Méthodes informatiques pour physiciens

introduction à C++ et
résolution de problèmes de physique par ordinateur

Leçon # 7 : L’ordinateur

Alessandro Bravar   Alessandro.Bravar@unige.ch
tél.: 96210   bureau: EP 206

assistants
Johanna Gramling   Johanna.Gramling@unige.ch
tél.: 96368   bureau: EP 202A

Mark Rayner   Mark.Rayner@unige.ch
tél.: 96263   bureau: EP 219

http://dpnc.unige.ch/~bravar/C++2015/L7
pour les notes du cours, les exemples, les corrigés, …

Plan du jour #7

Récapitulatif et corrigé de la leçon #6

Architecture de l’ordinateur

Représentation des nombres : nombres entiers
nombres à virgule flottante

Architecture binaire

« Interface graphique » DISLIN

Texte conseilles pour l’architecture de l’ordinateur
E. Lazard
Architecture de l'ordinateur

Référence DISLIN
H. Michels
The data plotting software DISLIN

Récapitulatif de la leçon #6

Adresse mémoire

opérateur &

Les pointeurs

déclaration et utilisation des pointeurs

```
int n;
int *ptr = &n;
```

tableaux et pointeurs

arithmétique des pointeurs

fonctions et pointeurs

Tableaux et pointeurs

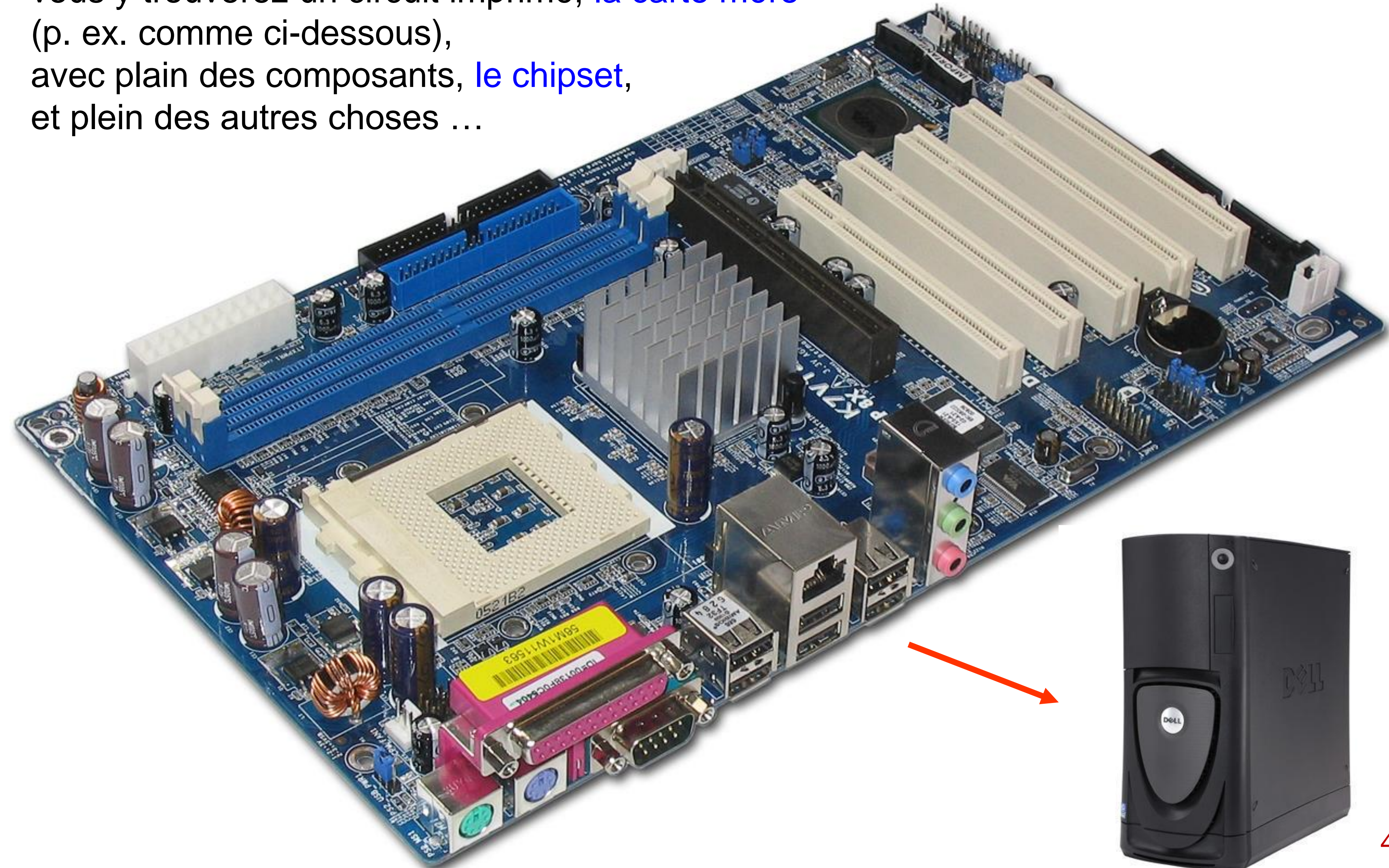
Passage des paramètres aux fonctions

Allocation dynamique de la mémoire

```
new
delete
```

L'ordinateur

Si vous ouvrez le boitier d'un ordinateur,
vous y trouverez un circuit imprimé, la carte mère
(p. ex. comme ci-dessous),
avec plain des composants, le chipset,
et plein des autres choses …

