice, 199         |
| attribut                  | switch()               | matrice              |
| rangement, 84             | aiguillage             | allocation, 199      |
| rangement                 | instruction, 241       | test                 |
| attribut, 84              | instruction            | else                 |
| strup()                   | aiguillage, 241        | instruction, 49, 239 |
| chaîne de caractères      | symbolique             | if                   |
| clonage, 271              | expression             | instruction, 49, 239 |
| clonage                   | élémentaire, 187       | instruction          |
| chaîne de caractères, 271 | élémentaire            | else, 49, 239        |
| stream                    | expression, 187        | if, 49, 239          |
| entrée-sortie, 274        | système                | test d'égalité       |
| strlen()                  | binnaire               | ==                   |
| chaîne de caractères      | unité, 65              | opérateur, 217       |
| longueur, 269             | unité                  | opérateur            |
| longueur                  | binnaire, 65           | ==, 217              |
| chaîne de caractères, 269 | système d'exploitation | test de différence   |

|                           |                            |                                     |
|---------------------------|----------------------------|-------------------------------------|
| <b>!=</b>                 | chaîne                     | non signé                           |
| opérateur, 217            | caractère, 134             | entier, 101                         |
| <b>opérateur</b>          | coercion                   | passage en paramètre                |
| <b>!=, 217</b>            | conversion explicite, 228  | vecteur, 131                        |
| <b>texte</b>              | complexe                   | pointeur, 82, 126                   |
| Emacs                     | flottant, 83, 125          | allocation, 127                     |
| éditeur, 288              | conversion, 193            | arithmétique, 232                   |
| éditeur, 287              | conversion explicite       | incomplet, 149                      |
| Emacs, 288                | coercion, 228              | void, 149                           |
| timer, 39                 | conversion explicite       | <b>réel</b>                         |
| très long                 | cast, 228                  | arithmétique, 116                   |
| entier                    | entier                     | flottant, 83, 111                   |
| long long int, 101        | intégral, 83, 101          | scalaire, 82                        |
| long long int             | non signé, 101             | standard, 146                       |
| entier, 101               | expression, 173            | structure, 82, 138                  |
| <b>type</b>               | flottant                   | bit, 144                            |
| agrégé, 82                | arithmétique, 83, 111, 116 | champ de bits, 144                  |
| allocation                | complexe, 83, 125          | incomplet, 149                      |
| pointeur, 127             | réel, 83, 111              | non finie, 139                      |
| arithmétique, 82          | fonction, 82, 147          | <b>taille variable</b>              |
| flottant, 83, 111, 116    | incomplet, 150             | vecteur, 132                        |
| intégral, 83              | incomplet, 149             | <b>typedef</b>                      |
| pointeur, 232             | fonction, 150              | mommage, 171                        |
| réel, 116                 | pointeur, 149              | union, 82, 141                      |
| <b>bit</b>                | structure, 149             | vecteur, 82, 128                    |
| structure, 144            | incomplets, 82             | caractère, 134                      |
| <b>booléen</b>            | intégral                   | linéarisation, 129                  |
| intégral, 83, 108         | arithmétique, 83           | passage en paramètre, 131           |
| <b>caractère</b>          | booléen, 83, 108           | taille variable, 132                |
| <b>char</b> , 99          | caractère, 83, 90          | void                                |
| chaîne, 134               | entier, 83, 101            | pointeur, 149                       |
| intégral, 83, 90          | énumération, 83, 109       | énumération                         |
| large, 98, 107            | large                      | intégral, 83, 109                   |
| vecteur, 134              | caractère, 98, 107         | <b>typedef</b>                      |
| <b>cast</b>               | linéarisation              | mommage                             |
| conversion explicite, 228 | vecteur, 129               | type, 171                           |
| champ de bits             | mommage                    | type                                |
| structure, 144            | typedef, 171               | mommage, 171                        |
| <b>char</b>               | non finie                  | <b>typographie</b>                  |
| caractère, 99             | structure, 139             | ◀ □ ▶ conventions <sup>25</sup> ◀ ▷ |

