

Le plan

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

- 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



C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON & R. KERYELL

2 / 361

Domaine d'utilisation

(I)

Pourquoi un cours de C ?

(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



C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON & R. KERYELL

3 / 361

C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON & R. KERYELL

4 / 361

Pourquoi un cours de C ?

-  Ce cours presuppose connaissance d'un langage informatique !

(II)

De nombreuses extensions

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

(I)



Problématique pédagogique

(I)

Hétérogénéité des élèves

(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



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
- Conclusion
- Index
- Table des matières



Ressources & Bibliographie

- <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

(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 ☺



► Bibliographie
(II)

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

| \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

$i \exists$ 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 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
- 17 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

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



C & Unix Story

- 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

- 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

(II)

C & Unix Story

- 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

(IV)

C & Unix Story

- 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

(III)

C & Unix Story

- 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>

(V)

C & Unix Story

- 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 & recrifiable gcc et pour d'autres langages

(VI)



C & Unix Story

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

(VII)



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 joli~~s~~ espaces ~~uuu~~
 - ▶ 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 `gotobed`
- Commentaires `/* Comme_en_terre */`
- Chaînes de caractères "chêne_de_caractère"



Le plan

- | | |
|--|--|
| <ul style="list-style-type: none"> 1 Introduction <ul style="list-style-type: none"> ● Bibliographie ● Petite histoire 2 Langage <ul style="list-style-type: none"> ● Généralités 3 Un ordinateur ? Mais qu'est-ce ? <ul style="list-style-type: none"> ● Les bases de la numérologie ● Calcul binaire 4 Déclarations <ul style="list-style-type: none"> ● Généralités ● Types des objets ● Portée 5 Expressions <ul style="list-style-type: none"> ● Sémantique opérateurs ● Coercition de type 6 Instructions 7 Préprocesseur 8 Bibliothèques | <ul style="list-style-type: none"> ● Modèle mémoire & allocation ● Entrées-sorties ● GNOME ● Éditeur exemple d'Emacs ● Tuning de son clavier ● Compilation ● Modularisation ● Mise au point ● Dévermineur (debugger) ● Analyse mémoire ● Extensions <ul style="list-style-type: none"> ● Du C pour le graphisme ● C++ ● Délires <ul style="list-style-type: none"> ● Le bêtisier ● Concours de programmes incompréhensibles ● Conclusion ● Index ● 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 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
- Conclusion
- Index
- Table des matières



Des instructions et expressions C...

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

► Généralités
(II)



Des instructions et expressions C...

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

```
1 au=u4 ;  
2 c  
uuu=  
4 uuuuuuuuuuuuuud [i] ;
```

Definition

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

- Expressions récursivement composées d'expressions

```
1 bu=u3u+u4*cos (x) ;
```



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

- 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 : `strupdup(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...

- ▶ 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)

(II)

... 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)

Programme hello world en C

(I)

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

```
1 /* Pour pouvoir utiliser proprement
   la fonction de sortie : */
3 #include <stdio.h>
/* Définit entre autres les codes de retour :
5 #include <stdlib.h>

7 int main(int argc, char* argv[]){
uu/* Affiche sur la « sortie-standard » */
9 uu puts("Hello, world!");
uu// Renvoie une marque de réussite :
11 uu return EXIT_SUCCESS;
}
```

- 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

- 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

(I)



Vers un programme exécutable

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

- Préprocesseur : gère dans code source les #quelque-chose, 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)

(II)



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.
3 $ ls -l
4 -rwxr-xr-x 1 keryell keryell 6918 2005-10-12 22:03 hello_world
5 -rwxr--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

- 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 }
```

C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON & R. KERYELL



37 / 361

(I)

Déclarer des fonctions

```

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;
}
```

C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON & R. KERYELL



38 / 361

Déclarer des fonctions

```

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

C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON & R. KERYELL



39 / 361

(III)

Types et expressions

- 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...



40 / 361

(II)

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   exchange=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
3 while (condition);

```



Boucle pour

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

```

1 for(expression_initiale;
2   uuuuuuexpression_de_test;
3   uuuuuuexpression_avant_rebouclage)
4   uuinstruction_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\%","");
5     write(fd, (char*)&pointers_to_fingers[0], buf.st_size)
6   }
}

```

(I)

Bien présenter les programmes

- 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

(I)

Bien présenter les programmes

- 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!!! ☺

(II)

Règle de base

- Un caractère dans un fichier coûte $\approx 1,5 \cdot 10^{-10}$ € 2008 ☺
- Ne pas hésiter à rajouter des sauts de lignes, des espaces épaisse... ☺
- 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 ! ☺

(I)

Indentation

- Idée : utiliser des espaces en tête de ligne pour faire ressortir localité/modularité/hierarchie des structure
- Intérêt du concept : portabilité de cette présentation en dehors de son propre éditeur ou IDE
- Caractère « tabulation » interprété différemment selon les systèmes et configuration
 - ▶ 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 ? ☺

(I)

(II)



Indentation

```

1 static_void skew_the_file ( int_number_of_skewing , char_*name )
{
3   int fd , i , j , skew , size_end , size_begin , size ;
5   struct stat buf;
7   pointers_to_fingers = malloc ( buf . st_size );
9   size = buf . st_size / sizeof ( int );
11  if ( random_pointers ){
12    for ( i = 0 ; i < number_of_skewing ; i ++ ){
13      for ( j = 0 ; j < size ; j ++ )
14        pointers_to_fingers [ j ] = ( int )( drand48 () * size ) * sizeof ( int );
15        testerreut (" Unable_to_write_pointer_file_random \"%s\" " ,
16                      write ( fd , ( char_* ) & pointers_to_fingers [ 0 ] , buf . st_size )
17                      , buf . st_size , name );
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        testerreut (" 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  }
28  else {
29    testerreut (" Unable_to_read_pointer_file \"%s\" " ,
30                read ( fd , ( char_* ) & pointers_to_fingers , buf . st_size ) , buf . st_size ,
31                name );
32    /* The_file_size_is_multiplied_by_number_of_skewing : */
33    for ( i = 1 ; i < number_of_skewing ; i ++ ){
34      skew = i % ( buf . st_size / sizeof ( int ) );
35      size_end = skew * sizeof ( int );
36      size_begin = buf . st_size - size_end ;
37      testerreut (" Unable_to_write_pointer_file_begin \"%s\" " ,
38                  write ( fd , ( char_* ) & pointers_to_fingers [ skew ] , size_begin )
39                  , size_begin , name );
40      testerreut (" Unable_to_write_pointer_file_end \"%s\" " ,
41                  write ( fd , ( char_* ) & pointers_to_fingers [ 0 ] , size_end )
42                  , size_end , name );
43    }
44    testerreut (" Unable_to_close_pointer_file \"%s\" " , close ( fd ) , name );
45  }
46  testerreut (" Unable_to_close_pointer_file \"%s\" " , close ( fd ) , name );
47 }
```



Indentation

(III)

(IV)



Indentation

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
- 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



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.

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



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



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

☰ des historiens des nombres !

■ C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON & R. KERYELL



61 / 361

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 $(\frac{3}{2})^{12} \approx 2^7 = 128 \rightsquigarrow$ base de la musique moderne européenne 12 quintes ≈ 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

■ C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON & R. KERYELL



63 / 361

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

■ C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON & R. KERYELL



62 / 361

Unités binaires

(II)

<i>Décimal</i>		<i>Binaire</i>		
<i>Facteur</i>	<i>Préfixe</i>	<i>Facteur</i>	<i>Préfixe</i>	<i>Détail</i>
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... ☺

■ C avant C – UV2 INF 446

■ C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON & R. KERYELL



64 / 361

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

■ C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON & R. KERYELL



65 / 361

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)

■ C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON & R. KERYELL



67 / 361

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 : 755_8

- umask

■ C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON & R. KERYELL



66 / 361

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



68 / 361

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



■ C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON & R. KERYELL

69 / 361

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 cours choisis dans vocabulaire anglais
 - Plus long à taper
 - Mais moyen mnémotechnique et CRC (mot anglais) ☺
- TANK WELT MOT HAL FATE MUSH



■ C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON & R. KERYELL

70 / 361

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:fefab:64ee
 - 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



■ C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON & R. KERYELL

71 / 361

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
- Index
- 17 Table des matières



■ C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON & R. KERYELL

72 / 361

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 $\mathbb{Z}/b^n\mathbb{Z}$ 

73 / 361

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 \sim 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, $m > n$
 - $+3_{10}, [0]_2 \sim [0000]_2$



75 / 361

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

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

- Pratique

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

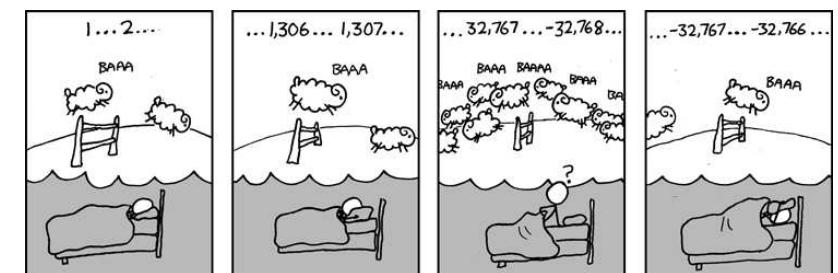


74 / 361

Nombres binaires en complément à 2

(II)

- $-3_{10}, [1]_2 \sim [1111]_2$
- \sim 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>



76 / 361

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



Objets en C

- 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)

(I)



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



Types en C

- 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

- 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 ℝ `float | [long]_double`
 - ▶ Flottants complexes (C99) : approximation de ℂ
(`float | [long]_double`)`_complex`

(I)



Déclaration de variable

`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 ≠ alias de pointeurs (C99)
 - `volatile` : ∃ 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 _



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



Mots-clés réservés en C

(II)

- Identifiant commençant par sig suivie par _ ou majuscule

Comparaison avec différents dialectes du C :

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



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	00	00	CTRL-\@	\0 NUL (Null prompt)
001	0000001	01	01	CTRL-A	SOH (Start Of Heading, console interrupt)
002	0000010	02	02	CTRL-B	STX (Start of TeXt, maintenance mode on HP)
003	0000011	03	03	CTRL-C	ETX (End of TeXt)
004	0000100	04	04	CTRL-D	EOT (End Of Transmission, not same as ETB)
005	0000101	05	05	CTRL-E	ENQ (ENquiry, goes with ACK; old HP flow control)
006	0000110	06	06	CTRL-F	ACK (ACKnowledge, clears ENQ logon hand)
007	0000111	07	07	CTRL-G	\a BEL (BELL)
008	0001000	10	08	CTRL-H	\b BS (BackSpace, works on HP)
009	0001001	11	09	CTRL-I	\t HT (Horizontal Tab)
010	0001010	12	0A	CTRL-J	\n LF (Line Feed) or NL (New Line) or EOL (End Of Line). Unix EOL.
011	0001011	13	0B	CTRL-K	\v VT (Vertical Tab)
012	0001100	14	0C	CTRL-L	\f NP (New Page, page eject) or FF (Form Feed)
013	0001101	15	0D	CTRL-M	\r CR (Carriage Return). Mac EOL (Win EOL is \r\n).



■ C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON & R. KERYELL

89 / 361

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



■ C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON & R. KERYELL

91 / 361

Table des caractères ASCII

(II)

014	0001110	016	0E	CTRL-N	SO (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 Acknowledged)
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 URLs)
036	0100100	044	24	\$	dollar sign
037	0100101	045	25	%	percent sign. %25(escape in URLs)



■ C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON & R. KERYELL

90 / 361

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	1111000	174	7C		vertical bar
125	1111001	175	7D	}	right curly brace
126	1111110	176	7E	~	tilde
127	1111111	177	7F	DEL	(DELETE or rubout), cross-hatch box



■ C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON & R. KERYELL

92 / 361

Table des caractères ASCII

- Propriétés intéressantes
 - ▶ Codable sur 7 bits
 - ▶ Il n'y a pas que des glyphs 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

(V)

Table des caractères ASCII

- 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

(VI)



93 / 361



94 / 361

Codage ISO-8859

- 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'HAŸ-LES-ROSES »)
- ... corrigé en ISO-8859-15 (latin-9) qui inclut au passage l'€
- Tout loge dans un entier de 8 bits ☺

(I)

Codage UNICODE

(I)



95 / 361



96 / 361

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
- ↗ 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
 ↗ × CHAR_BIT pour taille en bittaille
- Limites réelles définies 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
#endif