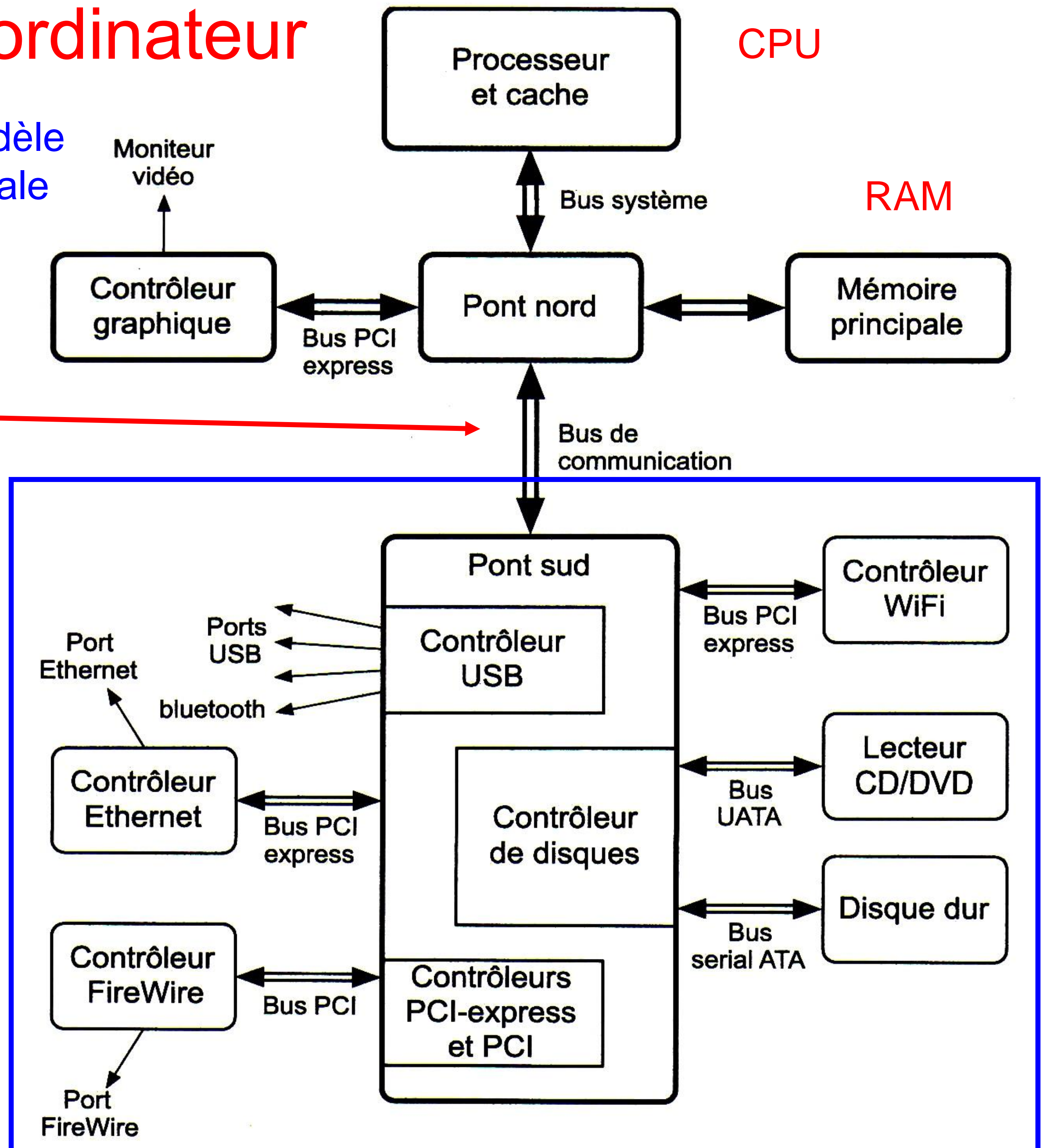


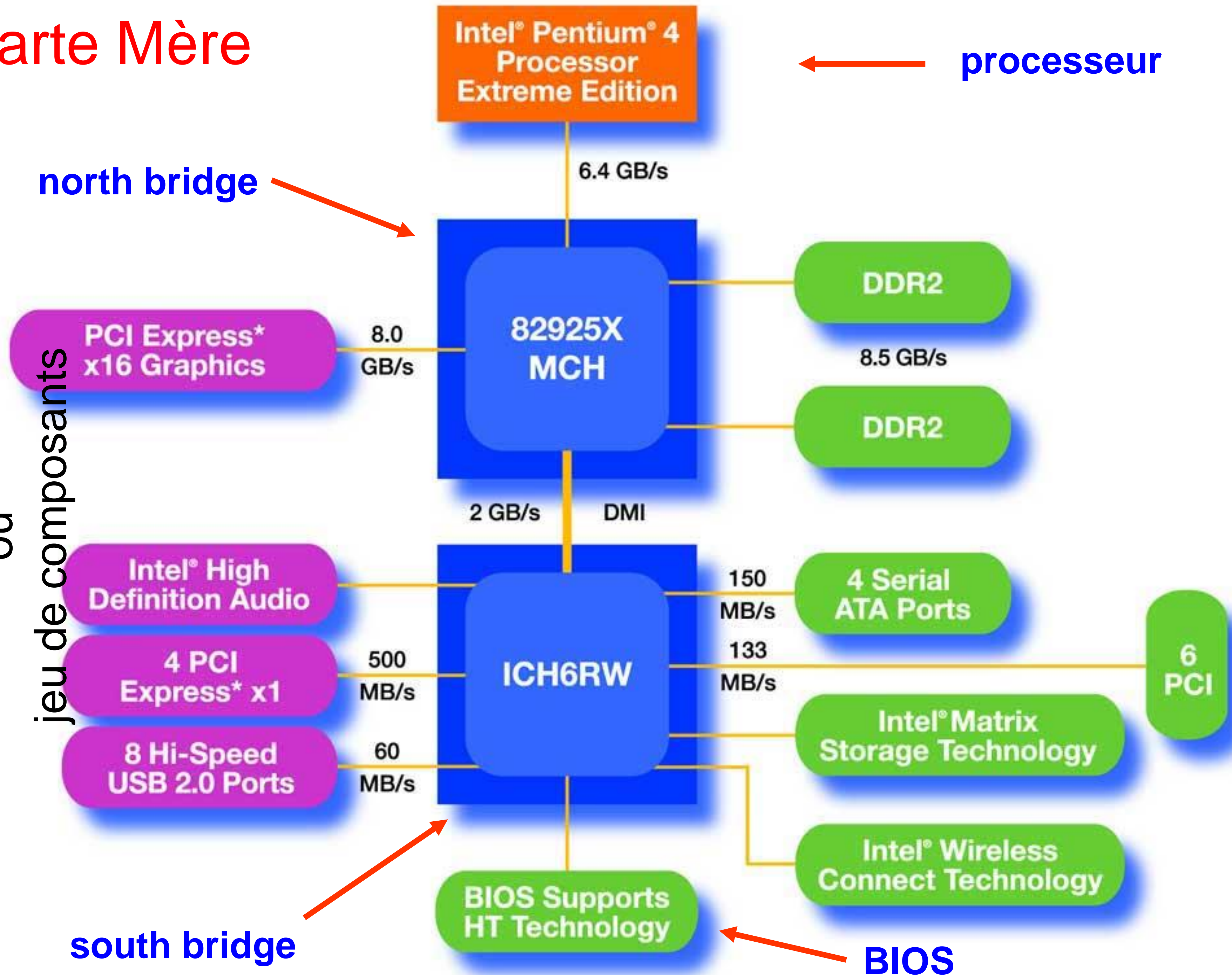
Structure d'un ordinateur

La structure dépend du modèle d'ordinateur, mais en générale tous les ordinateurs sont semblables.

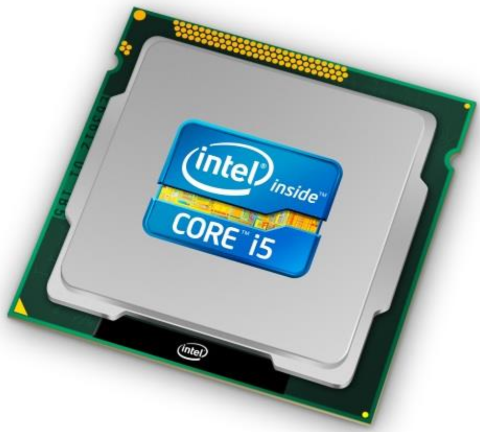
structure du bus
- lignes des données
- lignes des adresses
- lignes de contrôle

périphériques




Carte Mère
processeur
Intel® Pentium® 4 Processor Extreme Edition
6.4 GB/s
north bridge
82925X MCH
DDR2
8.5 GB/s
DDR2
PCI Express* x16 Graphics
8.0 GB/s
Chipset
ou
jeu de composants
2 GB/s
DMI
ICH6RW
Intel® High Definition Audio
4 PCI Express* x1
500 MB/s
8 Hi-Speed USB 2.0 Ports
60 MB/s
150 MB/s
4 Serial ATA Ports
133 MB/s
6 PCI
Intel® Matrix Storage Technology
Intel® Wireless Connect Technology
BIOS Supports HT Technology
south bridge
BIOS