|                       |                                 |                                        |
|-----------------------|---------------------------------|----------------------------------------|
| <code>uint16_t</code> | <code>valeur</code>             | <code>passage en paramètre</code>      |
| 16 bits               | <code>char</code>               | <code>type, 131</code>                 |
| entier, 105           | <code>limite, 99</code>         | <code>taille variable</code>           |
| entier                | <code>int</code>                | <code>type, 132</code>                 |
| 16 bits, 105          | <code>limite, 101</code>        | <code>type, 82, 128</code>             |
| <code>uint32_t</code> | <code>limite</code>             | <code>caractère, 134</code>            |
| 32 bits               | <code>char, 99</code>           | <code>linéarisation, 129</code>        |
| entier, 105           | <code>int, 101</code>           | <code>passage en paramètre, 131</code> |
| entier                | <code>long int, 101</code>      | <code>taille variable, 132</code>      |
| 32 bits, 105          | <code>long long int, 101</code> |                                        |
| <code>uint64_t</code> | <code>short int, 101</code>     |                                        |
| 64 bits               | <code>long int</code>           | <code>arithmétique</code>              |
| entier, 105           | <code>limite, 101</code>        | <code>fixe, 117</code>                 |
| entier                | <code>long long int</code>      | <code>fixe</code>                      |
| 64 bits, 105          | <code>limite, 101</code>        | <code>arithmétique, 117</code>         |
| <code>uint8_t</code>  | <code>short int</code>          | <code>visibilité</code>                |
| 8 bits                | <code>limite, 101</code>        | <code>variable, 46</code>              |
| entier, 105           | <code>Valgrind</code>           | <code>void</code>                      |
| entier                | déverminage                     | <code>pointeur</code>                  |
| 8 bits, 105           | mémoire, 342                    | <code>type, 149</code>                 |
| <code>UNICODE</code>  | mémoire                         | <code>pointeur, 149</code>             |
| caractère             | déverminage, 342                | <code>volatile</code>                  |
| codage, 98            | <code>variable</code>           | <code>attribut</code>                  |
| codage                | affectation, 45                 | <code>volatilité, 84, 161, 165</code>  |
| caractère, 98         | déclaration, 43, 84             | <code>volatilité</code>                |
| <code>union</code>    | initialisation, 43, 175         | <code>attribut, 84, 161, 165</code>    |
|                       | <code>globale, 46</code>        | <code>volatilité</code>                |
| <code>unité</code>    | <code>incomplète</code>         | <code>attribut</code>                  |
|                       | initialisation, 176             | <code>volatile, 84, 161, 165</code>    |
|                       | <code>initialisation</code>     | <code>volatilité</code>                |
|                       | déclaration, 43, 175            | <code>attribut, 84, 161, 165</code>    |
|                       | incomplète, 176                 | <code>vous êtes ici, 366</code>        |
| <code>UNIX</code>     | <code>locale, 46</code>         |                                        |
|                       | <code>scope, 46</code>          |                                        |
|                       | <code>visibilité, 46</code>     |                                        |
| <code>unsigned</code> | <code>vecteur</code>            | <code>wchar_t</code>                   |
| attribut              | <code>caractère</code>          | <code>caractère</code>                 |
| entier, 101           | <code>type, 134</code>          | <code>large, 107</code>                |
| entier                | <code>linéarisation</code>      | <code>large</code>                     |
| attribut, 101         | <code>type, 129</code>          | <code>caractère, 107</code>            |
|                       |                                 | <code>while</code>                     |

|                      |                     |                   |
|----------------------|---------------------|-------------------|
| instruction, 50, 242 | éditeur de lien, 36 | symbolique        |
| instruction          | édition de lien, 36 | expression, 187   |
| boucle, 50, 242      | élémentaire         | énumération       |
| wint_t               | agrégat             | entier            |
| entier               | expression, 188     | enum, 109         |
| large                | expression          | enum              |
| large                | agrégat, 188        | entier, 109       |
| entier, 107          | L-valeur, 189       | integral          |
| éditeur              | littérale, 183      | type, 83, 109     |
| Emacs                | lvalue, 189         | type              |
| texte, 288           | symbolique, 187     | integral, 83, 109 |
| texte, 287           | L-valeur            | étendu            |
| Emacs, 288           | expression, 189     | flottant          |
| éditeur de texte     | littérale           | long double, 111  |
| emacs, 12            | expression, 183     | long double       |
|                      | lvalue              | flottant, 111     |
|                      | expression, 189     |                   |

