2 - Quelques types simples

Ingrédients pour cuisiner les algorithmes

Application en C

J. Morinet-Lambert, M. Cadot, L. Pierron

http://iupalgo.free.fr/

© *Aucune diffusion autorisée en dehors du module d'enseignement*

13/10/04

Codage

J.Morinet-Lambert , M. Cadot, L. Pierron 2

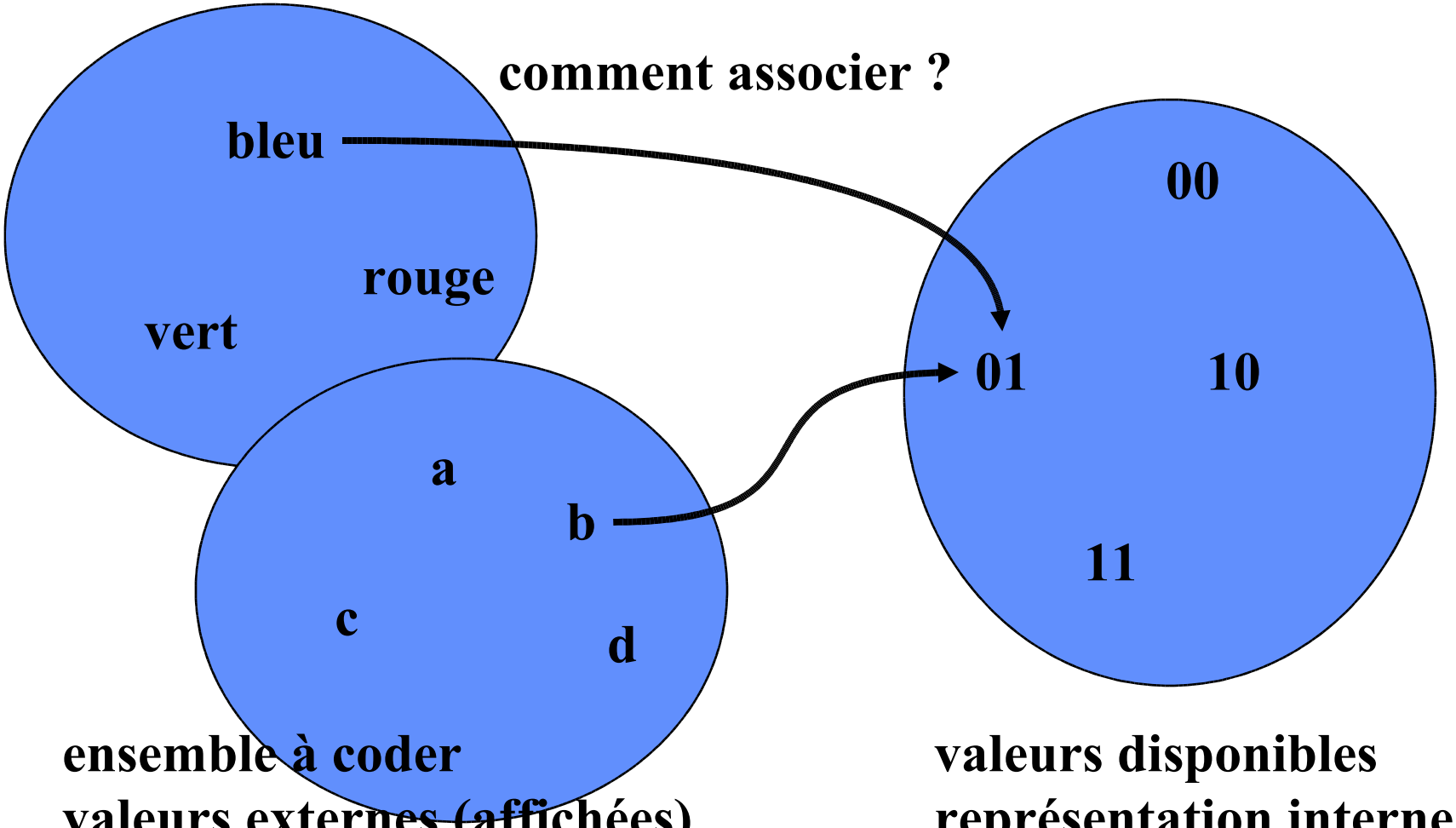
- transformation des informations
  - pour les rendre acceptables par une machine (traitement en binaire)
  - représentation interne à la machine, visibilité extérieure par utilisateur
  - exemple : taper 65 revient à taper 2 caractères ‘6’ puis ‘5’
    - interprétés ils deviendront la valeur entière codée 0000 0000 0100 0001 (ou 00 41 en hexa)
- normalisation : tendances **code ASCII, représentation des nombres**
  - rendre les applications exécutables sur des machines diverses
- codage => erreurs **de dépassement (overflow), d’arrondis, de précision**