Structure interne d'un processeur



registres :
cellules mémoire très proche du processeur

deux ALUs
(arithmetic and logic unit)
pour les nombres
- entiers
- virgule flottants

chef d'orchestre

Bus principal

Cache de niveau 1

Interface avec le bus

Registre PC

Registre RI

Banc de registres

Séquenceur

Tampon

Tampon

Horloge

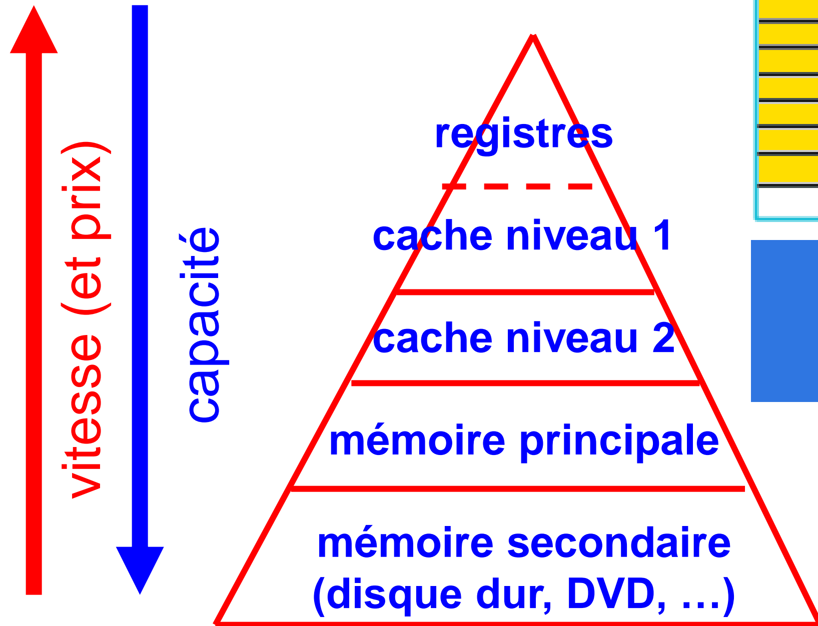
UAL

Tampon

Registre d'état

Organisation de la mémoire

deux fois la même structure
le processeurs peuvent s'échanger les informations



hiérarchie de la mémoire

Processeur

Cache L1

Cache L2

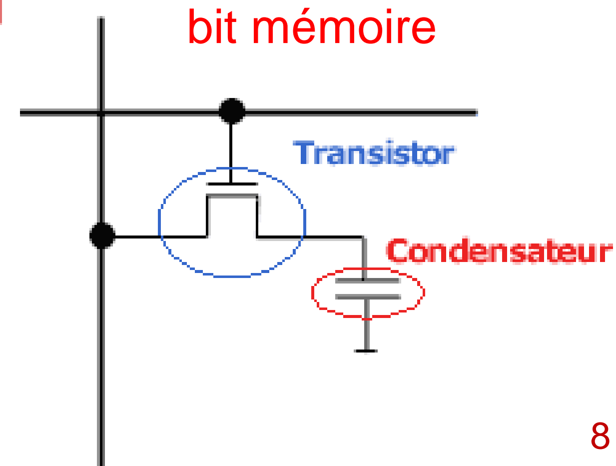
Mémoire vive (RAM)

Core d'exécution 1

Core d'exécution 2

4 Mo de cache L2 partagés dynamiquement entre les 2 cores

Mémoire principale



Le processeur et les instructions

Pour qu'un processeur puisse exécuter une instruction, il faut qu'il sache
1. de quelle instruction il s'agit
2. quelles sont les données sur lesquelles agir

C'est pourquoi une instruction sera stockée selon une méthode bien précise.

On divise ainsi une instruction en deux codes (p.ex. ADD r_0, r_1, r_2):
1. Le code opération, qui représente le type d'instruction
2. Le code opérande, qui représente les paramètres de l'instruction (adresse mémoire, constantes utilisées, registres ...)

Types principaux d'instructions :
1. Instructions d'opérations arithmétiques (addition, soustraction, division, multiplication)
2. Instructions d'opérations logiques
3. Instructions de transferts (entre différents registres, entre la mémoire et un registre, ...)
4. Instructions ayant rapport aux entrées et sorties

Etapes d'exécution :
les instructions sont exécutées toujours dans l'ordre suivant
1. Recherche de l'instruction
2. Lecture de l'instruction
3. Décodage de l'instruction
4. Exécution de l'instruction

Représentation des nombres

Pour comprendre le fonctionnement de l'ordinateur et les différentes types de variables utilisées, il faut comprendre comment les nombres sont traités.
L'ordinateur travaille uniquement avec deux valeurs, 0 et 1, appelées « bits » (bit = BInary digiT) et mémorisées dans des cases de taille limitée.
Cela limite la taille des nombres représentables.

Depuis toujours les ordinateurs calculent en binaire. On a à disposition 2 symboles (ou chiffres) 0 et 1, qui correspondent à 2 niveaux de tension dans l'ordinateur.

Toutes les données (aussi les instructions) sont représentées par une séquence de 0 et 1, typiquement regroupées en octets :

`char` et `bool` :	1 octet
`short` (int) :	2 octets
`int` :	4 octets
`float` :	4 octets
`double` :	8 octets

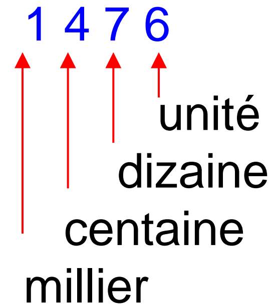
Ces séquences de 0 et 1 sont interprétées et traitées par l'ordinateur selon leur type.

Bases numériques

En général, dans la représentation des nombres, chaque chiffre a un significat bien défini selon sa position dans la séquence des *n* chiffres $x_{n-1}, x_{n-2}, \ldots x_1, x_0$:

$$X = x_{n-1}B^{n-1} + x_{n-2}B^{n-2} + \ldots + x_1 B^1 + x_0 B^0 = \sum_{i=0}^{n-1} x_i B^i$$

où *B* est la base de la représentation ($0 \leq x_i < B$).
Nous travaillons habituellement dans le système décimal dans lequel on utilise 10 chiffres (ou symboles), p.ex. le nombre



cf. sa représentation en chiffres romains MCDLXXVI
Ici on utilise des symboles différents pour les différentes puissances de 10.

La valeur maximale, que l'on peut exprimer avec *n* symboles, s'écrit avec tous les symboles égaux à *B* - 1, ce qui donne $B^n - 1$ (0 inclus) :

6 chiffres en base 10 :	$10^6 - 1 \rightarrow 999999$
16 chiffres (bits) en base 2 :	$2^{16} - 1 \rightarrow 65535$

Le nombre de chiffres pour représenter un nombre *N* est donné par int ($\log_B N$) + 1.

Base binaire

Chaque symbole d'un nombre binaire peut prendre seulement la valeur 0 ou 1, p.ex.:

$$10110101_2 = 1\times2^7 + 0\times2^6 + 1\times2^5 + 1\times2^4 + 0\times2^3 + 1\times2^2 + 0\times2^1 + 1\times2^0 = 181_{10}$$

longue séquence pour représenter un nombre (ici 8 chiffres !). Pour faciliter la lecture de nombres binaires, on utilise la base octale – 3 bits (8 symboles : 0, …, 7) et la base hexadécimale – 4 bits (16 symboles : 0, 1, …, 9, a, b, c, d, e, f) :

$$10110101_2 = 265_8 = b5_{16} = 181_{10}$$

Pour utiliser dans vos programmes des nombres en représentation

- octal : `0nnn`
- hexadécimal : `0xnnn`
- (pas de représentation binaire !)

```
int a = 0265;
cout << a;        imprimera 181

cin >> a;         0xb5 depuis le clavier
cout << a         imprimera 181
```

voir Base.cpp

Pour passer de la base 10 à la base 2, il faut décomposer le nombre en puissance de 2 :

181 / 2 = 90 + 1
90 / 2 = 45 + 0
45 / 2 = 22 + 1
22 / 2 = 11 + 0
11 / 2 = 5 + 1
5 / 2 = 2 + 1
2 / 2 = 1 + 0
1 / 2 = 0 + 1

$\Rightarrow$ 10110101

bit de poids faible : le moins significatif

bit de poids fort : le plus significatif

Arithmétique binaire

Les opérations arithmétique en binaire (algèbre de Bool) se réalisent de la même façon qu'en décimal, sauf qu'on a seulement deux chiffres à disposition pour les calculs.

addition :

0 + 0 = 0
0 + 1 = 1
1 + 0 = 1
1 + 1 = 0 et 1 de retenue
(cf. 7 + 5 = 2 et 1 de retenue !)

p. ex.

	binaire	décimal
	1 1 0 1 1 1 0	(110)
+	1 0 1 0 1 1	(43)
=	1 0 0 1 1 0 0 1	(153)

Il faut toujours additionner les nombres deux a deux : a + b + c = (a + b) + c !