# Le plan

- 1 Introduction
  - Bibliographie
  - Petite histoire
- 2 Langage
  - Généralités
- 3 Un ordinateur ? Mais qu'est-ce ?
  - Les bases de la numérologie
  - Calcul binaire
- 4 Déclarations
  - Généralités
  - Types des objets
  - Portée
- 5 Expressions
  - Sémantique opérateurs
  - Coercition de type
- 6 Instructions
- 7 Préprocesseur
- 8 Bibliothèques
- 9 Modèle mémoire & allocation
- 10 Entrées-sorties
- 11 GNOME
- 12 Éditeur exemple d'Emacs
  - Tuning de son clavier
- 13 Compilation
- 14 Modularisation
- 15 Mise au point
  - Dévermineur (debugger)
  - Analyse mémoire
- 16 Extensions
  - Du C pour le graphisme
  - C++
- 17 Délires
  - Le bêtisier
  - Concours de programmes incompréhensibles
- 18 Conclusion
- 19 Index
- 20 Table des matières

|                                                  |                                                  |                                              |          |
|--------------------------------------------------|--------------------------------------------------|----------------------------------------------|----------|
| <b>1</b>                                         | Introduction                                     | Boucle pour<br>Bien présenter les programmes | 51<br>52 |
| Le plan                                          | Règle de base                                    | 54                                           |          |
| Domaine d'utilisation                            | Indentation                                      | 55                                           |          |
| Pourquoi un cours de C ?                         | <b>3</b>                                         | Un ordinateur ? Mais qu'est-ce ?             |          |
| De nombreuses extensions                         | Le plan                                          | 59                                           |          |
| Problématique pédagogique                        | <b>Les bases de la numérologie</b>               |                                              |          |
| Hétérogénéité des élèves                         | Le plan                                          | 60                                           |          |
| <b>Bibliographie</b>                             | Un peu de numérologie                            | 61                                           |          |
| Le plan                                          | Représentation des entiers                       | 62                                           |          |
| Ressources & Bibliographie                       | Numérotation binaire                             | 64                                           |          |
| Monitorat                                        | Unités binaires                                  | 65                                           |          |
| Monitorat                                        | Autres bases dérivées du binaire                 | 67                                           |          |
| Monitorat                                        | Base 8 (octal)                                   | 68                                           |          |
| <b>Petite histoire</b>                           | Base 16 (hexadécimal)                            | 70                                           |          |
| Le plan                                          | Base 256                                         | 71                                           |          |
| Préhistoire                                      | Authentification avec mot de passe jetable S/Key | 72                                           |          |
| C & Unix Story                                   | Base 65536                                       | 73                                           |          |
| Notations typographiques utilisées dans ce cours | <b>Calcul binaire</b>                            |                                              |          |
| <b>2</b>                                         | Le plan                                          | 74                                           |          |
| Langage                                          | Représenter des nombres négatifs                 | 75                                           |          |
| Le plan                                          | Nombres binaires en complément à 2               | 77                                           |          |
| <b>Généralités</b>                               | <b>4</b>                                         | Déclarations                                 |          |
| Le plan                                          | Le plan                                          | 79                                           |          |
| Des instructions et expressions C...             | <b>Généralités</b>                               |                                              |          |
| ... en passant par les fonctions C...            | Le plan                                          | 80                                           |          |
| ... aux programmes C                             | Objets en C                                      | 81                                           |          |
| Programme minimal en C                           | Types en C                                       | 82                                           |          |
| Programme hello world en C                       | Types arithmétiques en C                         | 83                                           |          |
| Commentaires                                     | Déclaration de variable                          | 84                                           |          |
| Vers un programme exécutable                     | Identificateurs                                  | 85                                           |          |
| Compilation d'un programme                       | Mots-clés réservés en C                          | 87                                           |          |
| Déclarer des fonctions                           | <b>Types des objets</b>                          |                                              |          |
| Types et expressions                             | Le plan                                          | 89                                           |          |
| Variables et déclaration                         | Caractères                                       | 90                                           |          |
| Affectation de variables                         | Table des caractères ASCII                       | 91                                           |          |
| Visibilité des variables                         | Codage ISO-8859                                  | 97                                           |          |
| La vie est un long fleuve tranquille             | Codage UNICODE                                   | 98                                           |          |
| Tests                                            |                                                  |                                              |          |
| Boucle tant que                                  |                                                  |                                              |          |