Codage (2)

- Le codage interne en machine est binaire
  - chaque position binaire est appelée bit pour binary digit
  - pour faciliter la lecture on regroupe des bits par 8 : un octet ou BYTE
- On utilise la base hexadécimale plus compacte que la base binaire
  - les nombres sont alors préfixés par 0x
    - exemple 0x10 correspond à la valeur décimale 16, 0xFF à 255
- Codage :
  - pour ranger une valeur il faut savoir comment lui faire correspondre une configuration binaire : c'est le codage
  - pour interpréter la valeur lue dans un ou plusieurs octets il faut connaître le **mode de lecture** à appliquer : dépend du type de la donnée
    - scanf("%d", &n) taper 6 5 codé 0000 0000 0100 0001
    - scanf("%s",ch) taper 6 5 codé '6' '5' '\0' avec 0011 0110 0011 0101
  - sur un octet la configuration $36 peut correspondre à
    - un caractère '6' ou l'entier court 54... (3 * 16 + 6)
  - les nombres négatifs sont codés en complément à 2

Techniques de codage

J.Morinet-Lambert , M. Cadot, L. Pierron 4

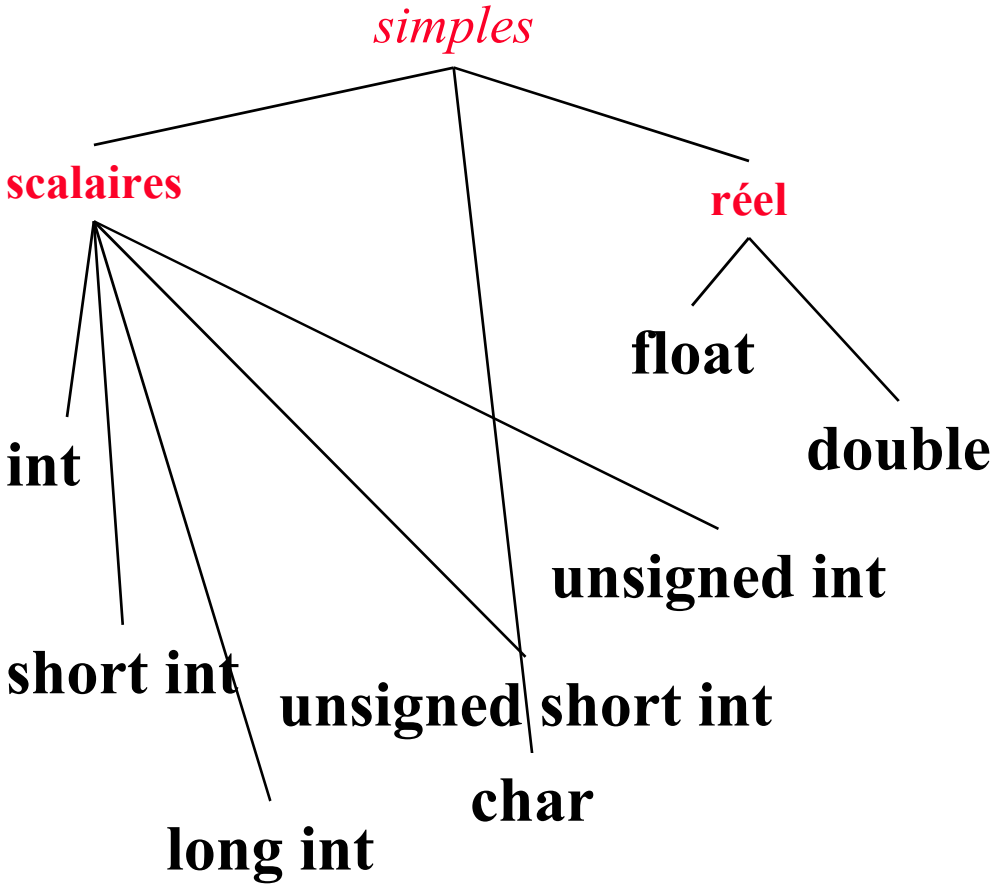


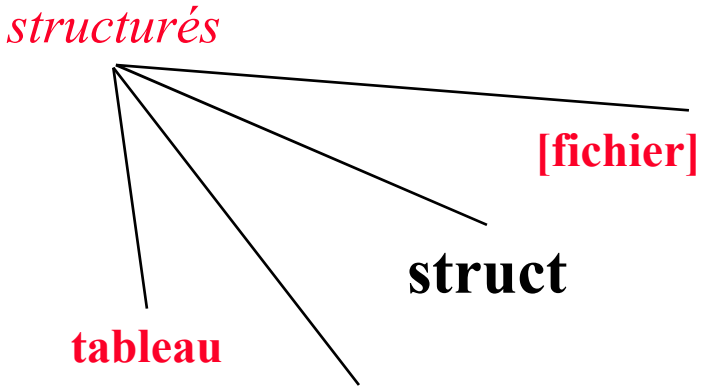
Les nombres négatifs

J.Morinet-Lambert , M. Cadot, L. Pierron 5

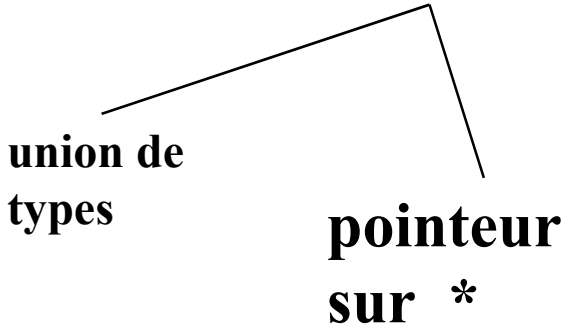
- exemple fictif sur 2 positions
- soit à coder 0,1,2,3 par les configurations 00 01 10 11