```

Nombres entiers

(I)

- 4 tailles d'entiers signés (par défaut)
 - **short int** ou **short** (pour les flémards)
 - **int**
 - **long int** ou **long** (pour les flémards)
 - **long long int** ou **long 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
#endif

```

Nombres entiers

```

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 ULONG_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 ☺

(III)

Nombres entiers

- 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

- 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_leastN_t` et `uint_leastN_t`
- Concept intéressant : type le plus rapide et suffisamment gros (parfois, si plus gros, plus rapide) `int_fastN_t` et `uint_fastN_t`
- Pour tout les cas, limites disponibles `TYPE_MIN`, `TYPE_MAX`, `UTYPE_MAX`. Exemple :

(I)

Types entiers étendus de taille connue

```

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

- 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

■ C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON & R. KERYELL



105 / 361

(I)

Booléens

- Traditionnellement en C, l'entier 0 valait « faux » et tout autre valeur signifiait « vrai »
- Rajout type `bool` dans C99 pour expliciter
- ↗ 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_ Bool
2 #define _true_ 1
3 #define _false_ 0

```

Énumérations

- 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_e_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

```

■ C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON & R. KERYELL



107 / 361

(I)

■ C avant C – UV2 INF 446
TÉLÉCOM Bretagne/Info/HPCAS
S. EVEN, S. GUELTON & R. KERYELL
106 / 361



Énumérations

- 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]*[ \t]*[^\t]*$)/enum { \1 = \2 };/'
```

■ C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON & R. KERYELL

108 / 361

■ C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON & R. KERYELL



Nombres flottants

- 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**)

■ C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON & R. KERYELL



109 / 361

(I)

Nombres flottants

- 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 : $E_{\text{réel}} = E_{\text{stocké}} - E_{\text{biais}}$

► 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	$\leq 3,6 \cdot 10^{-4951}$	$\leq 3,4 \cdot 10^{-4932}$	$\geq 1,2 \cdot 10^{4932}$	$\leq 5,42 \cdot 10^{-20}$	$\geq 18-21$

Nombreux paramètres définis dans `<float.h>`



110 / 361

Nombres flottants

- 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...)
- ∃ Nombreux choix d'arrondi (plus proche, +, -, vers 0,...)
- <http://groupes.ieee.org/groups/754> En cours de révision
http://en.wikipedia.org/wiki/IEEE_754r si vous voulez participer ☺



■ C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON & R. KERYELL



111 / 361

(III)

Conversion flottants → entiers à l'arrache

- En temps normal `int_uftoi(float_tf)` doit faire le travail
- ∃ 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_tf)
2 {
3     // Récupère les bits du flottant dans un entier :
4     uint32_t dw_u = *(uint32_t*) &f;
5     // La valeur spéciale où tous les bits sont à 0 code 0 :
6     if (dw_u == 0)
7         return 0;
8     // Récupère l'exposant codé sur 8 bits et compensation le biais :
9     char exp_u = (dw_u >> 23) - 127; // Suppose un char de 8 bits
10    if (exp_u < 0 || exp_u > 23)
11        /* Si l'exposant est négatif, de toute manière, le nombre vaut
12           */

```



112 / 361

■ C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON & R. KERYELL

(II)

Nombres flottants

- 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 : $E_{\text{réel}} = E_{\text{stocké}} - E_{\text{biais}}$

► 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	$\leq 3,6 \cdot 10^{-4951}$	$\leq 3,4 \cdot 10^{-4932}$	$\geq 1,2 \cdot 10^{4932}$	$\leq 5,42 \cdot 10^{-20}$	$\geq 18-21$

Nombreux paramètres définis dans `<float.h>`



110 / 361

Conversion flottants→entiers à l'arrache

(II)

```

12     .....moins que 1, donc arrondi à 0. Si c'est supérieur à 23, le nombre
13     .....est supérieur à 2^{24} en on décide de le jeter et de répondre
14     .....arbitrairement 0. Mmm... On pourrait gérer jusqu'à 2^{32} mais
15     .....faudrait corriger le code ci-après */
16     return 0;
17     /* Construit le nombre avec le 1 de poids fort qui est économisé dans
18     .....la norme IEEE-754, puis les 23 autres bits de la mantisse cadrés
19     .....en fonction de l'exposant :*/
20     int val= (1<<exp)+((dw&0x7FFFFF)>>(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 ≠ 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
 - ▶ TEX 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

Nombres flottants ≠ réels !

(I)

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

- Nombres flottants ≡ 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



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
5    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 ! ☺



Pourquoi des nombres flottants dénormalisés ? (I)

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

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

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

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

$$a \oplus b = 0$$

$$a \neq b$$

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



Vers des nombres flottants normalisés

- 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 M le plus grand pouvant loger dans les bits alloués pour la mantisse
- Idée 2
 - $\forall M \neq 0$: 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 ? (II)

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



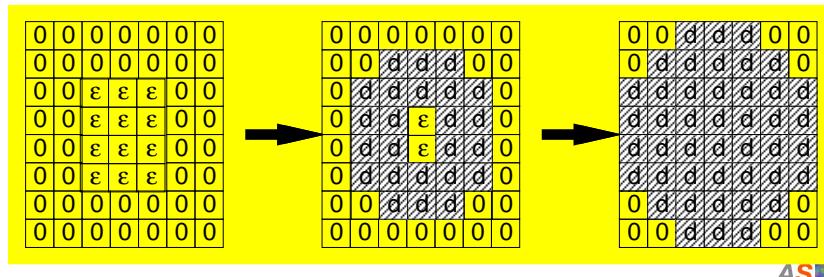
Dénormalisation flottante

(I)

- 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



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

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

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

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

Nombres complexes

(I)

- Rajout en C99 des flottants en version complexe \mathbb{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
- `type * identifiant;` définit `identifiant` comme variable de type pointeur vers `type`
- Profondeur arbitraire : on peut aussi manipuler adresse de cette variable
- `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

- ⚠ 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 ☺

(II)

Vecteurs

(I)

- Suite ordonnée d'objets de même type
 - Déclaration du style
- `typeU identifiant [dim1] [dim2] ... [dimn];`
- 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)

Vecteurs

(III)

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

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

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

- `floatU 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



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   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

(III)

```

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

```



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   ...
3

```

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

```

1 void affiche(double matrice[X][Y]){
...
3

```

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



Chaîne de caractères

(I)

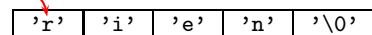
- Interaction avec monde extérieur ↗ besoin de chaînes de caractères
- **Sens de l'économie du C** : ne pas rajouter type spécial dans langage (\neq string en Basic ou Java)
- ↗ Besoin d'une « convention collective »
 - ▶ Chaîne en C = vecteur de caractères
 - ▶ Pas de stockage de la taille contrairement 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

```
1 char_u 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) ☺



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



Chaîne de caractères

(II)

- **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 ! ☺)



Structures de données

(I)

- Regroupement local séquentiel d'objets C hétérogènes

- Déclaration d'objets

```
1 struct {
  double x;
  double y;
  int couleur;
} p;
```

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



Structures de données

- 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

(II)



Type d'union de types

(I)

- Besoin de pouvoir stocker différents types d'objets dans une variable
 - Contrairement aux structures, champs exclusifs
- ```
1 union [nom-union] { (type champ) + } identifiant;
```
- ↗ Éviter suppositions de stockage (stockage des entiers par gros bout ou petit bout), de compactage, d'optimisation...



(II)

## Structures de données

```
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)



(II)

## Type d'union de types

```
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.type = Entier;
9 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

- 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...

(III)

## Champs de bits

- 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 uuuu//...

3 ----->unsigned_int_flowctrl :1;

4 ----->unsigned_int_default_port:4; /* Last_dev->if_port_value */

5 ----->unsigned_int_an_enable :1;

6 uuuu/* ... */

}

```

- 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)

## Autres types standards

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

- ∃ 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);
}
```

- 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 ☺
- Matériau de base
  - ▶ \* déclare pointeurs
  - ▶ [] déclare vecteurs/matrices
  - ▶ () déclare une fonction
- Comment déclarer tableau de pointeur ?

1 `char*u`

typiquement tableau de chaînes de caractères

- Comment déclarer fonction qui renvoie pointeur ?

1 `void*(size_tsize);`

■ C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON & R. KERYELL



149 / 361

# FAQ Composition opérateurs de déclaration (III)

- 1 `*u`p  
2 `(*u`p)()  
3 `int*(*u`p)()  
4 `int*(*u`p)(`double[] ,u`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 ☺

■ C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON & R. KERYELL



151 / 361

# 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*(*u`p)()

rapproche « pointeur » de p

▶ Comment déclarer pointeur vers matrice de double ?

1 `double*(*u`p\_matrix)[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 »



■ C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON & R. KERYELL

150 / 361

# 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 avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON & R. KERYELL

152 / 361

## Portée des déclarations

- 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 for (int i=0; ...)
4 ...
}
```

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

- ▶ Récursion

(I)

## Portée des déclarations

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 double fibonacci(double n){
5 if(n<=1){return 1;}
6 return fibonacci(n-1)+fibonacci(n-2);
7
8 int main(int argc, const char* argv[static 2]){
9 double v = atof(argv[1]);
10 (void)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)

## Portée des déclarations

(IV)

- ▶ 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
}
```

- ▶ 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 (≈ identique...) dans plusieurs fichiers
- Mais où se trouve la vraie version ? Où est le stockage de la variable ?



## Où déclarer les variables

(I)

- Différentes religions
- Déclaration en tête de fonctions
  - 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 masquages pour santé mentale des relecteurs...

