Informatique - Concepts Généraux

laurent.thiry@uha.fr

ENSISA - 12, rue des frères Lumière - 68093 Mulhouse (France)

Introduction

Objectifs

- Présenter les principaux concepts liés au domaine de "l'informatique"
  - Qu'est-ce que'un programme ? un compilateur ? un type ? etc.
- Apprendre les bases de la programmation (en langage C)
  - Tremplin vers d'autres langages: `C++`, `C#`, ou `Java`
  - Et d'autres matières: `structures de donnees`, `UML/SysML`, etc.

Introduction

Historiquement

- L'informatique est naît dans les années 1930 avec
  - Alan Turing, les automates (à ruban) et les langages impératifs
  - Alonzo Church, le lambda calcul et les langages fonctionnels
- C'est une branche des mathématiques qui s'intéresse ...
  - Au "calcul" (: trajectoire de missiles)
  - Au "raisonnement" (: modèle d'intelligence ?)

Introduction

Informatique

Science et techniques du traitement de l'*infor*mation par une machine auto*matique*

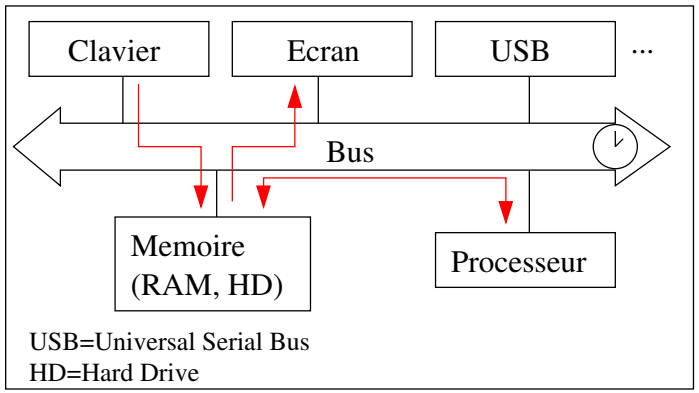
En anglais: computer science (science du calcul)
PC = Personal Computer

Information

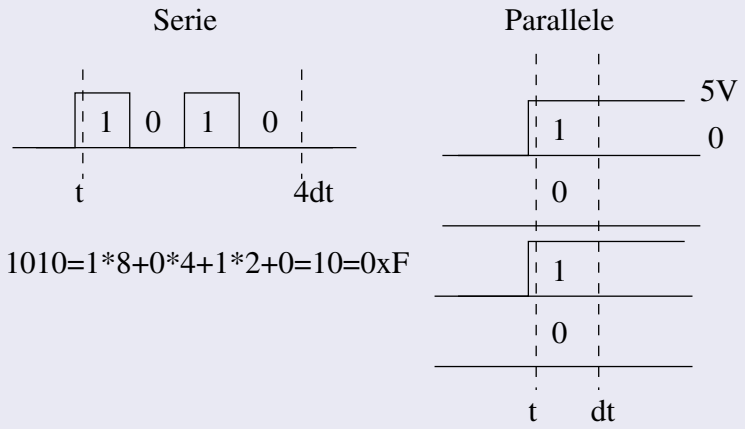
Elément de connaissance pouvant être codé, stocké, échangé ou transformé

Architecture d'une "machine"

- Un bus de communication
- Des périphériques (clavier, écran, disque dur, etc.)
- Une mémoire (volatile RAM et/ou persistente ROM)
- Un processeur/calculateur



Rmq. Différence Série/Parallèle (:USB)



Mémoire

Suite de cellules 0/1 appelées "bits" (:binary digits)
La position d'un élément est appelée "adresse"

- 4 bits = 1 code hexadecimal (0..9,A..F)
- 8 bits = 1 octet (= 256 combinaisons)
- 1 Ko=1000 octets, 1 Mo=1000Ko, 1 Go=1000Mo, 1 To=...
- 1 caractère peut être représenté par un octet (:code ascii)
  'a' = 97 = 61 = 0110 0001
- 4o (:32bits) ou 8o (=64bits) peuvent représenter des nombres
  Q. Quelle taille est nécessaire pour stocker une image en 256 niveaux de gris 1024*768 ?