Si l'addition de bits les plus significatifs produit une retenue (1), il y a débordement. L'ordinateur n'a pas assez de *cases* à disposition pour enregistrer la retenue : on ignore la retenue et continue l'exécution sans signaler l'erreur ! ⇒ erreur de débordement

voir Short.cpp

p.ex. avec 8 bits : 11111010 (250) + 1010 (10) = ~~1~~00000100 (260) → 100 (4)

(00000100 : 8 bits)

soustraction :

0 – 0 = 0
1 – 0 = 1
0 – 1 = 1 et 1 de retenue
1 – 1 = 0

l'utilisation des nombres négatifs est plutôt compliquée :
au lieu de soustraire p.ex. b de a : a – b, on additionne le complément de b à a : a + (– b)

multiplication :

La table « pythagorique » pour la multiplication est plutôt simple:

0	0
0	1

```
p. ex. :        1 0 1 0 1 1 0 × 1 1 0 1 0   (85 × 26)
              ---------------------------
                1 0 1 0 1 1 0
décalage          1 0 1 0 1 1 0
              -----------------
addition      1 0 0 0 0 0 0 1 0
décalage            0 0 0 0 0 0 0
décalage              1 0 1 0 1 1 0
              ---------------------
addition      1 0 0 0 1 0 1 1 1 1 0
décalage                0 0 0 0 0 0 0
               -----------------------
addition      1 0 0 0 1 0 1 1 1 1 0 0      (2210)
```

La multiplication entre entiers se réduit à une série de décalages et additions (opérations très simples pour le processeur).

La multiplication en binaire allonge fortement les nombres : multiplier deux nombres de n bits peut donner un résultat sur $2n$ bits ; on peut épuiser l'espace à disposition très rapidement.

multiplication par 2 : décalez tous les chiffres à gauche d'une position

division :

La division en binaire marche comme dans toute autre base y compris la base 10 :

dividende = diviseur * quotient + reste (p. ex. 57 / 5 → 57 = 5 * 11 + 2

On soustrait le diviseur du dividende en commençant par les bits de poids fort.
Elle consiste en une série de soustractions et décalages pour donner un quotient et un reste.

p. ex. 1 1 1 0 0 1 : 1 0 1 = 1 0 1 1 + reste 1 0 (57 / 5 = 11 + reste 2)

```
  1 1 1        |  1
- 1 0 1        |
    1 0 0      |  0
    1 0 0 0    |  1
  - 1 0 1      |
      1 1 1    |  1
    - 1 0 1    |
        1 0    | reste
```

(le résultat de la division est toujours 1 + reste)

division par 2 : décaler les chiffres à droite d'une position et
supprimer le dernier chiffre

Les nombres négatifs

Représentation « signe et valeur absolue »

Etant donné qu' on n'a pas à disposition le signe moins pour indiquer un nombre négatif, on utilise le bit de poids fort pour indiquer le signe des nombres :

0 pour le positifs, 1 pour les négatifs

La plage des valeurs passe de l'intervalle $[0, 2^n - 1]$ à l'intervalle $[-(2^{n-1} - 1), 2^{n-1} - 1]$ et on se retrouve avec deux représentations possible pour le 0.

cf. `unsigned int` - plage $[0, 2^n - 1]$
et `(signed) int` - plage $[-2^{n-1}, 2^{n-1} - 1]$

Les opérations arithmétiques sont plus compliquées qu'avec des nombres positifs car les règles d'addition des bits ne s'appliquent plus : addition de valeurs absolues et non de nombres eux-mêmes.

Au lieu de soustraire p. ex. b de a (a – b), on additionne le complément de b, C(b), à a :
b + C(b) = 0 (!) et a – b = a + C(b) (C(b) = – b, presque trivial en décimal avec le signe –).

Pour exprimer les nombres négatifs et effectuer des calculs sur les ordinateurs, on utilise la représentation en complément à 2.

Complément à 2

La représentation standard sur les ordinateurs pour exprimer les nombres entiers négatifs est la représentation en complément à 2 :

1. inversez tous les bits (complément à 1 : 0 → 1 et 1 → 0)
2. additionnez 1 en laissant tomber une éventuelle retenue finale

p.ex. : 52 = 0 0 1 1 0 1 0 0 → – 52 = 1 0 1 1 0 1 0 0 !

$C(52)_1$ = 1 1 0 0 1 0 1 1 → $C(52)_2 = C(52)_1 + 1$ = ~~1~~ 1 1 0 0 1 1 0 0 ≠ – 52

Il faut toujours spécifier le nombre de bits pour cette représentation.
Le bit de poids fort est toujours égal à 1 (comme pour le nombres négatifs).
Sur *n* bits la plage des valeurs admissibles passera à $[-2^{n-1}, 2^{n-1} - 1]$
(il y a une seule représentation pour le 0, sur 8 bit 10000000 est l'entier le plus petit).
Pour soustraire un nombre d'un autre, on additionne au deuxième le complément à 2 du premier.

représentation sur 8 bits

127 :	0 1 1 1 1 1 1 1
2 :	0 0 0 0 0 0 1 0
1 :	0 0 0 0 0 0 0 1
0 :	0 0 0 0 0 0 0 0
-1 :	1 1 1 1 1 1 1 1
-2 :	1 1 1 1 1 1 1 0
-128 :	1 0 0 0 0 0 0 0

propriétés du complément à 2 :

$$X + C_2(X) = 2^n$$
$$C_2(C_2(X)) = X$$

p. ex. : 118 – 36 = 118 +(- 36) = 82

118 :	0 1 1 1 0 1 1 0	
+(-36) :	1 1 0 1 1 1 0 0	$C_2(36)$
82 :	~~1~~ 0 1 0 1 0 0 1 0	

et la retenue finale est ignorée

Débordement

Imaginez un compteur à 6 chiffres :
incrémentation après incrémentation vous attendriez la valeur 999999.
Si vous y ajoutez 1, le compteur sera remis a zéro !
et on recommence compter à partir de 0, puis 1, 2, etc.

Comme l'espace mémoire (nombre de bits) pour enregistrer une variable est fini, on rencontre le même problème avec l'ordinateur.

Considérons une variable entier (`signed`) `int` sur 4 octets :
la valeur la plus grande autorisé est 2'147'483'647 (7F FF FF FF en hexadécimal).
(le premier bit, bit de pois fort, est utilisé pour le signe)
Si l'on addition 1 à cette variable, la nouvelle séquence de bits (80 00 00 00) correspond au plus grand nombre négative autorisé -2'147'483'648 (complément à 2 !), puis on décompte jusqu'à la valeur 0.

p.ex. :

```
int n = 2147483647;
n = n + 10;
cout << n;
```

affichera -2147483639 voir Debordement.cpp

Par contre, avec des variable à virgule flottante la valeur `1.#inf` ou `-1.#inf` sera affecté à la variable en question pour toujours (sans message d'erreur).

p.ex. :

```
double x = -1./0.;
x = x + 10.;
cout << x;
```

affichera `-1.#inf`

Nombres réels

Dans de nombreuses applications, on ne peut se contenter de calculer avec les nombres entiers :

on a besoin de nombres sortant de l'intervalle de représentation des entiers

on utilise des nombres décimaux

L'espace de stockage d'un nombre (# de bits) étant limité, tous les nombres réels possibles ne sont pas représentables et il y a une limite à la précision des calculs.

Représentations des nombres décimaux :

il faut définir l'emplacement d'une virgule séparant la partie entière et la partie décimale, les bits à droite de la virgule décimale correspondent aux puissances négatives de 2, le bits à gauche aux puissances positives.

$$X = x_{n-p-1}2^{n-p-1} + x_{n-p-2}2^{n-p-2} + \ldots + x_1 2^1 + x_0 2^0 + x_{-1}2^{-1} + \ldots + x_{-p}2^{-p} = \sum_{i=-p}^{n-p-1} x_i 2^i$$

Pour représenter un nombre décimal en base 2, exprimez le nombre en puissances de 2 comme pour les entiers p.ex. sur 8 bits, si la virgule est placée au milieu :

4,5 = 4 + 5/10 = 4 + 1/2 → 0 1 0 0 1 0 0 0

6,75 = 6 + 7/10 + 5/100 = 4 + 2 + 1/2 + 1/4 → 0 1 1 0 1 1 0 0

0,375 = 0 + 3/10 + 7/100 + 5/1000 = 0 + 1/4 + 1/8 → 0 0 0 0 0 1 1 0

La plage de nombres représentables est assez limitée.