■ C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON &amp; R. KERYELL



157 / 361

## 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 !

■ C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON &amp; R. KERYELL

158 / 361



## 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...

■ C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON &amp; R. KERYELL



159 / 361

## 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_ajolie_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



■ C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON &amp; R. KERYELL

160 / 361

## 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 __thread int ui;
```
- Cf discussion
<http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1364.htm>
http://en.wikipedia.org/wiki/Thread-local_storage
- Souvent sujet chaud des langages (comparer en PERL `my` et `local` !)

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;
```

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

## 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...



(I)

## Exemple de C : aliasing

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 }
}

```

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



(I)

## Pointeurs restreint

- 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,
 const void *restrict src,
 size_t n);
5 void *memmove(void *dest, const void *src, size_t n);

```

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



## Nommage de type

- 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 enum {N=1000,M=10000};
4
5 typedef double matrice[N][M];
6 matrice a,b;
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

```

(I)

## Nommage de type

```

20 void *valeur;
21 arbre1; // arbre1 est un élément
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 */

```



■ C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON &amp; R. KERYELL

169 / 361

■ C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON &amp; R. KERYELL

170 / 361

## Expression de type

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

```

(I)

## Expression de type

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



■ C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON &amp; R. KERYELL

171 / 361

■ C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON &amp; R. KERYELL

172 / 361

(II)

## Initialisations de variables

- 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 **pu=&chaine;
```

## Initialisations avec types compliqués

### Initialisation de vecteurs

```
1 /* chaîne 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"};
// On aurait pu faire pire avec studieux[8] !
6 /* chaîne immuable est un pointeur vers une chaîne constante
 dans les instructions du programme... */
8 char *chaine_immutable = "dans le code !";

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

### Initialisations incomplètes en C99

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



## Initialisations avec types compliqués

## Initialisations avec types compliqués

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

```
1 double gros_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



```
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 element {
11 type = Entier,
12 valeur.i = 3
13 };
15 typedef struct {
16 int x, y;
17 } vecteur [10];
19 vecteur v = {
20 {1, 3},
```

## Initialisations avec types compliqués

```

21 uu [5] u=u{u.y=u3} ,
22 uu [7] u=u{x=u1,u.y=u2u} ,
23 uu{u.x=u3,u.y=u4u}
};
```

## (IV)

## 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
- 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

## 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



181 / 361

## 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



183 / 361

## 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

(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 uuuuadresse_=&ici;
```



## Constantes agrégats

(I)

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

```
1 calcule((element){
2 uuuuuuuuuuuuuuuuuuuuuuuuuuu.u.type=_Entier,
3 uuuuuuuuuuuuuuuuuuuuuuuuuuu.u.valeur.i=_3
4 uuuuuuuuuuuuuuuuuuuuuuuuu});
```

## 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*`)0xE0000000 :

```
1 uu((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 : infixé

```
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

■ C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON &amp; R. KERYELL



189 / 361

## Priorité des opérateurs

(II)

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

↗ Rajouter des parenthèses si cela diffère du défaut ou pour clarifier ou assurer ☺

En ligne : `man operator`



190 / 361

## 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;
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
```

■ C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON &amp; R. KERYELL



191 / 361

## 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

► ...



192 / 361

■ C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON &amp; R. KERYELL

# 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
- Conclusion
- Index
- Table des matières



■ C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON &amp; R. KERYELL

193 / 361

# Opérateur ->

- 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...



■ C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON &amp; R. KERYELL

195 / 361

# Opérateur .

- 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);

```

■ C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON &amp; R. KERYELL

194 / 361



(I)

# Indication vecteur et matrices []

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

```

1 #include <stdio.h>
2 int main(int argc, char *argv[])
3 for (int i = 0; i < argc; i++)
4 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; // !!! :-)

```



■ C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON &amp; R. KERYELL

196 / 361



## Indication vecteur et matrices []

- Généralisation dans le cas des matrices

```
1 identifiant [dim1] [dim2] ... [dimn]
```

-   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} }, { {1, 2, 3}, {0x13, 0x14, 0x15} }};
```

|               |    |               |
|---------------|----|---------------|
| 7ffffdfd751b0 | 00 | matrice[0][0] |
| 7ffffdfd751b1 | 01 | matrice[0][1] |
| 7ffffdfd751b2 | 02 | matrice[0][2] |
| 7ffffdfd751b3 | 10 | matrice[1][0] |
| 7ffffdfd751b4 | 11 | matrice[1][1] |
| 7ffffdfd751b5 | 12 | matrice[1][2] |



## (II)

## Indication vecteur et matrices []

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



-  En C éléments commençant à 0 dans chaque dimension

## Appels de fonctions & procédures



## (I)

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

```
1 identifiant(expr1, ..., exprn)
```

- 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)

■ C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON &amp; R. KERYELL



201 / 361

## 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)

■ C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON &amp; R. KERYELL



203 / 361

## 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 ( $c_i = a_i \oplus b_i$ )

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



202 / 361

## 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_u=parallel_u|_(u1<<ub);
parallel_out(parallel);
```

- Mettre 1 bit  $b$  à 0

```
1 parallel_u=parallel_u&~(u1<<ub);
2 parallel_out(parallel);
```

- Garder  $b$  bits de poids faible

```
1 c_u=u&((1<<ub)-u1);
```

- Extraire bits  $b$  à  $c$  de  $a$

```
1 d_u=(a>>ub)&((1<<(c_u+u1-u_b))-u1);
```

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



204 / 361

## Optimisations en binaire

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

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

(I)



## Optimisations en binaire

```
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 T0;
3 T0 = (T1 & 0x55555555) + ((T1 >> 1) & 0x55555555);
4 T0 = (T0 & 0x33333333) + ((T0 >> 2) & 0x33333333);
5 T0 = (T0 & 0x0f0f0f0f) + ((T0 >> 4) & 0x0f0f0f0f);
6 T0 = (T0 & 0x00ff00ff) + ((T0 >> 8) & 0x00ff00ff);
7 T0 = (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 (déttection 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` :

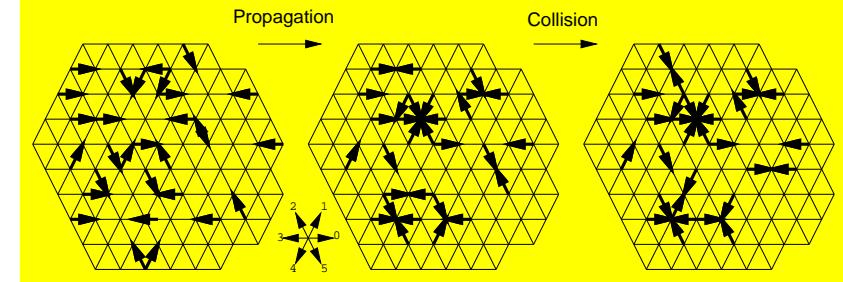
| quality | <code>q_a</code> | 0 | <code>q_x</code> | 0 | <code>q_y</code> |
|---------|------------------|---|------------------|---|------------------|
| 8       | 6                | 1 | 8                | 1 | 8                |

- `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 délimiteur 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 !

(I)

## Gaz sur réseau

- 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];
3 c_=lattice[TOP_LEFT]; // Particules qui montent à gauche
4 d_=lattice[LEFT]; // Particules qui vont à gauche
5 e_=lattice[BOTTOM_LEFT];
6 f_=lattice[BOTTOM_RIGHT];
7 s_=solid; // Une condition limite
8 ns_=~s;
9 r_=lattice[RANDOM]; // Un peu d'aléa
10 nr_=~r;
11 /* A triplet ? */
12 triple_=(a^b)&(b^c)&(c^d)&(d^e)&(e^f);
13 /* Doubles ? */
14 double_ad_=(ad&d~(b|c|e|f));
15 double_be_=(b&e&~(a|c|d|f));
16 double_cf_=(c&f&~(a|b|d|e));
17 /* The exchange of particles */
18 change_ad_=triple_|double_ad_|(r&double_be)|nr&double_cf);
19 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)

```

21 /* Where there is blowing , collisions are no longer valuable */
22 bl_=blow[N_DIR];
23 s_&=bl;
24 ns_&=~bl;
25 /* Effects the exchange where it has to do according the solid */
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

- Cylindre

■ C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

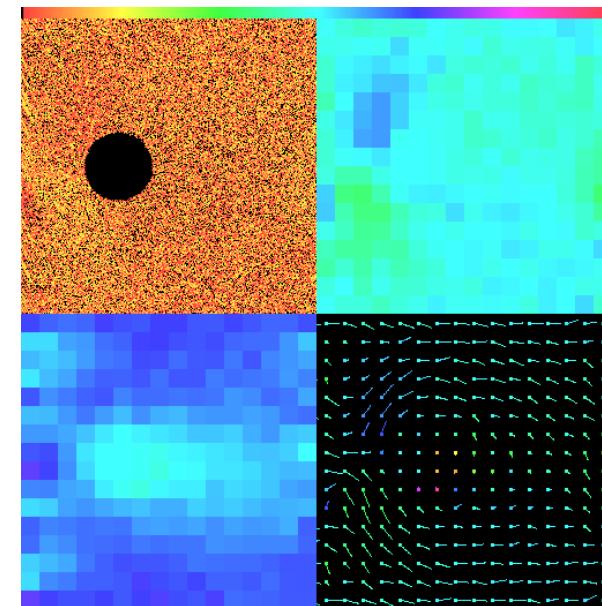
S. EVEN, S. GUELTON &amp; R. KERYELL



213 / 361

## (IV)

## Gaz sur réseau



## (V)

## Expressions logiques

- 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
- <sup>1</sup>  $expr_1 \& \& expr_2$
- ▶ Travaille sur opérateurs booléens

■ C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON &amp; R. KERYELL



215 / 361

## (I)

## Expressions logiques

- ▶ ~ 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 :  $1 \& 1 = 0$ , donc  
Vrai & Vrai = Faux... ☺  
Normalisation :  $!(\neg a \& \neg b)$
- Opérateur booléen ou inclusif

<sup>1</sup>  $expr_1 \text{ ou } expr_2$ 

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



216 / 361

■ C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON &amp; R. KERYELL

## 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`

■ C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON &amp; R. KERYELL

217 / 361



## Expression conditionnelle

(I)

1 `expression_cond ? expression_si-vraie : expression_si-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...