- avec 4 configurations on peut coder 4 nombres : 2 positifs , 2 négatifs
- quels nombres coder ?
- comment choisir les configurations ?
- solution 1) les compléments à 1
  - à coder 0 1 (positifs) -1 -0 (négatifs)
  - codé 00 01 10 11
  - évaluer la somme binaire de 1 + -0 = ? et -1 + -0 = ?
  - de plus on a deux codes pour la valeur 0
- on veut que la somme de 2 nombres opposés soit nulle
- solution 2) codage en complément à 2 retenu
  - à coder 0 1 -2 -1
  - codé 00 01 10 11
  - vérifier que la somme binaire 1 + -1 = 0

Classification des types de données





définis par l'utilisateur ou pour le système



pas de type booléen en C
ni de chaîne de caractères :-(

exemple


```
char c;                 /* déclarations  */
unsigned short int usi;
short int si;
c=0xA3;                  /* caractère non standard plage 128..255 */
usi= (short int) c ; /*transtypage forcé*/
si= usi ;               /* transtypage automatique */
/*afficher en char en int       en octal     en hexa     en dec */
c       £       65443 177643       ffa3           -93
usi             £     65443 177643       ffa3           -93
si      £       65443       177643       ffa3           -93
```

on a la même valeur : on lit différemment,
attention aux remplissages exemple : ff