Représentation en virgule flottante

Considérons la notation scientifique, p.ex. 234,567 $\rightarrow$ 2,34567 $\times 10^2$;
la virgule est toujours placé après le premier chiffre $\neq 0$ (nombre *normalisé*).
Le nombre est représenté par une mantisse (2,345657) et un exposant (2).

On procède de la même façon en binaire. A noter que la mantisse commence toujours par 1, pour exploiter mieux tous les bits à disposition on ne l'écrit pas et on n'exprime que la partie décimale, la *pseudo-mantisse*.

convention IEEE 754 :

les nombres en simple précision (float) possèdent une pseudo-mantisse sur 23 bits (correspondant aux puissances négatives de 2, de 2^{-1} à 2^{-23}), un exposant sur 8 bits et un bit de signe, la plage va à peu près de 10^{-38} à 10^{38},
les nombres en double précision (double) ont une pseudo-mantisse sur 52 bits (2^{-1} à 2^{-52}), un exposant sur 11 bits et un bit de signe, la plage va à peu près de 10^{-308} à 10^{308} :

bit de signe exposant (8 ou 11 bits) pseudo-mantisse (23 ou 52 bits)

pour exprimer les exposants négatifs, sa valeur réelle est décalée par excès d'un biais de 127 (1023 en précision double)

p.ex. : $500 = (1 + 1/2 + 1/4 + 1/8 + 1/16 + 1/64) \times 2^8 \rightarrow$ 0 10000111 11110100000…
$-3,5 = -(1 + 1/2 + 1/4) \times 2^1 \rightarrow$ 1 10000000 110000…

Précision

Contrairement aux nombres mathématiques, les nombres informatiques ont une précision finie. Les entiers sont compris dans un intervalle fini et les nombres à virgule flottante ont une précision et un intervalle limités :

au-delà de la valeur maximale $\rightarrow$ erreur d' *overflow*

en-deçà de la valeur minimale $\rightarrow$ erreur d' *underflow*

Les nombres et les calcules en virgule flottante sont imprécis en raison du nombre limité de bits de la représentation : tous les nombres réels ne sont pas représentables. Cette imprécision est qualifiée d'erreur d'arrondi.

En précision simple la plage de valeurs s'étende de ~ -10^{38} a ~$+10^{38}$, mais on a à disposition seulement 32 bits pour exprimer tous le nombres entre -10^{38} a $+10^{38}$ et la plus petit fraction représentable est 2^{-23} (~10^{-7}) ;

$\rightarrow$ on peut s'attendre à une précision relative de ~10^{-7} .

En précision double la précision relative est de ~ 2×10^{-16} .

p. ex.: 1'000'000,100 et 1'000'000,050 sont représenté par la même séquence de bits et p. ex. la limite (dérivée) $\lim_{\varepsilon\to 0}\frac{f(x+\varepsilon)-f(x)}{\varepsilon}$ n'existe pas

Valeurs spéciales

exposant	pseudo-mantisse	valeur
0	0	+/- 0
255	0	+/- infini, p.ex. x / 0
255	différent de 0	NaN (Not a Number), p.ex. 0 / 0

Exemple

Si l'on essaye de soustraire un petit nombre d'un nombre très grand, il y aura une annulation accidentelle, qui peut amener à des faux résultats.
Considérons une équation de seconde dégrée: $ax^2 + bx + c = 0$ avec $b \gg a, c$.
Les solutions sont données par

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

mais $\sqrt{b^2 - 4ac} \approx |b|$ si $b^2 \gg 4ac$ et

$$x_1 \approx 0 \qquad x_2 \approx -\frac{b}{a}$$

(il s'agit d'une annulation accidentelle)

Par contre, si on développe le discriminant en série de Taylor on trouve que

$$x_1 \approx -\frac{c}{b}$$

On peut réécrire la solution pour x_1 comme

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \cdot \frac{-b - \sqrt{b^2 - 4ac}}{-b - \sqrt{b^2 - 4ac}} = \frac{2c}{-b - \sqrt{b^2 - 4ac}} \approx -\frac{c}{b}$$

et une des deux annulations accidentelles disparaisse.

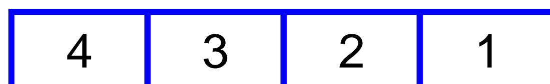
voir Racines.cpp (essayez avec a = 1, b = 1'000'000'000, c = 1)

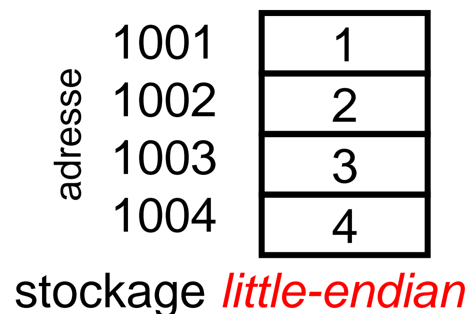
En générale il faut analyser les expressions que on utilise et les écrire d'un façon tel d'éviter la soustraction (ou addition) entre nombres trop différentes.

Mémorisation d'un nombre

Les variables d'un programme sont mises en mémoire sans indication de leur type. Cela peut amener à des résultat bizarres si le programmeur se trompe de type (voir p.ex. la division entre entiers) ou s'il essaie de stocker une valeur numérique plus grande que le maximum autorisé par le type de la variable (p.ex. débordement).

On peut stocker des valeurs numériques en mémoire qui occupent plusieurs octets commençant par l'octet de poids fort suivi des autres octets dans le cases suivantes ou par l'octet de pois faible.





stockage *big-endian*

stockage *little-endian*

Le deux solutions existent et chacune a été implémentée par certains processeurs. Les processeurs fabriqués par IBM et Motorola stockent les octets de poids fort d'abord et sont appelés gros-boutistes (*big-endian*); les processeurs fabriqués par Intel stockent les octets de poids faible d'abord et sont appelés petit-boutistes (*little-endian*). Chaque processeur est cohérent dans son travail et les opérations arithmétiques s'effectuent correctement en commençant toujours par les octets de poids faible quel que soit l'ordre de stockage. Le problème se pose lorsque l'on essaie de transférer des données d'un ordinateur à un autre.

Interface graphique DISLIN

voir

http://www.mps.mpg.de/dislin

ou

http://dpnc.unige.ch/~bravar/C++2014/compilateur/DISLIN

Interface graphique

Pour dessiner des graphes de fonctions, nous utiliserons le logiciel DISLIN
http://www.mps.mpg.de/dislin/ (v10.4 32-bit, parce que Dev-C++ travaille avec 32 bits !)

Il s'agit d'un ensemble de fonctions (logiciel) que vous pouvez appeler dans votre programme C++ pour dessiner des courbes et surfaces, faire des histogrammes, etc. … Comme pour la majorité des logiciels, vous n'avez pas besoin de connaître les détails de fonctionnement des fonctions de DISLIN, mais seulement savoir comment appeler ces fonctions (i.e. quels sont les arguments et comment les passer aux fonctions) et quel sera le résultat produit par ces fonctions.

Le logiciel DISLIN est composé de

1. le fichier en-tête `dislin.h` qui contient les prototypes (déclarations) des fonctions
2. la bibliothèque `dismgc_d.a` qui contient les fonctions graphiques déjà compilées

Cette bibliothèque est lisible seulement par l'ordinateur et, comme pour la majorité des logiciels, le *code source* n'est pas disponible.
Le logiciel contient plusieurs fonctions graphiques. On en verra quelque exemple.
Cela sera suffisant pour la suite.
L'utilisation du logiciel DISLIN n'est pas très différent de l'utilisation p.ex. de fonctions mathématiques définis dans la bibliothèque `cmath`.