■ C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON &amp; R. KERYELL

219 / 361



## Opérateurs à effet de bord

(II)

- ▶ Aussi une expression
- Incrémentation/décrémentation
  - ▶ Existe en version
    - Préfixe : agit avant renvoi de valeur
      - 1 `++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
```



■ C avant C – UV2 INF 446

■ C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON &amp; R. KERYELL

220 / 361

# 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



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



(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

- Modèle mémoire & allocation
- Entrées-sorties
- GNOME
- 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



(II)

Coercition : conversion explicite de type

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.



(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

- Modèle mémoire & allocation
- Entrées-sorties
- GNOME
- 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



(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

- Modèle mémoire & allocation
- Entrées-sorties
- GNOME
- 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



cast en anglais

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



(I)

Transtypage explicite ou cast

1 *(type) ↴ expression*

Intérêt multiple

- Promotions

```
1 double ↴ a;
int ↴ m,n;
3 a= (double) ↴ m/(double) ↴ n;
```

- Permet troncatures simplement

```
1 int ↴ i, ↴ c;
double ↴ a, ↴ b;
3 a= (int) ↴ b;
c= (char) ↴ i;
```

- Toutes sortes de puissantes sordicités ☺ ↗



Transtypage explicite ou cast

```
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;
/* bricolé d en lui rajoutant 1
à son dernier octet de mantisse . */
```

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... ↗

(II)

Pointeurs & manipulation d'adresses

- & (préfixe) permet de récupérer adresse (référence) d'un objet
- * (préfixe) permet de déréférencer un pointeur (accéder à objets 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 }

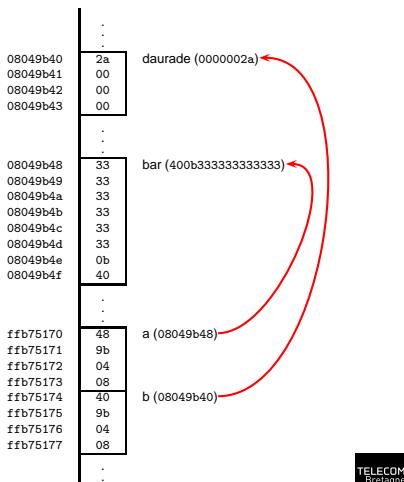
8 double ↴ bar;
int ↴ daurade;
10 ...
foo (&bar, &daurade);
```



(I)



Pointeurs & manipulation d'adresses



(II)

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

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

Liste d'expressions

(I)

1 $expression_1, expression_2$

- 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
- 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



■ C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON & R. KERYELL

233 / 361



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...

■ C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON & R. KERYELL

234 / 361

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)



■ C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON & R. KERYELL

235 / 361

Instructions élémentaires

(II)

```

1 volatile int entree;
2 /* Les labels sont bien pratiques pour simplifier
   la gestion des erreurs : */
4 gere_entree:
5 lire();
6 // Méchante attente active :
7 for(;;)
8 if(entree==3)
      goto gere_entree;

```



■ C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON & R. KERYELL



236 / 361

Tests if

(I)

Tests if

(II)

```

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

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



- 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)

Boucle while

(II)

- 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 'ç':
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...



```

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++;
3    while(i < f);
```

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

■ C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON & R. KERYELL



241 / 361

Boucle for

(I)

- Sucré 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 %d = %f\n", i, v[i]);
```



■ C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON & R. KERYELL

242 / 361

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

Branchements

```
1  goto label;
```

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

■ C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON & R. KERYELL



243 / 361

Instructions de saut

(II)

```
1  for(...)
2    for(...){}
3    ...
4    if(erreur_comise)
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)



■ C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON & R. KERYELL

244 / 361

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

■ C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON & R. KERYELL



245 / 361

- 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

■ C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

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



■ C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON & R. KERYELL

246 / 361

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 HISTO_ADDRESS(a, x, y)
2 (((a) << LOG2_XAXIS_HISTO_DEF) \
3 + ((x) << LOG2_YAXIS_HISTO_DEF)) \
4 + (y))

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

■ C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON &amp; R. KERYELL



247 / 361

# Inclusion de fichiers

(I)

```

1 /* Inclus ici un fichier d'en-tête 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



■ C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON &amp; R. KERYELL

248 / 361

# 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,x,y) \
2 (((a)<<(LOG2_XAXIS_HISTO_DEF) \
3 + ((x)<<(LOG2_YAXIS_HISTO_DEF)) \
4 + (y))
```

```
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);
```

C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON &amp; R. KERYELL



249 / 361

# 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 -D NDEBUG -D BITSIZE=64 -D 'X2(a)=(a)*(a)' -U RIEN ...
```

- Macrofonction ≠ fonction

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

```
2 #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

•Préprocesseur

# 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 is not
4 the value of the macro but the name of the macro... Who said C was a
5 simple language? */
6 #define STRINGIFY(symbol) STRINGIFY_SECOND_STAGE(symbol)
```

- Concaténation avec ##

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

C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON &amp; R. KERYELL



251 / 361

# 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

C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON &amp; R. KERYELL

C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON &amp; R. KERYELL



252 / 361

## Macroconstantes et macrofonctions

(V)

```

1 #ifndef NDEBUG
2 #define __assert(ex)---{if !(ex)){\
3 (void)fprintf(stderr,\n
4 "Assertion failed : file \"%s\", line %d\n", \
5 __FILE__, __LINE__);\n
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_FCPSOFT || defined VERIFY_MATCH
2 PLURAL int schpkno = 0;
3 PLURAL int ua, u, uv;
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 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



## Bibliothèques

- 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
  - ▶ ...

■ C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON &amp; R. KERYELL



257 / 361

(I)

## Bibliothèque standard

- Beaucoup de fonctions, types, variables, macros...
- 24 entêtes .h déclarant 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 !

■ C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON &amp; R. KERYELL



258 / 361

(I)

## Bibliothèque standard

- \_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 ☺

■ C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON &amp; R. KERYELL



259 / 361

(II)

## Contenu bibliothèque standard glibc

- 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

■ C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON &amp; R. KERYELL



260 / 361

(I)

## 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



## Le plan

- |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <ul style="list-style-type: none"> <li><b>1</b> Introduction           <ul style="list-style-type: none"> <li>• Bibliographie</li> <li>• Petite histoire</li> </ul> </li> <li><b>2</b> Langage           <ul style="list-style-type: none"> <li>• Généralités</li> </ul> </li> <li><b>3</b> Un ordinateur ? Mais qu'est-ce ?           <ul style="list-style-type: none"> <li>• Les bases de la numérologie</li> <li>• Calcul binaire</li> </ul> </li> <li><b>4</b> Déclarations           <ul style="list-style-type: none"> <li>• Généralités</li> <li>• Types des objets</li> <li>• Portée</li> </ul> </li> <li><b>5</b> Expressions           <ul style="list-style-type: none"> <li>• Sémantique opérateurs</li> <li>• Coercition de type</li> </ul> </li> <li><b>6</b> Instructions</li> <li><b>7</b> Préprocesseur</li> <li><b>8</b> Bibliothèques</li> </ul> | <ul style="list-style-type: none"> <li><b>9</b> Modèle mémoire &amp; allocation           <ul style="list-style-type: none"> <li>• Entrées-sorties</li> <li>• GNOME</li> <li>• Éditeur exemple d'Emacs</li> <li>• Tuning de son clavier</li> </ul> </li> <li><b>10</b> Compilation</li> <li><b>11</b> Modularisation</li> <li><b>12</b> Mise au point           <ul style="list-style-type: none"> <li>• Dévermineur (debugger)</li> <li>• Analyse mémoire</li> </ul> </li> <li><b>13</b> Extensions           <ul style="list-style-type: none"> <li>• Du C pour le graphisme</li> <li>• C++</li> </ul> </li> <li><b>14</b> Délires           <ul style="list-style-type: none"> <li>• Le bêtisier</li> <li>• Concours de programmes incompréhensibles</li> </ul> </li> <li><b>15</b> Conclusion</li> <li><b>16</b> Index</li> <li><b>17</b> Table des matières</li> </ul> |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|



## 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 noms (NSS) : où sont définis utilisateurs...
- Gestion utilisateurs et groupes
- Gestion configuration systèmes (paramètres matériels)



## 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

(I)

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)



■ C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON &amp; R. KERYELL

265 / 361



```
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

■ C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON &amp; R. KERYELL

266 / 361

# Chaînes de caractères

(I)

- Existents 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 TO,
2 #####const void* restrict FROM, size_t SIZE)
```

- Copie de chaîne



■ C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON &amp; R. KERYELL

267 / 361

```
1 char* strcpy(char* restrict TO,
2 #####const char* restrict FROM)
```

⚠⚠⚠ À place disponible endroit de destination... Source principale de piratage/bug ☺

- Copie limitée de chaîne de caractère

```
1 char* strncpy(char* restrict TO,
2 #####const char* restrict FROM, size_t SIZE)
```

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

☰ Nombreuses autres fonctions



■ C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON &amp; R. KERYELL

268 / 361

# strup : LA fonction la plus utile du C

(I)

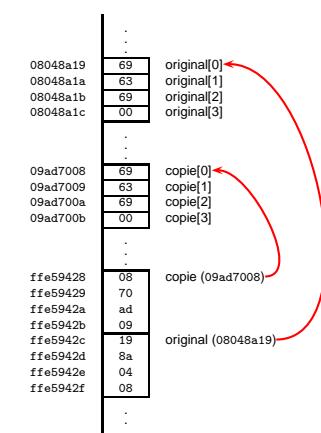
- Souvent besoin de dupliquer une chaîne (original alloué automatiquement sur pile ou non modifiable...)
- Demande mesure longueur, allouer avec `malloc()` sans oublier '\0' final puis recopier
- `strup()` fait tout ça d'un coup! ☺

```
1 char *original = "ici";
2 char *copie = strdup(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 Entrées-sorties
  - Modèle mémoire & allocation
  - Entrées-sorties
    - GNOME
    - 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



## Entrées-sorties sur stream

- 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

- 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...

(II)



## 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

- %ld 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

(I)

- 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, "
 "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



(III)

## Affichage formaté avec printf

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

- Ǝ 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

- 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)
{
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...

■ C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON &amp; R. KERYELL



277 / 361

(I)

## goto non locaux

- 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);
```

■ C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON &amp; R. KERYELL

278 / 361

(I)

## goto non locaux

```

1
2 int main()
3 /* La subtilité est dans les volatile nécessaires
4 pour prévenir le compilateur qu'il va se passer
5 des choses bizarre et donc que les variables mises
6 en registre doivent être synchronisées souvent
7 avec la mémoire sinon plantage en -O3... */
8 char * volatile ut = NULL; /* des données... */
9 char ** volatile ptr_tab = NULL;
10
11 /* Sauvegarde dans buff l'état courant : */
12 if (setjmp(buff))
13 // Affiche des infos
14 fprintf(stderr, "pointer_ut: %p\n", ut);
15 fprintf(stderr, "pointer_ptr_tab: %p\n", ptr_tab);
16 fflush(stderr);
17 free(ptr_tab[1]);
18 free(ptr_tab);
19 return 10;
```

■ C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON &amp; R. KERYELL



279 / 361

(II)

## goto non locaux