Chaque type de donnée possède

- des propriétés
  - la taille varie (en nombre d'octets)
    - choisir en fonction des besoins
    - optimum de précision, temps de calcul, espace
  - choisir le format d'affichage sur l'écran
  - connaître les transpositions automatiques en C
    - ???= (int)x + (longint) y quel type de résultat ???
    - transtypage : forcer un changement de type : (type) var
- des opérateurs
  - attention à l'ordre d'exécution
- ses propres fonctions et procédures
  - dans des bibliothèques
  - qu'il faut nommer explicitement en début de programme
  - exemple : pour saisir des valeurs numériques

```
#include <stdio.h>
```

Les types entiers

définition	description	domaine min	domaine max	nb. d'octets
char	caractère	-128	127	1
short	entier court	-32768	32767	2
int	entier standard	-32768	32767	2
long	entier long	-2147483648	2147483647	4
[unsigned char	caractère	0	255	1]
unsigned short	entier court	0	65535	2
unsigned int	entier standard	0	65535	2
unsigned long	entier long	0	4294967295	4

- les limites sont définies dans le fichier <limits.h>
- attention : taille variable selon les stations : int sur 2 ou 4 octets
- comment savoir ? sizeof(*type*)

formats des entiers

- int : sur 4 octets
- short int : sur 2 octets

- %i ou %d pour des variables déclarées int
- %ni ou %nd pour afficher le nombre sur n chiffres
- %o ou %x pour afficher la valeur octale ou hexadecimale

- %c char

les opérateurs

J.Morinet-Lambert , M. Cadot, L. Pierron 11

- normaux
  / (division entière ou rationelle) * + -
  % (division entière) modulo (reste de la division entière)
- comparateurs
  == égal != différent
  <= inférieur ou égal >= supérieur ou égal
- opérateur logiques
  ! not && et || ou
- d'affectation
  += ajouter à …
- d'incrémentation ou de décrémentation : ATTENTION !!!!!
  i++ (après) ou --j (avant)
  n=7; j=3; n = ++j ; /* écriture douteuse après j vaut 4 n vaut 4*/
  i=6; m=3; m = i++ ; /* **attention** !!!! après i vaut 7 m vaut :6 */
  m = --i; /*i vaut 6, m vaut 6*/ m = i--; /* i vaut 5 m vaut 6 */

Les types réels

J.Morinet-Lambert , M. Cadot, L. Pierron 12

- **formats**
  - –%f décimal exemples : 3.1416 ou 123.4
  - –%e flottant : signe mantisse 10 exposant (signé)
  - –exemple : 1234 10^{-1} noté 1234e-1 ou 1234E-1 ou 1.234E2

définition	précision	mantisse	domaine min	domaine max	nb.d'octets
float	simple	6	3.4 * 10-38	3.4 * 10+38	4
double	double	15	1.7 * 10-308	1.7 * 10+308	8
long double	suppl.	19	3.4 * 10-4932	1.1 * 10+4932	10

- **attention** au point (en anglais) et à la virgule

Le type réel

J.Morinet-Lambert , M. Cadot, L. Pierron 13

- précision
- attention à l'utilisation de grands et petits nombres
  - parfois : (x /x)*y != (x * y) /x et ((x*y)/(y*x) != 1
  - conséquence :
    - ne pas tester les égalités x == y
    - tester la variation fabs(x-y) < epsilon
- opérateurs **binaires * / + -**
- **comparateurs cf. entiers < > = <= >= !=**
- fonctions **fabs pow (x,y) sqrt (root) arctan sin cos… log log10 exp**
- **dans la librairie math.h**
- **à ajouter lors de l'édition de lien :** gcc fich.c -o fich -lm
- transfert réels vers entiers
  - modf(x, &i) place la partie entiere de x dans i

les booléens : n'existent pas en C

J.Morinet-Lambert , M. Cadot, L. Pierron 14

- deux valeurs possibles : vrai et faux
- codage en C : faux : 0
  vrai : n'importe quel nombre différent de 0
- les opérateurs
  - mono-opérateur : not symbolisé par !
  - exemple c= !(15>12); /* que vaut c ? */
  - bi-opérateurs : and (et logique .)&& or (ou logique +) ||
- utilisé :
  - dans les instructions conditionnelles IF…
  - dans les conditions d'arrêt des itérations de type WHILE …
- exemples peu lisibles