Pour utiliser DISLIN, on doit dire au compilateur où aller chercher ces fichiers (avec Dev-C++, il faut modifier certaines options du compilateur).

Installation de DISLIN (Windows)

Téléchargez DISLIN depuis http://www.mps.mpg.de/dislin (pour windows) (v10.4 32-bit !) choisissez le fichier dl_10_mg.zip et suivez les instructions d'installation :

1) Décompressez le fichier dl_10_mg.zip

2) Lancez l'installation : cliquez deux fois sur l'icone setup.exe dans le dossierdl_10_mg

3) Installez le logiciel dans le répertoire C:\dislin.

4) Maintenant il faut dire au compilateur Dev-CPP où sont les fichier
   `dismg_d.a` (bibliothèque) et `dislin.h` (fichier en tête).

Dans Outils (Dev-CPP), sélectionnez Options du Compilateur puis Compilateur et ajoutez la ligne suivante au compilateur

```
C:\dislin\dismg_d.a -luser32 -lgdi32 -lopengl32
```

Toujours dans Options du Compilateur sélectionnez Répertoires puis Répertoires C++ et ajoutez la ligne suivante

```
C:\dislin\real64
```

Par contre, depuis une console DOS, vous pouvez instruire directement le compilateur où aller chercher ces fichier (essayez !) :

g++ test.cpp –o test C:\dislin\dismg_d.a -luser32 -lgdi32 -lopengl32 –I C:\dislin\real64

Exemple 1

Ce programme dessine une ligne qui connecte un ensemble de points (5).

```
#include <iostream>
#include "dislin.h"

int main() {
  const int nPoints = 5;
  double x[nPoints] = {1.,2.,3.,4.,5.};
  double y[nPoints] = {1.,3.,7.,9.,13.};

  metafl("XWIN");
  qplot(x,y,nPoints);

  metafl("PDF");
  qplot(x,y,nPoints);

  return 0;
}
```

fichier en-tête dislin.h

nombre de points

tableau de coordonnées x

tableau de coordonnées y

affichage sur l'écran

graphique de x vs y
on appel la fonction `qplot` pour dessiner la ligne qui connecte chaque point x-y

sortie dans un fichier PDF

voir Exemple0.cpp

Philosophie de DISLIN

Pour dessiner la fonction f(x)

1. tabuler la fonction dans deux tableaux :
   - coordonnées x
   - coordonnées y : valeurs de la fonction f dans les points x (i.e. f(x))

2. passer les tableaux à des fonctions de DISLIN pour dessiner le graphique

Pour dessiner la fonction f(x, y)

1. tabuler la fonction dans un tableau bidimensionnel :
   - coordonnées x = premier indice
   - coordonnées y = deuxième indice
   - coordonnées z : valeurs de la fonction f dans les points x et y (i.e. f(x, y))

2a. définir le graphique
2b. passer le tableau à des fonctions de DISLIN pour dessiner le graphique

Référence

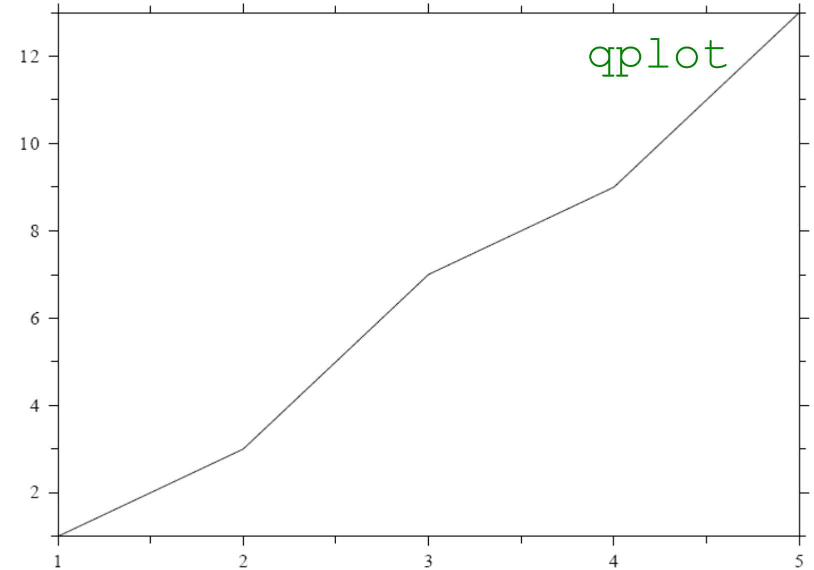
H. Michael

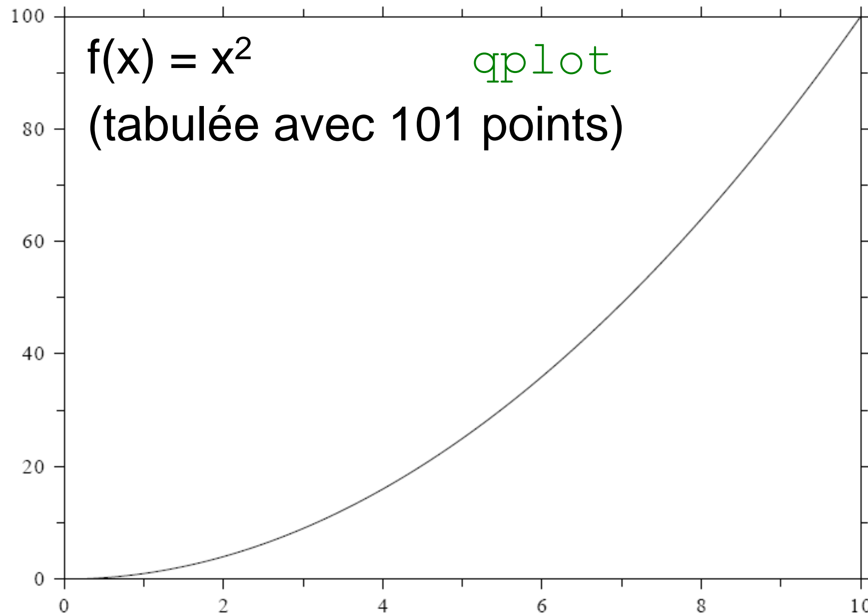
The Data Plotting Software DISLIN v. 10

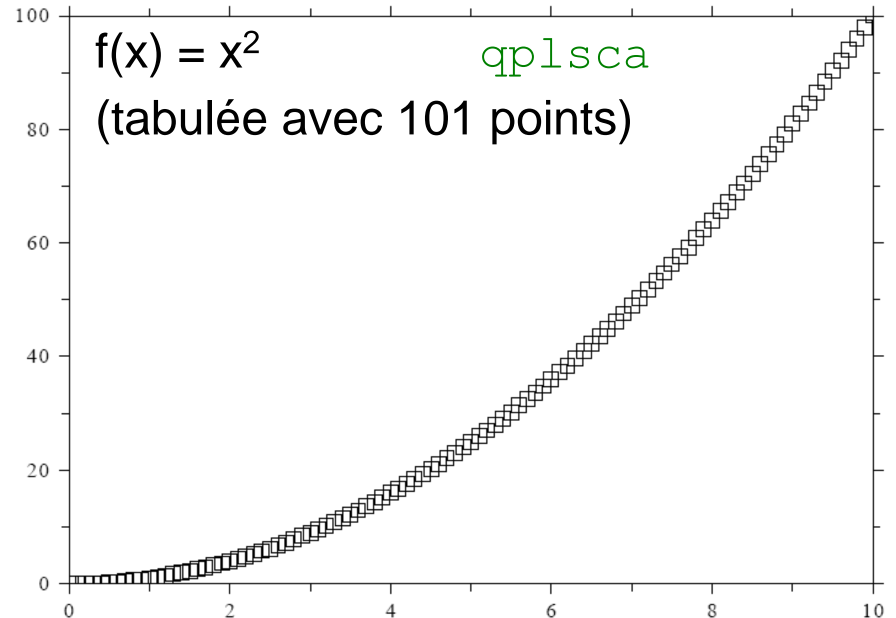
ou http://dpnc.unige.ch/~bravar/C++2014/DISLIN/dislin-9.5.pdf

Dessins crées avec DISLIN
utilisant des fonctions graphiques différentes

Exemple0.cpp
Exemple1.cpp
Exemple1b.cpp







Analyse du programme

1. Avant d'utiliser n'importe quelle fonction, il faut la déclarer (comme pour les variables). Les fonctions de DISLIN sont déclarées dans le fichier en-tête `dislin.h`, donc avant l'utilisation de ces fonctions, il faut inclure ce fichier dans le programme :

```
#include "dislin.h"
```

cf. `#include <iostream>` : pour inclure des fichiers en-tête du système, on utilise les symboles < et >, pour inclure des fichiers créés par l'utilisateur, on utilise les `" "`, et le fichier termine par `.h` .