```

1 /* Ce bloc simule un catch () à la Java... */
2 /* Ici commence l'équivalent du bloc du try à la Java */
3 ut = (char *) malloc(100);
4 fprintf(stderr, "pointer_ut: %p\n", ut);
5 ptr_tab = (char **) malloc(sizeof(char*)*2);
6 fprintf(stderr, "pointer_ptr_tab: %p\n", ptr_tab);
7 ptr_tab[1] = ut;
8 /* Lance une exception à la Java */
9 throw_here();
10 /* On ne devrait jamais arriver jusqu'ici...
11 return 0;
12 }
```

■ C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON &amp; R. KERYELL

280 / 361

(III)



# 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

■ C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON &amp; R. KERYELL



281 / 361

# Glib

## Sous Linux/Debian

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

■ C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON &amp; R. KERYELL



283 / 361

- Modèle mémoire & allocation
- Entrées-sorties
- GNOME
  - Éditeur exemple d'Emacs
  - Tuning de son clavier
- 9 Compilation
- 10 Modularisation
- 11 Mise au point
  - Dévermineur (debugger)
  - Analyse mémoire
- 12 Extensions
  - Du C pour le graphisme
  - C++
- 13 Délires
  - Le bêtisier
  - Concours de programmes incompréhensibles
- 14 Conclusion
- 15 Index
- 16 Table des matières

# GNOME

- 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>

■ C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON &amp; R. KERYELL



282 / 361

# (I)

# Le plan

- 1 Introduction
  - Bibliographie
  - Petite histoire
  - GNOME
- 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

■ C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON &amp; R. KERYELL



284 / 361

## Éditeur de texte

- 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...



## (I)

## Emacs : LE éditeur

- 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

- ▶ 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



## (II)

## Emacs : LE éditeur

- 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 !)
    - 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 **verbetexte**...
- 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 ?

■ C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON &amp; R. KERYELL

289 / 361



## Concepts de base

(II)

### 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

■ C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON &amp; R. KERYELL

290 / 361



## 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 ☺

■ C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON &amp; R. KERYELL



## 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

■ C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON &amp; R. KERYELL

292 / 361



## 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...:-)
alias e=emacsclient -alternate-editor emacs
3 # Set the default editor
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

(I)

- 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 *window*
- 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

(II)

- 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 `perl gdb`)
  - PDB (Python) (M-x `pdb`)
  - bash debugger (M-x `bashdb`)
  - JDB (Java)
- Exécution suivi 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

```
1 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

- 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



(I)

## Nécessité d'un système de gestion de version

- 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 uchar 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...)



• Éditeur exemple d'Emacs

## Gestion de version avec Emacs

- 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 !
  - ~ Evite 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)



(I)

## 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 Exp \$ */

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 donersnse 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

(I)

- 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

- 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
-
- Modèle mémoire & allocation
 - Entrées-sorties
 - GNOME
- Éditeur exemple d'Emacs
- Tuning de son clavier
 - Compilation
 - Modularisation
 - Mise au point
 - Dévermineur (debugger)
 - Analyse mémoire
 - Extensions
 - Du C pour le graphisme
 - C++
 - Délires
 - Le bêtisier
 - Concours de programmes incompréhensibles
 - Conclusion
 - Index
 - Table des matières



Retour aux sources : clavier QWERTY

- 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/...)



(I)

Des accents !

- 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



(I)

Revamper un clavier AZERTY en QWERTY

- 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^{ème} 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
```



Touches Meta et Alt

- 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

```
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
```



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
- 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



(II)

Avec l'expérience

- 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



Principe

- Ordinateur capable d'exécuter que instructions machines
 - 1 uuuu1dr2,_456(r5)
 uuuuaddr1,r2,r3
 3 uuuujmp0x1234
- Besoin de traducteur langage haut niveau vers instructions machine ≡ compilation
 - Allocation des objets dans mémoire
 - Génération séquences d'instructions équivalentes au programme C



(I)

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



■ C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON & R. KERYELL

321 / 361

Compilateur gcc

(II)

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

■ C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON & R. KERYELL

322 / 361

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 indispensables 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
```



■ C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON & R. KERYELL

323 / 361

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`



■ C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON & R. KERYELL

324 / 361

Makefile

(III)

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

- Modèle mémoire & allocation
- Entrées-sorties
- GNOME
- 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



Modularisation

(I)

Fonctions

(I)

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

- 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`



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



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 Modularisation
- 10 Mise au point
 - Dévermineur (debugger)
 - Analyse mémoire
- 11 Extensions
 - Du C pour le graphisme
 - C++
- 12 Délires
 - Le bêtisier
 - Concours de programmes incompréhensibles
- 13 Conclusion
- 14 Index
- 15 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 déterminer 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
- `stepi` et `nexti` : 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
 - Dévermeur (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



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 :

■ C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON & R. KERYELL



341 / 361

(IV)

Mise au point allocation mémoire

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

■ C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON & R. KERYELL



343 / 361

Mise au point allocation mémoire

```
--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



■ C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON & R. KERYELL

342 / 361

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
 - Modularisation
 - Mise au point
 - Dévermineur (debugger)
 - Analyse mémoire
- 14 Extensions
 - Du C pour le graphisme
 - C++
 - Délires
 - Le bêtisier
 - Concours de programmes incompréhensibles
- 15 Conclusion
- 16 Index
- 17 Table des matières



■ C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON & R. KERYELL

344 / 361

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



Nouveaux formats de données

- 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



CUDA

<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)



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



C++

... en 1 transparent ☺

- Langage orienté objet basé sur C
 - Rajoute fonctions dans structures ↵ **classes**
 - Polymorphisme
 - Surcharge d'opérateurs du langage (+ - , () []...)
 - Exceptions
 - Patron 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.orienté-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
 - 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 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 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



Choses à (ne pas) faire

<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. »

(I)

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
 - 15 Concours de programmes incompréhensibles
 - 16 Conclusion
 - 17 Index
 - 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"] ⇐⇒ &(*(unix+"\021%six\012\0")) ⇐⇒
 "%six\012\0"
• (unix)["have"]+"fun"-0x60 ⇐⇒ "have"[1]+"fun"-0x60 ⇐⇒
 "fun"\u+_(‘a’-0x60) ⇐⇒ "fun"\u+_1 ⇐⇒ "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-Programming-Contest.html>



# Retour aux sources de l'informatique

(I)

**Vous avez désormais des connaissances suffisantes pour :**

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



- 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



## 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



- Modèle mémoire & allocation
- Entrées-sorties
- GNOME
- 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



## Le plan

# 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

- Modèle mémoire & allocation
- Entrées-sorties
- GNOME
- 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

```
#define || ou logique booléen négation
définition préprocesseur, 253 opérateur, 205
macroconstant préprocesseur, 253 opérateur
macrofonction préprocesseur, 253 ou logique booléen, 217
 () opérateur
 appel fonction négation
 opérateur, 201 opérateur, 205
 opérateur décrémentation
 appel fonction, 201 opération, 221
 * opération
 déréférencement de pointeur décrémentation, 221
 opérateur, 229 opérateur
 multiplication opérateur
 opérateur, 205 déréférencement de champ
 constante opérateur, 197
 préprocesseur opérateur
 constante, 109, 165 déréférencement de champ, 197
 compilation conditionnelle multiplication, 205
 préprocesseur, 257 addition
 préprocesseur opérateur, 205
 compilation conditionnelle, 257 opérateur
 #include opérateur
 inclusionfichier addition, 205
 préprocesseur, 249 / division
 préprocesseur incrémentation
 inclusionfichier, 249 opérateur, 205
 opérateur opérateur
 et logique incrémentation, 221
 opérateur, 205 < inférieur
 adresse d'un objet liste d'expressions
 opérateur, 229 opérateur, 233
 opérateur opérateur
 et logique, 205 liste d'expressions, 233
 adresse d'un objet, 229 <<
 - décalage à gauche
 et logique booléen opérateur, 77
 opérateur, 217 opérateur
 opérateur décalage à gauche, 205
 et logique booléen, 217 <=
 - inférieur ou égal
 C avant C – UV2 INF 446
```

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON &amp; R. KERYELL



360 / 361

```
int8_t, 105
int8_t, 105
uint8_t, 105
entier, 105
uint8_t
entier, 105
addition
+
opérateur, 205
opérateur
+, 205
adresse d'un objet
&
opérateur, 229
opérateur
&, 229
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(), 277
puts()
chaîne de caractères, 277
agrégat
expression
élémentaire, 189
élémentaire
expression, 189
C avant C – UV2 INF 446
```

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON &amp; R. KERYELL



360 / 361

```
opérateur, 217
opérateur
inférieur ou égal, 217
indice
opérateur, 197
opérateur
indice, 197
complément restreint
opérateur, 205
complément restreint
opérateur, 77
négation
opérateur, 205, 217
opérateur
complément restreint, 205
complément restreint, 77
negation, 205, 217
caractère
codage, 73
codage
caractère, 73
entier
int16_t, 105
uint16_t, 105
int16_t
entier, 105
uint16_t
entier, 105
16 bits
et logique
&
opérateur, 205
&, 205
opérateur
ou logique exclusif
-
opérateur
test d'égalité, 217
test d'égalité
opérateur, 217
opérateur
test d'égalité
opérateur, 217
opérateur
et logique booléen
&&
opérateur, 205
opérateur
supérieur, 217
supérieur
opérateur, 217
opérateur
supérieur ou égal, 217
supérieur ou égal
opérateur, 217
opérateur
ou logique booléen
|||
opérateur, 217
opérateur
décalage à droite
opérateur, 205
opérateur
décalage à droite, 205
opérateur
décalage à droite, 205
opérateur
expression conditionnelle
opérateur, 221
opérateur
expression conditionnelle,
221
opérateur
négation logique, 217
opérateur
négation logique, 217
64 bits
entier
int32_t, 105
uint32_t, 105
int32_t
entier, 105
uint32_t
entier, 105
32 bits
entier
int64_t, 105
uint64_t, 105
int64_t
entier, 105
uint64_t
entier, 105
8 bits
entier
TÉLÉCOM Bretagne/Info/HPCAS
S. EVEN, S. GUELTON & R. KERYELL
360 / 361
C avant C – UV2 INF 446
```

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON &amp; R. KERYELL



360 / 361