Quelques mesures

- Image en niveau de gris = ?
  786Ko
- Image couleur RGB (Red-Green-Blue) = ?
  786Ko*3 = 2.3Mo
- Film de 60 secondes et 25 fps (Frame per Second) = ?
  2.3Mo*60*25 = 3.4Go

Code ascii

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

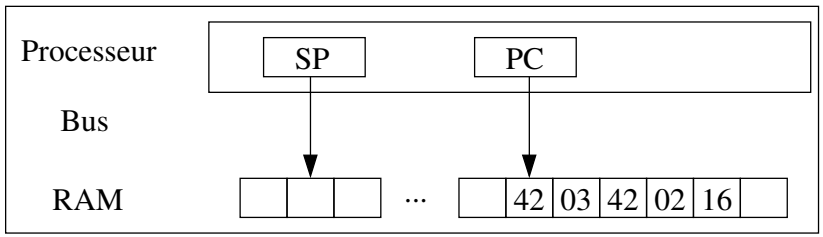
Source: www.LookupTables.com

Processeur

Circuit logique séquentiel associé à un jeu d'instructions pour lire/écrire en mémoire, réaliser des opérations arithmétiques ou logiques, etc.

En général, 2 registres sont utilisés pour

1. Les données manipulées (ex. Stack Pointer SP)
2. Les instructions à exécuter (ex. Program Counter PC)



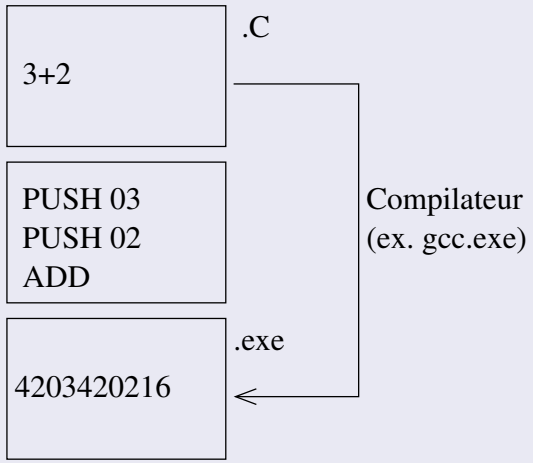
Exemple d'interprétation

- "42 x" = Push x (in the stack)
  `M[SP]=x; SP=SP+1; PC=PC+2;`
- "16" = Add
  `M[SP-2]=M[SP-1]+M[SP]; SP=SP-2; PC=PC+1;`

- Q. Ré-Exprimer le programme en utilisant Push/Add, et montrer l'évolution de la pile
- Le langage précédent est appelé "assembleur" et est fourni avec la doc technique du processeur

Principe

- Il existe des lanngages de plus haut niveau (ex. C) et des "compilateurs" qui transforment un programme compréhensible C en fichier exécutable (420342...)



Introduction

En général, un programme se compose

- De variables pour stocker une information (sans se soucier de son adresse mémoire)
  Ex. nom=”John”, age=23, notes=[12,15,14], etc.
- D'opérations, appelées sous-programmes ou fonctions, pour réaliser une fonctionnalité particulière
  Ceci, en utilisant les variables et des instructions de base (:tests ou boucles) ou en utilisant d'autres sous-programmes
  Ex. anniversaire, moyenne, ajouterNote, etc.

Les variables

- Elles sont définies par un nom (:lettre suivie d'un nombre quelconque de lettres ou de chiffres)
- Elles ont un "type" pour connaître la taille mémoire nécessaire pour stocker leurs valeurs
- Il existe des types de base (int, char, float) et des types complexes (structures et tableaux)
- Un programme peut aussi définir de nouveaux types - ex. programmation orientée objets

Q. Quelle est le type des variables précédentes ?
Quelle est la taille mémoire utilisée ?

Instructions usuelles

Les constructions liées aux variables sont

- La définition/déclaration
  Ex. `char nom[20]; int age;`
- L'affectation
  Ex. age=23; nom[1]='a';
- L'acces
  Ex. age+1;