  32 && 2.3 donne 1

  !65.34 donne 0

  0 || !(32 > 12) donne 0

Expressions booléennes tables de vérité

- NOT

V	F
F	V

majeur = ! mineur;

- AND

	V	F
V	V	F
F	F	F

recu = (moy>10)&&(!notelim) ;

- OR

	V	F
V	V	V
F	V	F

recale=(moy<10) ||(notelim);

Attention : si on a (V || expr1) ou (F && expr2)
le deuxième terme n'est pas évalué
car il n'est pas utile pour connaître le résultat (cf. exo TP2)

Les opérateurs booléens (2)

- affecter une valeur

  majeur = (age >= 18); masculin = (debnumss == 1);

- loi de DE Morgan (math)
  - not(x and y) équivaut à (not x) or (not y)
  - not(x or y) équivaut à (not x) and (not y)
- application informatique : transformations de texte de programme

  if (expressioncond) inst1; else inst2 ; /* devient */

  if (!(expressioncond)) inst2 ; else inst1;
  - utile surtout si inst1 est vide, exemple
  - si (trouve ou findeliste) /*rien faire */ sinon /* suivant*/ i++;
  - Devient if ((! trouve) && (! findeliste)) i++ ;

Mathématiques ≠ Informatique

J.Morinet-Lambert , M. Cadot, L. Pierron 17

- Codage réel : discontinuité des valeurs (selon la précision du codage)
- Codage des nombres : borné (selon la taille)
- Formules
  - linéariser les expressions : obligatoire
  - respecter les priorités

  a:=4; b:=5; printf("%d %d", a+b/5 , (a+b)/5); donne 5 puis 1.8
  - Parenthéser pour lever toute ambiguïté : a + b /5 ou (a+b)/5 ?
- Pas d'indice ni exposant
  - avant après : une seule étape possible

    u = u +1; est la traduction informatique de l'expression mathématique $u_n = u_{n-1} + 1$
  - cf... suite de Fibonacci $u_n = u_{n-1} + u_{n-2}$

    utiliser des variables intermédiaires u = w + z donne

    à l'étape suivante z =w; w= u; u = w+z;

Le type caractère

- Type scalaire, codé sur un octet
- Codage = valeur hexadécimale (0x00 à 0xFF ou en décimal 0..255)
- Normalisation
  - pour les codes entre 0..127
  - American Standard Code for Information Interchange
  - caractères non imprimables 0..31 (communication)
  - caractère non affichable : espace, backspace (32,127)
  - différence minuscule (a :97) et majuscule (A : 65)
  - norme étendue(unicode), codage sur 8bits pas toujours accepté (MIME et MIME etendu)
- **Exercice d'application**
  - **rechercher sur le web les tables ascii et mime (8859-1)**

Caractères (suite)

- Relation ordre
  - lexicographique, respecte ordre alphabétique
  - espace avant majuscule puis minuscules
  - exemple : dupon < duponD < dupond
- Déclaration char c;
- Affectation : c='K';
- Affichage : putchar(c) ;
- Saisie : int C; /* étrange mais vrai */
  C = getchar(); /* ou scanf("%c",C); */
- Remarque 1 : comme on connaît le format inutile de le spécifier dans ces modules spécialisés
- Remarque 2 : C n'est pas un langage fortement typé entier ou char peu importe on stocke une valeur hexadécimale correspondant au code

exemples d'affichages


```
char A = 225;
char B = 'a';
 int  C = '\a';    /* place la valeur du code */
 putchar('x');   /* afficher la lettre x      */
 putchar('?');   /* afficher le symbole ?     */
 putchar('\n');  /* retour à la ligne         */
 putchar(65);    /* afficher le symbole dont le  */
                 /* code vaut 65  (ASCII: 'A')  */
 putchar(A);     /* afficher la lettre avec   */
                 /* le code 225 (ASCII: 'ß')  */
 putchar(B);     /* affiche la valeur de B soit 'a'  */
 putchar(C);     /* beep sonore ??              */
```