2. Dessin d'un graphique : `qplot (x,y,n)` connecte les `n` points de coordonnées (`x`, `y`) par une ligne ; les coordonnées des points sont enregistrées dans les tableaux `x` et `y` de dimension `n`. Avant l'appel de la fonction `qplot` il faut remplir les tableaux `x` et `y`.

prototype de la fonction `qplot` : `void qplot(double[], double[], int);`

3. Par défaut, le programme dessine le graphique sur l'écran. On peut rediriger le graphique dans un fichier de type PDF en utilisant la fonction `metafl("PDF")`.

4. Pour *dessiner* une fonction *f(x)*, il faut d'abord *tabuler* la fonction, i.e. remplir un tableau avec les valeurs de x_i et un tableau avec les valeurs de la fonction $f(x_i)$;

p.ex.:

```
for (int i=0; i<n; i++) {
   x[i]=i;
   y[i]=f(i); }
```

voir Exemple1.cpp et Exemple1b.cpp

Exemple 2

Ce programme (un peu plus compliqué) dessine deux fonctions sur le même *graphe*.

initialisation

```
metafl("XWIN");
disini();
name("axe X","X");
name("axe Y","Y");
ticks(4,"X");
ticks(2,"Y");
titlin("Dessin de f(x)",1);
titlin("f(x) = x**2",3);
titlin("f(x) = x**3/2",4)
```

affichage sur l'écran
initialisation de DISLIN
étiquettes des axes X et Y
segmentation des axes X et Y
titre

système de coordonnées

```
graf(x[0],x[PTS-1],x[0],x[PTS-1]/10.,y1[0],y1[PTS-1],y1[0],y1[PTS-1]/10.);
title();
```

```
thkcrv(10);
color("RED");
curve(x,y1,PTS);
color("GREEN");
curve(x,y2,PTS);
color("FORE");
grid(1,1);
```

couleur du graphe
premier graphe (on a déjà rempli le vecteurs x et y)
couleur du deuxième graphe
deuxième graphe (même tableau)
grille

```
disfin();
```

fermeture de DISLIN

voir Exemple2.cpp et Exemple2b.cpp

Exemple 3

Graphique tridimensionnel :
ce programme dessine une fonction bidimensionnelle

initialisation

```
metafl("PDF");
setfil("fxy.pdf");
filmod("VERSION");
disini();
name("axe-X","X");
name("axe-Y","Y");
name("axe-Z","Z");
titlin("f(x,y) = 2 * sin(x) * cos(y)",4);
```

impression dans un fichier PDF
nom du fichier
initialisation de DISLIN
étiquettes des axes X, Y et Z
titre

```
graf3d(0.,360.,0.,90.,0.,360.,0.,90.,-3.,3.,-3.,1.);
title();
```

système de coordonnées

```
color("GREEN");
surmat(&z[0][0],PTSX,PTSY,1,1);
```

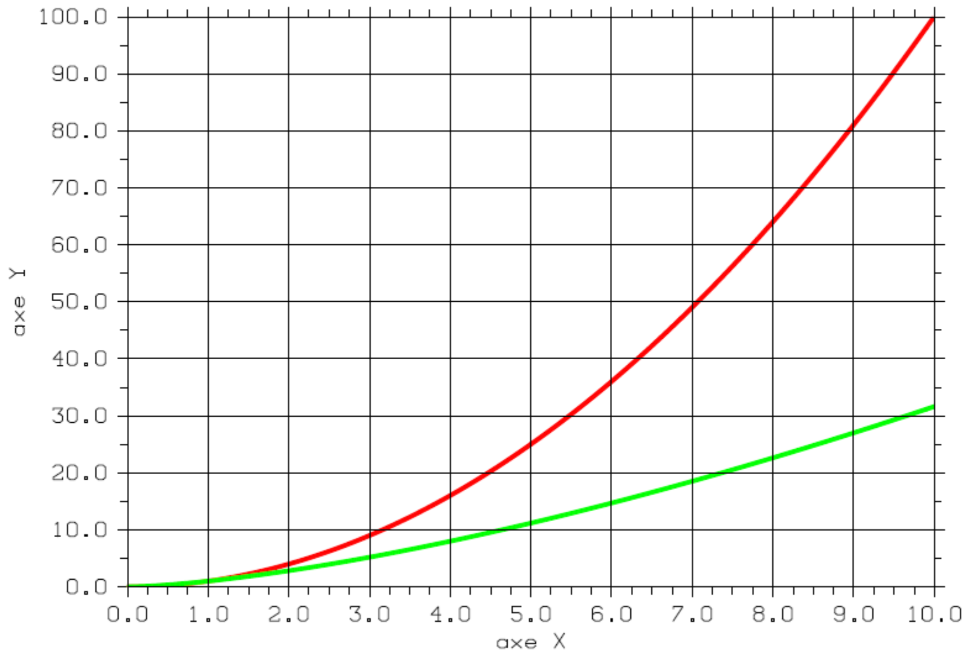
couleur du graphe

surface `z = f(x,y)`
la fonction est tabulée dans la matrice `z`
la matrice z est passé à la fonction `surmat`
par pointeur : `&z[0][0]` est l'adresse –
mémoire du premier élément du tableau `z`

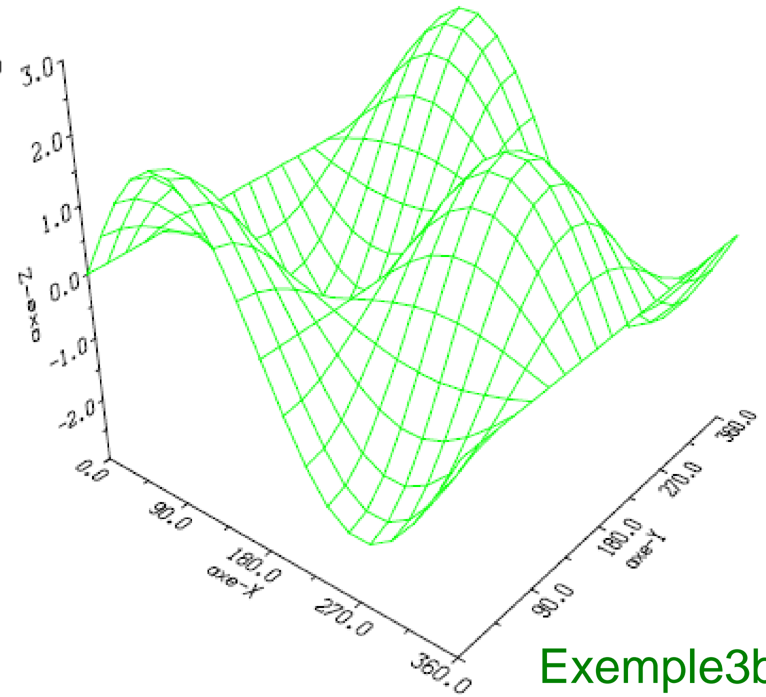
```
disfin();
```

fermeture de DISLIN

voir Exemple3.cpp et Exemple3b.cpp



Exemple2b.cpp



Exemple3b.cpp

Analyse des programmes 2 et 3

1. Pour utiliser des fonctions graphiques plus avancées, il faut d'abord initialiser DISLIN avec la fonction `disini()` (prototype : `void disini();`).
En même temps, on dirige le graphique vers l'écran avec la fonction `metafl("XWIN")`.