Instructions usuelles

Formellement

- La déclaration

```
<Decl> ::= <Type> <Nom> [<Int>]*;
<Type> :: int | char
```

- L'affectation

```
<Aff>  ::= <Nom> = <Expr>;
| <Nom>[<Expr>] = <Expr>;
<Expr> ::= <Int> | <Expr> + <Expr> | ...
```

- L'acces

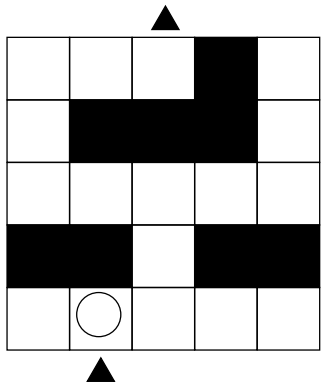
```
<Expr> ::= <Nom> | <Nom>[<Expr>]
```

Remarques

- Les règles précédentes définissent la "grammaire" du langage
- Ces règles génèrent un ensemble de termes, et un terme est correct s'il appartient à cet ensemble
- Sinon c'est une erreur de syntaxe
  Ex. `int x`
  `char t[][];`
- A partir de là, toute variable doit être: 1) déclarée/définie et 2) affectée à une valeur avant d'être utilisée

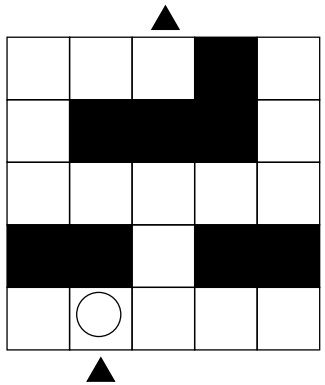
Exemple

Proposer un ensemble de variables pour réaliser le jeu du labyrinthe
Quel sont les sous-programmes possibles ?



Exemple

Une solution parmi n



```
int x, y;    // ou int p[2];
x=2; y=5;
int z[5][5]; // ou int z[25];
z[0][0]=0; z[1][4]=1; ...

droite(); haut(); haut(); gauche(); ...
```

Le labyrinthe

- Proposer une réalisation de haut, gauche, droite, etc.
- Est-ce que le code suivant peut représenter le labyrinthe ? Expliquer.
  `lab = [3,1,2,3,6,2,1,2,5];`
- Meme question avec `lab = 0x1381B0;`
- Quelle est la "meilleure" représentation du labyrinthe ?

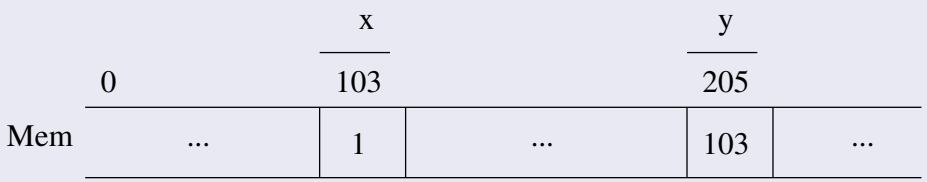
Instructions usuelles

- L'opérateur & retourne l'adresse d'une variable
- Le type "adresse" est appelé "pointeur" et est noté *
  Ex. `int* x;` est l'adresse d'un entier
- L'opérateur * est utilisé pour accéder à la valeur associée à une adresse ou pour la modifier
- Quelle est la valeur de x après le programme suivant ?

```
int x; x=1;
int* y; y=&x;
*y = *y+1;
```