```
réel
type, 117
somme
flottant, 117
type
81
flottant, 117
intégral, 85
pointeur, 233
réel, 117
virgule
fixe, 117
ASCII
caractère
codage, 93
codage
caractère, 93
assembleur, 37
associativité
opérateur
priorité, 193
priorité
opérateur, 193
attribut
alias
restrict, 85, 169
auto
rangement, 85, 161
const
constante, 85, 165
constante
const, 85, 165
entier
long, 101
long long, 101
short, 101
signed, 101
unsigned, 101
long
entier, 101
long long
base 8
nombre
représentation, 69
représentation
nombre, 69
base 16
nombre
représentation, 69
représentation
nombre, 69
base 2
nombre
représentation, 69
nombre
représentation, 65
bibliothèque
libc, 13
standard, 261
base 2048
nombre
représentation, 73
représentation
nombre, 73
base 65536
nombre
représentation, 73
représentation
nombre, 73
baud
nombre
représentation, 65
représentation
nombre, 65
base 8
nombre
représentation, 69
représentation
nombre, 69
base 16
nombre
représentation, 69
représentation
nombre, 69
base 2
nombre
représentation, 65
TÉLÉCOM Bretagne/Info/HPCAS
S. EVEN, S. GUELTON & R. KERYELL
360 / 361
C avant C – UV2 INF 446
```



360 / 361

optimisation  
arithmétique, 209  
opérateur  
arithmétique, 205  
logique, 217  
représentation  
nombre, 65  
système  
unité, 65  
unité  
système, 65  
  
bit  
nombre  
représentation, 65  
représentation  
nombre, 65  
structure  
type, 145  
type  
structure, 145  
  
bool  
booléen  
entier, 109  
entier  
booléen, 109  
  
booléen  
bool  
entier, 109  
entier  
bool, 109  
intégral  
type, 85, 109  
type  
intégral, 85, 109  
  
boucle  
break  
instruction, 241, 245  
continue  
instruction, 241, 245  
do  
446  
  
■ C avant C – UV2 INF 446

instruction, 49, 245  
for  
instruction, 53, 245  
instruction  
break, 241, 245  
continue, 241, 245  
do, 49, 245  
for, 53, 245  
while, 49, 241  
  
while  
instruction, 49, 241  
branchement  
goto  
instruction, 245  
break  
aiguillage  
instruction, 241, 245  
boucle  
instruction  
aiguillage, 241, 245  
boucle, 241, 245  
  
C++  
langage  
objet, 353  
objet  
langage, 353  
calloc()  
allocation  
mémoire, 269  
mémoire  
allocation, 269  
large  
chaîne, 137  
type, 97, 109  
wchar\_t, 109  
  
caractère  
16 bits  
codage, 73  
ASCII  
codage, 93  
  
char  
instruction, 101  
CHAR\_BIT  
taille, 101  
chaîne  
large, 137  
type, 133  
codage  
16 bits, 73  
ASCII, 93  
ISO-10646, 97  
ISO-646, 93  
ISO-8859-1, 97  
ISO-8859-15, 97  
latin-1, 97  
latin-9, 97  
UNICODE, 97  
constante  
expression, 185  
expression  
constante, 185  
intégral  
type, 85, 89  
ISO-10646  
codage, 97  
ISO-646  
codage, 93  
ISO-8859-1  
codage, 97  
ISO-8859-15  
codage, 97  
large  
chaîne, 137  
type, 97, 109  
wchar\_t, 109  
latin-1  
codage, 97  
latin-9  
codage, 97  
taille

instruction, 49, 245  
type, 101  
CHAR\_BIT  
taille, 101  
chaîne  
large, 137  
type, 133  
codage  
16 bits, 73  
ASCII, 93  
ISO-10646, 97  
ISO-646, 93  
ISO-8859-1, 97  
ISO-8859-15, 97  
latin-1, 97  
latin-9, 97  
UNICODE, 97  
constante  
expression, 185  
expression  
constante, 185  
intégral  
type, 85, 89  
ISO-10646  
codage, 97  
ISO-646  
codage, 93  
ISO-8859-1  
codage, 97  
ISO-8859-15  
codage, 97  
large  
chaîne, 137  
type, 133  
vchar\_t  
large, 109  
  
case  
aiguillage  
instruction, 241  
instruction  
aiguillage, 241  
  
cast  
conversion explicite  
type, 229  
type  
conversion explicite, 229  
  
champ  
. . .  
opérateur, 197  
opérateur  
., 197  
champ de bits  
structure  
type, 145  
type  
structure, 145  
  
char  
caractère  
type, 101  
type  
caractère, 101  
  
char  
limite

CHAR\_BIT, 101  
type  
char, 101  
chaîne, 133  
intégral, 85, 89  
large, 97, 109  
vecteur, 133  
UNICODE  
codage, 97  
vecteur  
type, 133  
vchar\_t  
large, 109  
  
case  
aiguillage  
instruction, 241  
instruction  
aiguillage, 241  
  
cast  
conversion explicite  
type, 229  
type  
conversion explicite, 229  
  
champ  
. . .  
strdup(), 273  
longueur  
strdup(), 269  
longueur  
strdup()  
clonage, 273  
strlen()  
longueur, 269  
  
champ de bits  
structure  
type, 145  
type  
structure, 145  
  
char  
caractère  
type, 101  
type  
caractère, 101  
  
char  
limite

valeur, 101  
valeur  
limite, 101  
  
CHAR\_BIT  
caractère  
taille, 101  
taille  
caractère, 101  
  
chaîne  
caractère  
large, 137  
type, 133  
large  
caractère, 137  
type  
caractère, 133  
chaîne de caractères  
affichage  
puts(puts(), 277  
puts(puts())  
affichage, 277  
chaîne de caractères  
clonage  
strdup(), 273  
longueur  
strdup(), 269  
longueur  
strdup()  
clonage, 273  
strlen()  
longueur, 269  
  
clavier  
configuration, 317  
  
clonage  
chaîne de caractères  
strdup(), 273  
strdup()  
chaîne de caractères, 273  
  
codage  
16 bits  
caractère, 73



compilation conditionnelle  
#if  
préprocesseur, 257  
préprocesseur  
#if, 257  
  
complexe  
constante  
expression, 185  
double complex  
flottant, 125  
expression  
constante, 185  
float complex  
flottant, 125  
flottant  
double complex, 125  
float complex, 125  
long double complex, 125  
type, 85, 125  
long double complex  
flottant, 125  
type  
flottant, 85, 125  
  
complément restreint  
~  
opérateur, 205  
nombre  
négatif, 205  
négatif  
nombre, 205  
opérateur  
~, 205  
  
complément restreint  
~  
opérateur, 77  
nombre  
négatif, 77  
négatif  
nombre, 77  
opérateur  
446  
  
■ C avant C – UV2 INF 446

complément vrai  
nombre  
négatif, 77  
négatif  
nombre, 77  
complément à 2  
nombre  
négatif, 77  
négatif  
nombre, 77  
composition  
déclaration  
opérateurs, 153  
opérateurs  
déclaration, 153  
configuration  
clavier, 317  
const  
attribut  
constante, 85, 165  
constante  
attribut, 85, 165

expression  
binaire, 185  
caractère, 185  
complexe, 185  
décimale, 185  
entière, 185  
flottante, 185  
hexadécimal, 185  
octale, 185  
flottante  
expression, 185  
hexadécimal  
expression, 185  
octale  
expression, 185  
préprocesseur  
#define, 109, 165  
construction  
expression, 189  
continue  
boucle  
instruction, 241, 245  
instruction  
boucle, 241, 245  
contrôle de flot, 49  
conventions  
typographie, 25  
conversion  
type, 193  
conversion explicite  
coercion  
type, 229  
coercion  
coercion, 229  
conversion explicite  
cast  
type, 229  
cast  
type  
cast, 229  
court

entier  
short int, 101  
short int  
entier, 101  
  
DDD  
débogueur, 341  
  
default  
aiguillage  
instruction, 241  
instruction  
aiguillage, 241  
  
division  
/  
opérateur, 205  
opérateur  
/, 205  
  
do  
boucle  
instruction, 49, 245  
instruction  
boucle, 49, 245  
  
double  
double  
flottant, 113  
flottant  
double, 113  
  
double  
double  
flottant, 113  
flottant  
double, 113  
  
double complex  
complex  
flottant, 125  
flottant  
complexe, 125  
  
durée de vie  
objet, 161  
  
dynamic  
durée de vie  
objet, 161  
  
dynamique

allocation  
mémoire, 269  
mémoire  
allocation, 269  
  
débogage  
débogueur  
mise au point, 337  
mise au point  
débogueur, 337  
  
débogueur  
DDD, 341  
débogage  
mise au point, 337  
GDB, 337  
gdb, 13  
mise au point  
débogage, 337  
  
début  
main()  
programme, 33, 37, 197  
programme  
main(), 33, 37, 197  
  
décalage à droite  
>>  
opérateur, 205  
opérateur  
>, 205  
  
décalage à gauche  
<<  
opérateur, 205  
opérateur  
<, 205  
  
décimal  
constante  
expression, 185  
expression  
constante, 185  
  
déclaration  
composition  
opérateurs, 153



Purify  
mémoire, 341  
Valgrind  
mémoire, 341

effet de bord, 33  
affectation  
opération, 221  
opération  
affectation, 221

**else**

instruction  
test, 49, 241  
test  
instruction, 49, 241

Emacs  
texte  
éditeur, 289  
éditeur  
texte, 289

emacs  
éditeur de texte, 13

entier  
éditeur de texte, 13

16 bits  
int16\_t, 105  
uint16\_t, 105

32 bits  
int32\_t, 105  
uint32\_t, 105

64 bits  
int64\_t, 105  
uint64\_t, 105

8 bits  
int8\_t, 105  
uint8\_t, 105

attribut  
long, 101  
long long, 101  
short, 101  
signed, 101

unsigned, 101  
bool  
booléen, 109  
booléen  
bool, 109

court  
short int, 101

enum  
énumération, 109

int  
normal, 101  
int16\_t  
16 bits, 105

int32\_t  
32 bits, 105

int64\_t  
64 bits, 105

int8\_t  
8 bits, 105

intégral, 85, 101

16 bits, 105

32 bits, 105

32 bits, 105

32 bits, 105

64 bits, 105

unsignd, 101

wint\_t, 109

large  
long  
long int, 101

long attribut, 101

long int  
long, 101

long long  
attribut, 101

long long int  
très long, 101

nombre  
représentation, 61

non signé  
type, 101

normal  
int, 101

représentation  
fermeture

scanf(), 281  
sortie, 277

scanf()  
entrée, 281

sortie  
fprintf, 277

libération  
mémoire, 269

mémoire  
libération, 269

scanf(), 281  
sortie, 277

scanf()  
entrée, 281

sortie  
fprintf, 277

libération  
mémoire, 269

libération  
mémory, 269

inclusionfichier  
#include  
préprocesseur, 249

compilateur, 325

compilateur, 325

débogueur, 13

débogueur, 13

variable, 45

système d'exploitation, 25

branchement  
instruction, 245

instruction  
branchement, 245

incomplets  
type, 81

incomplète  
initialisation  
variable, 177

constante  
expression, 185

constante, 185

gettimeofday, 41

globale  
variable, 45

GNU  
système d'exploitation, 25

branchement  
instruction, 245

instruction  
branchement, 245

incomplets  
type, 81

incomplète  
initialisation  
variable, 177

hexadécimal  
constante  
expression, 185

constante, 185

nombre  
représentation, 69

nombre, 61  
short  
attribut, 101  
short int  
court, 101

signed  
attribut, 101

très long  
long long int, 101

type  
intégral, 85, 101  
non signé, 101

uint16\_t  
16 bits, 105

uint32\_t  
8 bits, 105

uint64\_t  
64 bits, 105

uint8\_t  
8 bits, 105

unsignd, 101

wint\_t  
large, 109

énumération  
enum, 109

entière  
constante  
expression, 185

entrée  
format  
scanf(), 281  
format, 281

entrée-sortie  
fclose()  
fermeture, 273

fclose()  
fermeture, 273

TELECOM Bretagne  
HPC ASI

■ C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON &amp; R. KERYELL

360 / 361

fclose(), 273  
fopen()  
ouverture, 273  
ouverture  
fopen(), 273  
stream, 273

enum  
entier  
énumération, 109  
énumération  
entier, 109

expression, 29, 181  
agréat  
élémentaire, 189

binnaire  
constante, 185

caractère  
constante, 185

complexe  
constante, 185

constante  
binnaire, 185  
caractère, 185  
complexe, 185  
décimale, 185

entièr, 185  
flottante, 185

hexadécimal, 185

octale, 185

construction, 189

décimale  
constante, 185

entièr  
constante, 185

flottante  
constante, 185

hexadécimal  
constante, 185

L-valeur  
élémentaire, 189

FIBONACCI, 157

fixe  
arithmétique  
virgule, 117

littérale  
élémentaire, 185

lvalue  
élémentaire, 189

octale  
constante, 185

symbolique  
élémentaire, 189

type, 173  
élémentaire  
agrégat, 189  
L-valeur, 189  
littérale, 185  
lvalue, 189  
symbolique, 189

expression conditionnelle  
?:  
opérateur, 221

opérateur  
?:, 221

extension  
nombre  
précision, 77

précision  
nombre, 77

fclose()  
entrée-sortie  
fermeture, 273

fermeture  
entrée-sortie, 273

entrée  
constante, 185

flottante  
constante, 185

hexadécimal  
constante, 185

L-valeur  
élémentaire, 189

fixe  
arithmétique  
virgule, 117

float  
flottant  
petit, 113

petit  
flottant, 125

flottant  
complexe, 125

arithmétique  
non associativité, 117

somme, 117

type, 85, 113, 117

double complex  
float complex, 125

long double complex, 125

type, 85, 125

double  
double, 113

double  
double, 113

double complex  
complexe, 125

dénormalisé  
nombre, 121

float  
petit, 113

float complex  
complexe, 125

long double  
étendu, 113

long double complex  
complexe, 125

nombre  
dénormalisé, 121

TELECOM Bretagne  
HPC ASI

■ C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON &amp; R. KERYELL

360 / 361

normalisé, 121

non associativité  
arithmétique, 117

normalisé  
nombre, 121

petit  
float, 113

réel  
type, 85, 113

somme  
arithmétique, 117

type  
arithmétique, 85, 113, 117

arithmétique, 85, 113, 117

complexe, 85, 125

réel, 85, 113

étendu  
long double, 113

flottante  
constante  
expression, 185

expression  
constante, 185

fonction, 29, 329

incomplet  
type, 149

paramètre, 49

type, 81, 149

incomplet, 149

fopen()  
entrée-sortie  
ouverture, 273

ouverture  
entrée-sortie, 273

for  
boucle  
instruction, 53, 245

instruction  
boucle, 53, 245

format  
entrée  
entrée-sortie

TELECOM Bretagne  
HPC ASI

■ C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON &amp; R. KERYELL

360 / 361

+++, 221

indentation  
programme  
présentation, 57

présentation  
programme, 57

indice  
[]  
opérateur, 197

opérateur  
[], 197

infixe  
opérateur, 189

inférieur  
<  
opérateur, 217

inférieur ou égal  
<=  