|                                               |     |                                                 |     |
|-----------------------------------------------|-----|-------------------------------------------------|-----|
| Caractères de base                            | 99  | Variables temporaires                           | 180 |
| Nombres entiers                               | 101 |                                                 |     |
| Types entiers étendus de taille connue        | 105 | <b>5 Expressions</b>                            |     |
| Caractères larges                             | 107 | Le plan                                         | 181 |
| Booléens                                      | 108 | Définition des expressions                      | 182 |
| Énumérations                                  | 109 | Expressions élémentaires littérales             | 183 |
| Nombres flottants                             | 111 | Expressions élémentaires symbolique             | 187 |
| Conversion flottants → entiers à l'arrache    | 114 | Constantes agrégats                             | 188 |
| Nombres flottants ≠ réels !                   | 116 | Constantes L-valeurs (lvalues)                  | 189 |
| Algorithme de sommation de flottants          | 118 | Constructions d'expression                      | 190 |
| Vers des nombres flottants normalisés         | 120 | Priorité des opérateurs                         | 191 |
| Pourquoi des nombres flottants dénormalisés ? | 121 | Conversions de type                             | 193 |
| Dénormalisation flottante                     | 123 |                                                 |     |
| Nombres complexes                             | 125 | <b>Sémantique opérateurs</b>                    |     |
| Pointeurs                                     | 126 | Le plan                                         | 195 |
| Vecteurs                                      | 128 | Opérateur .                                     | 196 |
| Vecteur en paramètre de fonction              | 131 | Opérateur ->                                    | 197 |
| Chaîne de caractères                          | 134 | Indication vecteur et matrices []               | 198 |
| Chaînes de caractères larges                  | 137 | Appels de fonctions & procédures                | 202 |
| Structures de données                         | 138 | Arithmétique                                    | 203 |
| Type d'union de types                         | 141 | Arithmétique binaire                            | 204 |
| Champs de bits                                | 144 | Optimisations en binaire                        | 207 |
| Autres types standards                        | 146 | Applications codage binaire : multispin-coding  | 210 |
| Types de fonctions                            | 147 | Application utilisant des additions 9 et 6 bits | 211 |
| Types incomplets                              | 149 | Gaz sur réseau                                  | 212 |
| FAQ Composition opérateurs de déclaration     | 151 | Expressions logiques                            | 217 |
| <b>Portée</b>                                 |     | Expression conditionnelle                       | 220 |
| Le plan                                       | 154 | Opérateurs à effet de bord                      | 221 |
| Portée des déclarations                       | 155 | Taille d'un objet                               | 223 |
| Où déclarer les variables                     | 159 |                                                 |     |
| Vie réelle des objets                         | 160 | <b>Coercition de type</b>                       |     |
| Objets non modifiables                        | 164 | Le plan                                         | 224 |
| Des pointeurs à problèmes...                  | 166 | Coercition : conversion explicite de type       | 225 |
| Exemple de C : pointeurs                      | 167 | cast en anglais                                 | 227 |
| Exemple de C : aliasing                       | 168 | Transtypage explicite ou cast                   | 228 |
| Pointeurs restreint                           | 170 | Pointeurs & manipulation d'adresses             | 230 |
| Nommage de type                               | 171 | Arithmétique sur pointeurs                      | 232 |
| Expression de type                            | 173 | Liste d'expressions                             | 234 |
| Initialisations de variables                  | 175 |                                                 |     |
| Initialisations avec types compliqués         | 176 | <b>6 Instructions</b>                           |     |
|                                               |     | Le plan                                         | 235 |
|                                               |     | Instructions                                    | 236 |