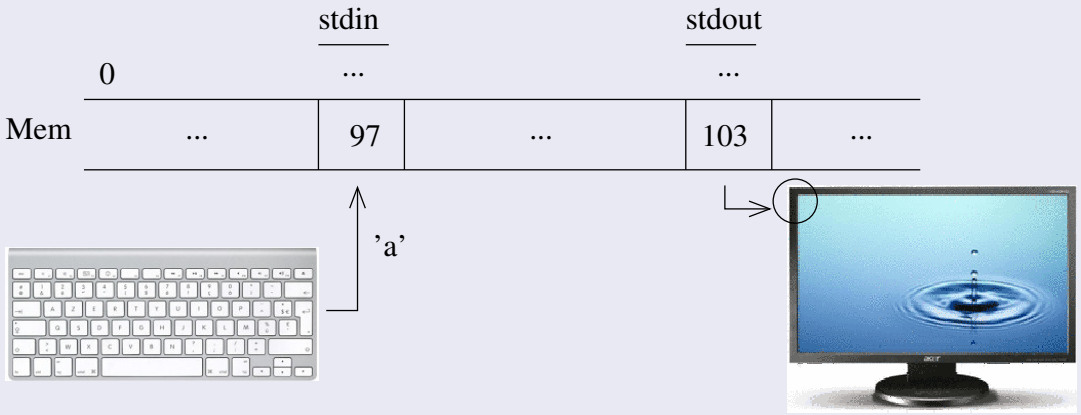
- Comment écrire à l'écran ou lire un caractère sur le clavier en utilisant les éléments précédents (FILE* stdin/stdout)

Graphiquement



&x est l'adresse de la variable x (=103)
*y est la valeur de l'adresse y (=*103=1)

Pour les IO



Instructions génériques

Il existe 3 instructions fondamentales

- Les tests
  `<Ins> ::= if(<Expr>) { <Ins>* } else { <Ins>* }`
  Ex. `if (age<0) {age=-age;} else {}`
  La partie "else" est omise si son bloc est vide
  Les accolades sont omises s'il n'y a qu'une instruction
- Quelle est la nature de l'expression entre parenthèses ?
- Comment représenter graphiquement un test ?
- Donner un code équivalent à:
  `if (a&&b) { X } else { Y }`

Instructions génériques

- Les boucles finies
  `<Ins> ::= for(<Aff>,<Exp>,<Aff>) { <Ins>* }`
  Ex. `int i,r; for(i=1;i<10;i=i+1) {r=r*i;}`
- Ré-exprimer le programme pécédent sans "for"
- Les boucles indéfinies
  Ex. `while (i<10) {r=r*i;i=i+1;}`
  Quelle est la syntaxe du while ?

Exercice

Soit le tableau de notes ci-dessous.

- Donner les variables nécessaires pour stocker le tableau
- Proposer un programme qui calcule la moyenne m
- Proposer un autre programme qui retourne le nom n du meilleur étudiant

John	15
Paul	18
Jane	12

Comment réutiliser les programmes "moyenne" et "meilleur" ? (Source)

Les sous-programmes

- Il est possible de donner un nom à un bloc d'instruction
- Si un bloc calcule une valeur alors le type de celle-ci est le type du bloc
- La valeur renvoyée est donnée par `return <Expr>;`
- Le type `void` est utilisé pour montrer qu'il n'y a pas de valeur retournée
  Ex. `void anniversaire() { age=age+1; }`
  - "age" est appelée variable globale
  - Les variables déclarées dans le bloc sont appelées variables locales ; elles n'existent qu'entre les accolades
- L'exécution/appel d'un sous programme consiste à écrire son nom
  Ex. `anniversaire();`

Les sous-programmes

- Il est possible d'ajouter des paramètres à un sous programme
  Ex.
```
int plus(int x, int y) { return x+y; }
int abs(int n) {
 if (n<0) { return -n; } else { return n;} }
```
- Le programme est appelé alors avec des valeurs de paramètre
  Ex.
```
int x; x=plus(2,abs(-2));
```
- Les paramètres sont passés par valeur !
  Ex. Que vaut y dans le sous programme suivant ?

```
int x,y; x=1;
void f(int n) { n=0; }
f(x);
y = x;
```

Les sous programmes

- Pour résoudre le problème précédent, on utilise des pointeurs
- On parle de passage par référence
- Exemple

```
int x,y; x=1;
void f(int* n) { *n=0; }
f(&x);
y = x; // vaut 0
```

- Ex. Proposer un programme qui échange 2 valeurs `swap(a,b)`

Les sous programmes

Exercices. Définir (et représenter graphiquement):