opérateur, 217

initialisation  
déclaration  
variable, 45, 177

incomplète  
variable, 177

variable  
déclaration, 45, 177

incomplète, 177

inline, 41  
instruction, 29, 237

instruction  
aiguillage  
break, 241, 245

aiguillage  
break, 241, 245

case, 241  
default, 241

default  
switch(), 241

switch()  
aiguillage, 241

test  
else, 49, 241

if, 49, 241

while  
break, 241, 245

continuer, 241, 245

do, 49, 245

for, 53, 245

while, 49, 241

branchement  
goto, 245

break  
aiguillage, 241, 245

boucle, 241, 245

break  
aiguillage, 241

continue  
boucle, 241, 245

continue  
boucle, 49, 245

default  
aiguillage, 241

default  
boucle, 49, 245

else  
test, 49, 241

for  
boucle, 53, 245

goto  
branchement, 245

if  
test, 49, 241

introduction  
programme  
présentation, 53

présentation  
programme, 53

retour de fonction  
return, 245

intégral  
arithmétique  
type, 85

booléen  
type, 85, 109

caractère  
type, 85, 89

entier  
type, 85, 101

type  
type

TELECOM Bretagne  
HPC ASI

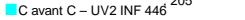
■ C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

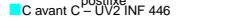
S. EVEN, S. GUELTON &amp; R. KERYELL

360 / 361



[] indice, 197  
 % modulo entier, 205  
 et logique &, 205  
 ou logique exclusif ^, 205  
 et logique booléen &&, 217  
 ou logique booléen ||, 217  
 ~ ou logique exclusif, 205  
 ! négation logique, 217  
 ~ complément restreint, 205  
 complément restreint, 77  
 négation, 205, 217  
 addition +, 205  
 adresse d'un objet &, 229  
 affectation =, 45  
 appel fonction (), 201  
 arithmétique binaire, 205  
 associativité priorité, 193  
 binaire arithmétique, 205  
 logique, 217  
 champ .. 197  
 comparaison, 217  
 complément restreint ~ 205  

**TÉLÉCOM Bretagne/Info/HPCAS**  
**S. EVEN, S. GUELTON & R. KERYELL**  
**360 / 361**



taille objet, 225  
 soustraction -, 205  
 supérieur >, 217  
 supérieur ou égal >=, 217  
 taille objet sizeof(), 225  
 test d'égalité ==, 217  
 test de différence !=, 217  
 opérateurs composition  
     déclaration, 153  
 déclaration composition, 153  
 opération  
     ++ incrémentation, 221  
     -- décrémentation, 221  
 affectation arithmétique, 221  
 modulo entier %, 205  
 multiplication \* , 205  
 négation  
     binaire, 217  
 arithmétique effet de bord, 221  
 affectation, 221  
 décrémentation --, 221  
 effet de bord affectation, 221  
 incrémentation ++, 221  
 ouverture entrée-sortie  
     fopen(), 273  
 préfixe fopen() entrée-sortie, 273  

**TÉLÉCOM Bretagne/Info/HPCAS**  
**S. EVEN, S. GUELTON & R. KERYELL**  
**360 / 361**



auto attribut, 85, 161  
 register attribut, 85, 161  
 static attribut, 85  
 register attribut rangement, 85, 161  
 rangement attribut, 85, 161  
 représentation base 8 nombre, 69  
 base 16 nombre, 69  
 base 2 nombre, 65  
 base 2048 nombre, 73  
 base 256 nombre, 73  
 base 65536 nombre, 73  
 baud nombre, 65  
 binaire récursion, 157  
 récursivité, 157  
 réel arithmétique type, 117  
     flottant type, 85, 113  
     nombre attribut entier, 101  
     signe valeur, 101  
     valeur limite, 101  
 bit nombre, 65  
 entier nombre, 61  
 hexadécimal nombre, 69  
 nombre base 8, 69  
 base 16, 69  
 base 2, 65  
 base 2048, 73  
 base 256, 73  
 saut instruction, 245  

**TÉLÉCOM Bretagne/Info/HPCAS**  
**S. EVEN, S. GUELTON & R. KERYELL**  
**360 / 361**



taille objet, 225  
 taille objet opérateur, 225  
 somme arithmétique flottant, 117  
     nombre arithmétique, 117  
 sortie format fprintf(), 277  
     fscanf() format, 277  
 sous-routine, 29 subtraction  
     - opérateur, 205  
     opérateur >, 217  
 standard bibliothèque, 261  
     attribut rangement, 85  
     rangement attribut, 85  
 static attribut  
     rangement, 85  
     rangement attribut, 85  
 strdup() chaîne de caractères clonage, 273  
     chaîne de caractères, 273  
 stream entrée-sortie, 273  
 strlen() chaîne de caractères longueur, 269  
     longueur chaîne de caractères, 269  
 système binaire unité, 65  
     unité binaire, 65  
     système d'exploitation  

**TÉLÉCOM Bretagne/Info/HPCAS**  
**S. EVEN, S. GUELTON & R. KERYELL**  
**360 / 361**



!= opérateur, 217  
opérateur !=, 217  
texte  
  Emacs éditeur, 289  
  éditeur, 289  
  Emacs, 289  
timer, 41  
très long  
  entier long long int, 101  
  long long int entier, 101  
type  
  agrége, 81  
  allocation pointeur, 129  
  arithmétique, 81 flottant, 85, 113, 117 intégral, 85, 113, 117 pointeur, 233 réel, 117  
  bit structure, 145  
  booléen intégral, 85, 109  
  caractère char, 101 chaîne, 133 intégral, 85, 89 large, 97, 109 vecteur, 133  
  cast conversion explicite, 229  
  champ de bits structure, 145  
  char caractère, 101  
■ C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON &amp; R. KERYELL



360 / 361

chaîne caractère, 133 coercition conversion explicite, 229 complexe flottant, 85, 125 conversion, 193 conversion explicite coercition, 229 conversion explicite cast, 229 entier intégral, 85, 101 non signé, 101 expression, 173 flottant arithmétique, 117 flottant, 85, 113 scalaire, 81 standard, 145 structure, 81, 137 bit, 145 champ de bits, 145 incomplet, 149 non finie, 141 taille variable vecteur, 133 typedef mommage, 173 union, 81, 141 vecteur, 81, 129 caractère, 133 linéarisation, 129 passage en paramètre, 133 taille variable, 133 void pointeur, 149 énumération intégral, 85, 109 large caractère, 97, 109 typedef mommage vecteur, 129 type mommage typedef mommage, 173 non finie structure, 141 typographie conventions, 25

uint16\_t 16 bits entier 16 bits, 105  
uint32\_t 32 bits entier 16 bits, 105  
uint64\_t 64 bits entier 64 bits, 105  
uint8\_t 8 bits entier 8 bits, 105 Valgrind  
UNICODE caractère codage, 97 caractère, 97  
union type, 81, 141  
unité binaire système, 65 système, 65  
UNIX système d'exploitation, 17  
unsigned attribut entier attribut, 101  
vector vecteur caractère type, 133 linéarisation type, 129  
■ C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON &amp; R. KERYELL



360 / 361

instruction, 49, 241  
instruction boucle, 49, 241  
wint\_t entier large entier, 109  
éditeur Emacs texte, 289  
texte, 289  
Emacs, 289  
éditeur de texte emacs, 13  
■ C avant C – UV2 INF 446

éditeur de lien, 37  
édition de lien, 37 élémentaire agrégat expression, 189 expression agrégat, 189 L-valeur, 189 littérale, 185 lvalue, 189 symbolique, 189 L-valeur expression, 189 littérale expression, 185 value expression, 189

symbolique expression, 189  
énumération entier enum, 109 enum entier, 109 intégral type, 85, 109 type intégral, 85, 109 étendu flottant long double, 113 long double flottant, 113



360 / 361

## 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
  - Modularisation
  - Mise au point
  - Dévermineur (debugger)
  - Analyse mémoire
- 14 Extensions
  - Du C pour le graphisme
  - C++
- 15 Délires
  - Le bêtisier
- 16 Concours de programmes incompréhensibles
- 17 Conclusion
- Index
- Table des matières

■ C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON &amp; R. KERYELL

■ C avant C – UV2 INF 446

TÉLÉCOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON &amp; R. KERYELL

361 / 361

• Table des matières

|                                                  |                     |
|--------------------------------------------------|---------------------|
| <b>1</b>                                         | <b>Introduction</b> |
| Le plan                                          |                     |
| Domaine d'utilisation                            |                     |
| Pourquoi un cours de C ?                         |                     |
| De nombreuses extensions                         |                     |
| Problématique pédagogique                        |                     |
| Hétérogénéité des élèves                         |                     |
| ● Bibliographie                                  |                     |
| Le plan                                          |                     |
| Ressources & Bibliographie                       |                     |
| Monitorat                                        |                     |
| ● Petite histoire                                |                     |
| Le plan                                          |                     |
| Préhistoire                                      |                     |
| C & Unix Story                                   |                     |
| Notations typographiques utilisées dans ce cours |                     |
| <b>2</b>                                         | <b>Langage</b>      |
| Le plan                                          |                     |
| ● Généralités                                    |                     |
| Le plan                                          |                     |
| Des instructions et expressions C...             |                     |
| ... en passant par les fonctions C...            |                     |
| ... aux programmes C                             |                     |
| Programme minimal en C                           |                     |
| Programme hello world en C                       |                     |
| Commentaires                                     |                     |
| Vers un programme exécutable                     |                     |
| Compilation d'un programme                       |                     |
| Déclarer des fonctions                           |                     |
| Types et expressions                             |                     |
| Variables et déclaration                         |                     |
| Affectation de variables                         |                     |
| Visibilité des variables                         |                     |
| La vie est un long fleuve tranquille             |                     |
| Tests                                            |                     |
| Boucle tant que                                  |                     |
| Boucle pour                                      |                     |
| Bien présenter les programmes                    |                     |
| ■ C avant C – UV2 INF 446                        |                     |

TELECOM Bretagne/Info/HPCAS

|                                                   |                                         |
|---------------------------------------------------|-----------------------------------------|
| ● Règle de base                                   | 52                                      |
| ● Indentation                                     | 53                                      |
| <b>3</b>                                          | <b>Un ordinateur ? Mais qu'est-ce ?</b> |
| ● Le plan                                         | 57                                      |
| ● Les bases de la numérologie                     |                                         |
| 4 Le plan                                         | 58                                      |
| 6 Un peu de numérologie                           | 59                                      |
| 7 Le plan                                         | 60                                      |
| 8 Numérotation binaire                            | 62                                      |
| 9 Unités binaires                                 | 63                                      |
| 10 Autres bases dérivées du binaire               | 65                                      |
| 13 Base 8 (octal)                                 | 66                                      |
| 14 Base 16 (hexadécimal)                          | 68                                      |
| 15 Base 256                                       | 69                                      |
| 16 Authentication avec mot de passe jetable S/Key | 70                                      |
| 23 Base 65536                                     | 71                                      |
| ● Calcul binaire                                  |                                         |
| 24 Le plan                                        | 72                                      |
| 24 Représenter des nombres négatifs               | 73                                      |
| 24 Nombres binaires en complément à 2             | 75                                      |
| <b>4</b>                                          | <b>Déclarations</b>                     |
| ● Le plan                                         | 77                                      |
| ● Généralités                                     |                                         |
| 30 Le plan                                        | 78                                      |