|                                          |     |                                                   |     |
|------------------------------------------|-----|---------------------------------------------------|-----|
| Instructions élémentaires                | 237 | Éditeur de texte                                  | 287 |
| Tests if                                 | 239 | Emacs : LE éditeur                                | 288 |
| Aiguillage switch                        | 241 | Un éditeur abordable                              | 291 |
| Boucle while                             | 242 | Concepts de base                                  | 292 |
| Boucle do                                | 243 | Kit de survie                                     | 293 |
| Boucle for                               | 244 | Emacs est lourd                                   | 295 |
| Instructions de saut                     | 245 | Vers une analyse grammaticale des commandes Emacs | 297 |
| <b>7</b>                                 |     | Compilation                                       | 298 |
| Préprocesseur                            |     | Correction erreurs de syntaxe à la volée          | 299 |
| Le plan                                  | 247 | Débogueurs dans Emacs                             | 300 |
| Préprocesseur                            | 248 | Tags                                              | 301 |
| Préprocesseur C (CPP)                    | 249 | Speedbar                                          | 302 |
| Inclusion de fichiers                    | 250 | Édition de fichiers binaires                      | 303 |
| Macroconstantes et macrofonctions        | 251 | Nécessité d'un système de gestion de version      | 304 |
| Compilation conditionnelle               | 256 | Gestion de version avec Emacs                     | 306 |
| <b>8</b>                                 |     | Gestion de version en local                       | 308 |
| Bibliothèques                            |     | Gestion de version à distance                     | 309 |
| Le plan                                  | 258 | Gestion de version avec Emacs pour CVS et SVN     | 311 |
| Bibliothèques                            | 259 | TRAMP                                             | 312 |
| Bibliothèque standard                    | 260 | Modes spécifiques à des langages                  | 313 |
| Contenu bibliothèque standard glibc      | 262 | ● Tuning de son clavier                           |     |
| ● Modèle mémoire & allocation            |     | Le plan                                           | 314 |
| Le plan                                  | 265 | Retour aux sources : clavier QWERTY               | 315 |
| Modèle mémoire du C                      | 266 | Revamper un clavier AZERTY en QWERTY              | 316 |
| Allocation dynamique                     | 267 | Des accents !                                     | 317 |
| Chaînes de caractères                    | 269 | Touches Meta et Alt                               | 318 |
| _strdup : LA fonction la plus utile du C | 271 | Avec l'expérience                                 | 320 |
| ● Entrées-sorties                        |     | <b>10</b>                                         |     |
| Le plan                                  | 273 | Compilation                                       |     |
| Entrées-sorties sur stream               | 274 | Le plan                                           | 321 |
| Affichage formaté avec printf            | 276 | Principe                                          | 322 |
| Lecture formatée avec scanf              | 279 | Compilateur gcc                                   | 323 |
| goto non locaux                          | 280 | Makefile                                          | 325 |
| ● GNOME                                  |     | <b>11</b>                                         |     |
| Le plan                                  | 283 | Modularisation                                    |     |
| GNOME                                    | 284 | Le plan                                           | 328 |
| Glib                                     | 285 | Modularisation                                    | 329 |
| <b>9</b>                                 |     | Fonctions                                         | 330 |
| Éditeur exemple d'Emacs                  |     | Fichiers                                          | 331 |
| Le plan                                  | 286 | Utiliser des symboles                             | 332 |

12

## Mise au point

Le plan

333

## Le bêtisier

353

Dévermineur (debugger)

Le plan

334

L'octal qui passe à la trappe

354

Débogueur

335

Choses à (ne pas) faire

355

Débogueur GNU gdb

337

Débogueur ddd

340

356

Analyse mémoire

Le plan

341

357

Mise au point allocation mémoire

342

358

359

360

361

362

13

## Extensions

Le plan

346

Du C pour le graphisme

Le plan

347

363

CUDA

348

364

Nouveaux formats de données

C++

Le plan

349

365

C++

Délires

Le plan

350

366

351

367

14

15

## Conclusion

368

Le plan

352

369

Conclusion

353

370

16

## Index

371

Le plan

354

372

17

## Table des matières

373

Le plan

355

374

## Vous êtes ici !

393