- Une fonction retournant 1 si un nombre $x \in [a, b]$, 0 sinon
- Une fonction calculant la puissance x^n
- Une fonction retournant la valeur max d'un tableau t de taille n
- Une fonction qui calcule le montant d'un compte bancaire pour l'année n, avec un montant initial de 100 euros et 5 pour cent d'intérêts
  Note. Le type `float` représente les nombres réels (à virgule "flottante")

Les sous programmes

- Tout programme doit avoir un sous programme principal
  `void main() { ... }`
- Il existe des bibliothèques de variables/fonctions prédéfinies
  Ex. La bibliothèque standard inputs-outputs (stdio) utilisable en ajoutant en en-tête
  `#include <stdio.h>`
- Pour afficher un entier à l'écran
  `printf("%i",12);`
- Pour lire un entier au clavier
  `int c; scanf("%i",&c);`

La fonction "printf"

- `printf("...");` affiche à l'acran les pointillés
- `printf("%s",s);` affiche la chaîne de caractères "s"
- `printf("%i",i);` affiche l'entier "i" (idem pour un char %c ou un float %f)
- Il existe des caractères spéciaux: `\n` représente un retour à la ligne, `\t` une tabulation, etc.

La fonction "scanf"

- `scanf("%s",s);` lit une chaîne de caractères et place la valeur dans "s"
- `scanf("%i",&i);` lit un entier dans "i" (idem pour un char %c ou un float %f)

Remarques/Questions

- Que fait le programme suivant ?

```
char s[20];
//s = "comment va ?";
strcpy(s,"comment va ? ");
printf("%s",s);
scanf("%s",s);
printf("%s",s);
```

- Que peux faire la fonction "strcpy" (STRing CoPY)?
- D'où vient-elle ?
- Comment afficher chaque caractère de "s" sur une ligne ?

Exercices

- Proposer un programme pour afficher le labyrinthe vu précédemment
- Proposer un programme qui lit une touche au clavier, déplace le héros, et affiche le labyrinthe tant que l'on n'est pas arrivé à la sortie
- Démonstration ! (Source)
- Réaliser un programme qui simule un processeur (voir modèle SP/PC)
- Voir aussi TD1

Types et Instructions fondamentales

Introduction

- Il existe des types primitifs (entiers, caractères, réels, et pointeurs) et des types complexes (tableaux, structures)
- Le typage permet d'éviter des erreurs
  Ex. 12+"vingt trois" est égal à ? abs("un") == ?
- Chaque type est associé à un ensemble d'opérateurs
  Ex. Opérateur de calcul ou de conversion

Les entiers (et booléens)

- Il représente les valeurs entières comprises entre +/- 2Go (sur 4o et 1bit de signe)
  Ou: 0 et 4Go dans le cas nom signé (`unsigned int x;`)
- Il est possible de réduire l'ensemble des valeurs avec `short int` soit 2o et des valeurs dans +/-32768
- Les opérateurs usuels sont les opérateurs arithmétiques (`+,-,*,etc.`) et logiques (`==,<,<=,etc.`)

Les (entiers et) booléens

- Les entiers servent à représenter les valeurs booléennes:
  0=faux, tout le reste=vrai
  Voir instructions `if (bool) ...` et `while (bool) ...`
- Les opérateurs logiques utilisables alors
  - la négation (!)
  - la conjonction (&&)
  - la disjonction (||)

Les entiers et booléens

Execrices. Proposer des fonctions permettant de

- Connaître le quotient et le reste de la division entière a/b
- Afficher la valeur numérique de a/b avec n chiffres après la virgule
- Calculer le pgcd de 2 nombres a et b
  `pgcd(a,b)=a` si `b=0`, `=pgcd(b,a%b)` sinon
  Quelle est la valeur de `pgcd(6,3)` ?
- Comment convertir un nombre hexa (: 4int) en décimal et réciproquement ?