| 32 Objets en C                                    | 79                                      |
| 33 Types en C                                     | 80                                      |
| 34 Types arithmétiques en C                       | 81                                      |
| 35 Déclaration de variable                        | 82                                      |
| 37 Identificateurs                                | 83                                      |
| 40 Mots-clés réservés en C                        | 85                                      |
| 41 ● Types des objets                             |                                         |
| 43 Le plan                                        | 87                                      |
| 44 Caractères                                     | 88                                      |
| 46 Table des caractères ASCII                     | 89                                      |
| 47 Codage ISO-8859                                | 95                                      |
| 48 Codage UNICODE                                 | 96                                      |
| 49 Caractères de base                             | 97                                      |
| 50 Nombres entiers                                | 99                                      |



S. EVEN, S. GUELTON & R. KERYELL

361 / 361

• Table des matières

|                                                   |                     |
|---------------------------------------------------|---------------------|
| ● Types entiers étendus de taille connue          |                     |
| ● Caractères larges                               |                     |
| ● Booléens                                        |                     |
| ● Énumérations                                    |                     |
| ● Nombres flottants                               |                     |
| ● Conversion flottants → entiers à l'arrache      |                     |
| ● Nombres flottants ≠ réels !                     |                     |
| ● Algorithme de sommation de flottants            |                     |
| ● Vers des nombres flottants normalisés           |                     |
| ● Pourquoi des nombres flottants dénormalisés ?   |                     |
| ● Dénormalisation flottante                       |                     |
| ● Nombres complexes                               |                     |
| ● Pointeurs                                       |                     |
| ● Vecteurs                                        |                     |
| ● Vecteur en paramètre de fonction                |                     |
| ● Chaîne de caractères                            |                     |
| ● Chaînes de caractères larges                    |                     |
| ● Structures de données                           |                     |
| ● Type d'union de types                           |                     |
| ● Champs de bits                                  |                     |
| ● Autres types standards                          |                     |
| ● Types de fonctions                              |                     |
| ● Types incomplets                                |                     |
| ● FAQ Composition opérateurs de déclaration       |                     |
| ● Portée                                          |                     |
| ● Le plan                                         |                     |
| ● Portée des déclarations                         |                     |
| ● Où déclarer les variables                       |                     |
| ● Vie réelle des objets                           |                     |
| ● Objets non modifiables                          |                     |
| ● Des pointeurs à problèmes...                    |                     |
| ● Exemple de C : pointeurs                        |                     |
| ● Exemple de C : aliasing                         |                     |
| ● Pointeurs restreint                             |                     |
| ● Nomage de type                                  |                     |
| ● Expression de type                              |                     |
| ● Initialisations de variables                    |                     |
| ● Initialisations avec types compliqués           |                     |
| ● Variables temporaires                           |                     |
| <b>5</b>                                          | <b>Expressions</b>  |
| ● Le plan                                         | 105                 |
| ● Définition des expressions                      | 106                 |
| ● Expressions élémentaires littérales             | 107                 |
| ● Expressions élémentaires symbolique             | 109                 |
| ● Constantes agrégats                             | 112                 |
| ● Constantes L-valeurs (lvalues)                  | 114                 |
| ● Constructions d'expression                      | 116                 |
| ● Priorité des opérateurs                         | 118                 |
| ● Conversions de type                             | 119                 |
| ● Sémantique opérateurs                           | 121                 |
| ● Le plan                                         | 123                 |
| ● Opérateur .                                     | 124                 |
| ● Opérateur ->                                    | 126                 |
| ● Indicateur vecteur et matrices []               | 129                 |
| ● Appels de fonctions & procédures                | 132                 |
| ● Arithmétique                                    | 135                 |
| ● Arithmétique binaire                            | 139                 |
| ● Optimisations en binaire                        | 142                 |
| ● Applications codage binaire : multipspin-coding | 144                 |
| ● Application utilisant des additions 9 et 6 bits | 145                 |
| ● Gaz sur réseau                                  | 147                 |
| ● Expressions logiques                            | 149                 |
| ● Expression conditionnelle                       | 151                 |
| ● Opérateurs à effet de bord                      | 152                 |
| ● Taille d'un objet                               | 153                 |
| ● Coercition de type                              | 155                 |
| ● Le plan                                         | 157                 |
| ● Coercition : conversion explicite de type       | 158                 |
| ● cast en anglais                                 | 162                 |
| ● Transtypage explicite ou cast                   | 164                 |
| ● Pointeurs & manipulation d'adresses             | 165                 |
| ● Arithmétique sur pointeurs                      | 166                 |
| ● Liste d'expressions                             | 168                 |
| <b>6</b>                                          | <b>Instructions</b> |
| ● Le plan                                         | 171                 |
| ● Instructions                                    | 173                 |
| ● Instructions élémentaires                       | 174                 |
| ● Tests if                                        | 178                 |



TELECOM Bretagne/Info/HPCAS

361 / 361

• Table des matières

|                                           |                                |
|-------------------------------------------|--------------------------------|
| ● Aiguillage switch                       | 239                            |
| ● Boucle while                            | 240                            |
| ● Boucle do                               | 241                            |
| ● Boucle for                              | 242                            |
| ● Instructions de saut                    | 243                            |
| <b>7</b>                                  | <b>Préprocesseur</b>           |
| ● Le plan                                 |                                |
| ● Préprocesseur                           |                                |
| ● Préprocesseur C (CPP)                   |                                |
| ● Inclusion de fichiers                   |                                |
| ● Macroconstantes et macrofonctions       |                                |
| ● Compilation conditionnelle              |                                |
| <b>8</b>                                  | <b>Bibliothèques</b>           |
| ● Le plan                                 |                                |
| ● Bibliothèques                           |                                |
| ● Bibliothèque standard                   |                                |
| ● Contenu bibliothèque standard glibc     |                                |
| ● Modèle mémoire & allocation             |                                |
| ● Le plan                                 |                                |
| ● Modèle mémoire du C                     |                                |
| ● Allocation dynamique                    |                                |
| ● Chaînes de caractères                   |                                |
| ● strdup : LA fonction la plus utile du C |                                |
| ● Entrées-sorties                         |                                |
| ● Le plan                                 |                                |
| ● Entrées-sorties sur stream              |                                |
| ● Affichage formaté avec printf           |                                |
| ● Lecture formatée avec scanf             |                                |
| ● goto non locaux                         |                                |
| ● GNOME                                   |                                |
| ● Le plan                                 |                                |
| ● GNOME                                   |                                |
| ● Glib                                    |                                |
| <b>9</b>                                  | <b>Éditeur exemple d'Emacs</b> |
| ● Le plan                                 |                                |
| ● Éditeur de texte                        |                                |
| ● Emacs : LE éditeur                      |                                |
| ● ■ C avant C – UV2 INF 446               |                                |

TELECOM Bretagne/Info/HPCAS

S. EVEN, S. GUELTON & R. KERYELL

361 / 361

• Table des matières

|                                    |                           |
|------------------------------------|---------------------------|
| ● Dévermineur (debugger)           |                           |
| ● Le plan                          | 332                       |
| ● Débogueur                        | 333                       |
| ● Débogueur GNU gdb                | 335                       |
| ● Débogueurddd                     | 338                       |
| ● Analyse mémoire                  |                           |
| ● Le plan                          |                           |
| ● Mise au point allocation mémoire |                           |
| <b>13</b>                          | <b>Extensions</b>         |
| ● Le plan                          |                           |
| ● Du C pour le graphisme           |                           |
| ● Le plan                          |                           |
| ● CUDA                             |                           |
| ● Nouveaux formats de données      |                           |
| ● C++                              |                           |
| ● Le plan                          |                           |
| ● C++                              |                           |
| <b>14</b>                          | <b>Délires</b>            |
| <b>15</b>                          | <b>Conclusion</b>         |
| ● Le plan                          | 345                       |
| ● Conclusion                       | 346                       |
| <b>16</b>                          | <b>Index</b>              |
| ● Le plan                          | 348                       |
| <b>17</b>                          | <b>Table des matières</b> |
| ● Le plan                          | 349                       |
| ● Vous êtes ici !                  | 388                       |



TELECOM Bretagne/Info/HPCAS

361 / 361

TELECOM Bretagne/Info/HPCAS

361 / 361