2. Les dimensions du graphe sont définies avec la fonction `graf`.

```
graf(xmin, xmax, xorig, xstep, ymin, ymax, yorig, ystep)
```

xmin, xmax – limites inférieures et supérieures de l'axe des x (horizontal)
xorig – premier *marqueur*
xstep – distance entre les *marqueurs*

3. La fonction `curve(x,y,n)` connecte les `n` points de coordonnées (`x`, `y`) par une ligne; avec cette fonction on peut dessiner plusieurs courbes sur le même graphique.

4. On peut aussi choisir la couleur de la courbe avec la fonction `color`.

5. Pour terminer DISLIN (fermer tous les fichiers ouverts), on appelle la fonction `disfin`.

6. (ex. 3) Une fonction bidimensionnelle (surface) peut être dessinée avec la fonction `qplsur`. D'abord il faut tabuler la fonction *f(x,y)* : on remplit un tableau bidimensionnel avec la valeur de la fonction à chaque point (x,y). En réalité (x,y) sont des indices de points. Le tableau est passé à la fonction par pointeur.

prototype de la fonction `qplsur` : `void qplsur(double*, int, int);`

Quelques fonctions de DISLIN

dislin.h — fichier en-tête, contient les prototypes des fonctions
(ouvrez le fichier dislin.h pour voir les prototypes)

dismgc_d.a — librairie de dislin en précision `double`, contient les fonctions de dislin déjà compilées, utilisée pendant l'édition de lien (2ème phase de la compilation) pour la création de l'exécutable
lisible seulement par l'ordinateur, le code source n'est pas disponible

disini() — fonction d'initialisation de DISLIN
elle doit être appelée avant l'utilisation d'autres fonctions de dlislin

disfin() — dernière fonction pour terminer DISLIN

qplot(x, y, n) — cette fonction dessine une ligne entre les points (x, y);
x et y sont deux tableaux de cordonnées (x, y), n le nombre de points

```
const int n = .. ; double x[n], y[n];
```

qplsca(x, y, n) — cette fonction dessine des marqueurs aux points (x, y)

curve(x, y, n) — cette fonction dessine une courbe à travers les points (x, y)

graf(……) — génération d'un système de coordonnées (voir exemples 2 et 3)

endgrf() — fin du graphique

qplsur(ptrf,nx,ny) cette fonction dessine une fonction bidimensionnelle (surface), les valeurs de la fonction f = f(x,y) sont enregistrées dans un tableau bidimensionnel; le tableau est passé à la fonction par pointeur

```
const int nx .. = , ny = ..;
double f[nx][ny];
qplsur(&f[0][0],nx,ny);
```

qplclr(ptrf,nx,ny) cette fonction dessine les valeurs d'une fonction bidimensionnelle aux points (x, y) (voir qplsca)

surmat(ptrf,nx,ny,1,1) la fonction bidimensionnelle est tabulée sous forme de matrice

metafl("PDF") pour enregistrer le graphique dans un fichier PDF (autres possibilités: XWIN → écran windows, POST → postscript, etc.)

color("RED") change la couleur de la courbe (RED, BLACK, GREEN, YELLOW, WHITE, BLUE, MAGENTA, etc.)

name(…) ajoute des *étiquettes* aux axes

titlin(…) ajoute le titre au graphique

titile() dessine le système des coordonnées après graf

setfil(…) pour choisir le nom du fichier du graphique

Pour plus d'informations, consultez le manuel de DISLIN : dislin-9.5.pdf.

En-tête `dislin.h`

Pour passer les arguments aux fonctions de DISLIN il faut connaître les prototypes des fonctions DISLIN définis dans le fichier en-tête `dislin.h`.

Le fichier en-tête `dislin.h` est situé dans le dossier `dislin\real64`.

extrait du fichier `dislin.h` :

```
. . .
void qplot  (const double *xray, const double *yray, int n);
void qplsca (const double *xray, const double *yray, int n);
void qplpie (const double *xray, int n);
void qplbar (const double *yray, int n);
void qplclr (const double *zmat, int n, int m);
void qplsur (const double *zmat, int n, int m);
void qplcon (const double *zmat, int n, int m, int nlv);
void quad3d (double xm, double ym, double zm,
             double xl, double yl, double zl);
int  rbfpng (char *cbuf, int nmax);
void rbmp   (const char *cfil);
int  readfl (int nu, unsigned char *cbuf, int nbyte);
void reawgt (void);
void recfll (int nx, int ny, int nw, int nh, int ncol);
void rectan (int nx, int ny, int nw, int nh);
void rel3pt (double x, double y, double z, double *xp, double *yp);
void resatt (void);
void reset  (const char *cname);
. . .
```

Résumé

Représentations des nombres

Base binaire

Conversion entre base binaire et décimale

Arithmétique binaire

Représentations des nombres à virgule flottante

Précision des nombres à virgule flottante

Interface graphique DISLIN

Exercices – série 7

Exercices

1. Exprimez les nombres décimaux 94, 141, 163 et 191 en base 2, 8 et 16.

2. Donnez sur 8 bits les représentations « signe et valeur absolue » et complément à 2 des nombres décimaux -47, -99, -102 et -118.

3. Démontrez les propriétés du complément à 2 :

$$X + C_2(X) = 2^n$$
$$C_2(C_2(X)) = X$$

4. Considérez le cas d'une variable **`short`**. Quel est le plus grand entier négatif supporté par ce type de variables ? Ecrivez un programme qui décrémente à chaque itération la variable **`short`** `m`, initialisée à 0, de 10'000. Imprimez `m` à chaque étape.
Que se passe-t-il ? Comment interprétez-vous les résultats ?

5. Les erreurs d'arrondi viennent du fait que les variables **`float`** (ou **`double`**) ne sont que des approximations de nombres réels. Soit `a` une **`float`** = 333333. Ecrivez un programme déclarant les variables b = 1 - 1/a, c = 1/b - 1 et d = 1/c + 1. Quelle est la valeur algébrique de d ? Que retourne votre programme ? Pourquoi ?

6. Considérez la suite numérique : $U_{n+1} = -\frac{3}{8}U_n^2 + \frac{9}{4}U_n - \frac{3}{8}$

Montrez que si U_0 = 3 ou si U_0 = 1/3 la suite est constante. Ecrivez un programme qui affiche les 100 premières itérations. Comment interprétez-vous les résultats ?
Refaites l'exercice avec la suite : $U_{n+1} = -\frac{4}{9}U_n^2 + \frac{26}{9}U_n - \frac{4}{9}$

(attention : le résultat peut être différent sur Windows et le Mac)

Exercices DISLIN

1. Dessinez la fonction $f(x) = 1 / (1 + x^2)$ entre 1 et 100 avec 100 points.
Au lieu de `qplot`, essayez `qplsca`; à chaque coordonnée, un symbole sera dessiné.

2. Dessinez sin(x) et cos(x) sur le même graphe !
(voir Exemple2b.cpp)

3. Dessinez la fonction $f(x,y) = x^2 + y^2$ entre -10 et 10 avec 21 x 21 points.

Deuxième Control Continu

23 Avril 2015
10h15 – 12h15

Salle 202 Science I

Amenez vos portables !

Vous pouvez utiliser toutes les notes du cours (incl. les corrigées 2015), des textes C++, …

interdit : e-mail, téléphone, Skype, des recherches sur la toile, facebook …