Le type chaîne de caractères

J.Morinet-Lambert , M. Cadot, L. Pierron 21

- n'existe pas en C
- tableau de caractères dont le dernier est \0

'H'	'e'	'l'	'l'	'o'	\0	??	??

- Déclaration

  char ch[8]; /* nomchaine [nombre de caractères] */

  partie utile entre le premier (d'indice 0) et \0

  partie inutilisée après \0

  exemples char ch[6] = "Hello" ; /* correct */

- Dépassements

'H'	'e'	'l'	'l'	"o"	\0

  erreur de compilation char ch[4]="Hello";

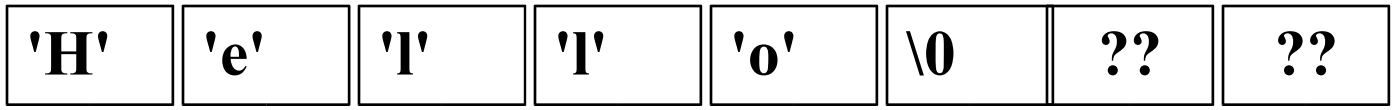
  erreur d'exécution char ch[5]="Hello";

'H'	'e'	'l'	'l'	'o'	\0

Affectation de chaînes

- Affecter une valeur en déclaration/'initialisation

  char ch[] = "AU REVOIR"; /* attention "chaines" 'c' pour les char */
- IO sur les chaînes
  - bibliothèque standard : scanf("%s",ch); printf("%s",ch);
  - bibliothèque <string.h> : gets(ch); ou affichage : puts(ch);
- Accès à un caractère
  - ch[i] le caractère d'indice i
  - utilisation c = ch[2]; /* c de type char */
  - i est un indice d'accès (obligatoirement de type scalaire)



Gestion de chaînes

- relation d'ordre (lexicographique)sur les chaînes
  - espace avant tout caractère alphabétique
  - DUPON <DUPOND<DUPONT
- fonctions de la bibliothèque <string.h>

  strcat(ch1,ch2); **/*** concatène ch2 à la suite de ch1 */

  l= strlen(ch); **/*** nb de caractères sans compter la marque de fin */

  b=strcmp(ch1,ch2); **/*** résultat >O si ch1 suit ch2, <0 si ch1 précède ch2 et O si égal */

  chu= strup(ch); **/*** transforme en majuscules */

  strcpy(x,y) ; /* recopie y dans x pas d'affectation*/
- ® **utiliser des fonctions :** ranger le résultat dans une variable

Déclarer des chaînes

J.Morinet-Lambert , M. Cadot, L. Pierron 24

1. char cha[10]; /* tableau de caracteres */
2. char chb[10]="bonjour"; /* initialisation */
3. char chc[]="bonsoir"; /* taille calculée à la compilation */

5. char *chd="salut";
6. char *che; /* pointeur sur le premier caractère */

- en fait c'est un tableau de caractères
- cha représente l'***adresse*** du premier élément du tableau
- même écriture scanf("%s",cha); et scanf("%s",che);
  printf("%s %s",cha, che);
- ATTENTION une instruction de saisie par chaîne (pb de buffer)

Adresse en mémoire

J.Morinet-Lambert , M. Cadot, L. Pierron 25

- ch représente l'adresse du premier caractère
- ch[1] va chercher la valeur de l'octet situé à l'adresse de ch +1 octet
- ch[8] va chercher la valeur de l'octet situé à l'adresse du premier caractère(ch) +8 octets

- conclusions : la mémoire est considérée comme un tableau d'octets
- on peut taper dans n'importe quel octet de la mémoire par erreur