Les caractères

- Les caractères sont codés sur 1o (: 256 valeurs possibles)
- Ils se représentent entre simple quotes, ex. `'a'` pour faire la distinction avec la variable `a`
- Le principal opérateur est la conversion `(int)'x'`
  Réciproquement, `(char)67` correspond à `'a'`
- Les opérateurs sur les entiers sont alors utilisables
- Un tableau de caractères est appelé aussi "string" (voir `string.h`)

Les caractères

Exercices.

- Comment convertir une minuscule en majuscule ?
- Comment convertir un texte en majuscules ?
- Soit un tableau t de 20 caractères. Comment crypter et décrypter ce tableau en prenant un caractère k comme clef ? Principe: `t[i]=(t[i]+k)%256`
- Comment améliorer le cryptage en utilisant une séquence de clefs ?
- Comment calculer le nombre d'utilisation d'une lettre 'c' ?
- Comment trouver la lettre la plus utilisée ?

Les pointeurs

- Un pointeur représente une adresse, et une position, en mémoire
- Si l'élément pointé est de type T alors un pointeur sur ce type d'élément est de type T*
- L'opérateur & retourne l'adresse d'un élément
- L'opérateur * retourne la valeur pointée
  `*(&x) == ?`

Les pointeurs

- La fonction `sizeof` retourne la taille mémoire nécessaire pour stocker une valeur de type T
  `sizeof(int) == ?` `sizeof(short int) == ?`
  `sizeof(char) == ?`
- La fonction `malloc(n)`[a] permet de réserver un emplacement mémoire ; elle retourne alors un pointeur vers celui-ci
- Il est nécessaire de spécifier le type pointé avec une convertion de type
- Ex. `int* p=(int*)malloc(sizeof(int));`
- Quel est l'intérêt de
  `int* p=(int*)malloc(10*sizeof(int));` ?

[a] Inclure la bibliothèque stdlib

Les pointeurs

- Dans l'expression précédente, p peut être interprété comme un tableau de taille 10
- L'acces à l'élément i est donné alors par `*(p+i)`
  Similaire à `p[i]`
- En particulier, `char*` est souvent utilisé pour représenter des chaînes de caractères (où strings[a])
- Une chaîne se représente entre double quotes "
  Ex. `char* t = "hello";`

[a] Voir aussi string.h

Les pointeurs

Exercices. Un signal est modélisé par une séquence d'entiers de taille n fixé[a]

- Proposer une fonction retournant le signal constant $s(i) = c$
- Idem pour une rampe $s(i) = c.i$
- Comment multiplier un signal par une constante k ?
- Comment faire la somme ou le produit de 2 signaux ?
- Comment calculer l'intégral $S(i) = s(i) + S(i-1)$?
- Comment obtenir la dérivée $d(i) = s(i) - s(i-1)$?
- Comment obtenir les optimum ? les valeurs dans [x-k,x+k] ?
- Comment représenter une séquence de signaux ?

[a] Une constante peut être définie de la façon suivante: const int n=10;

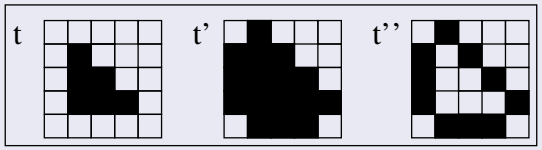
Les tableaux

- Ils permettent de manipuler plus facilement des pointeurs sur des collections
- Un tableau t de taille 15 et dont les éléments sont de type entier sera défini par `int t[15];`
  A la place de `int* t=(int*)malloc(15*sizeof(int))`
- Les indexes d'un tableau comment à 0, et l'accès au 3ème élément sera donné par `t[2]`
  A la place de `*(t+2)`
- Un tableau peut avoir plusieurs dimensions
  Par ex., une image 256x1024 pourra se représenter par `int i[256][1024];`

Les tableaux

Exercices. En prenant le modèle d'images ci dessous avec des valeurs 0/1:

- Comment inverser une images (: pour obtenir le négatif) ?
- Comment superposer deux images (en utilisant max) ?
- Comment faire un flou ?
  $t'[i][j] = max(t[i-1][j], t[i+1][j], t[i][j-1], t[i][j+1])$
- Comment faire une détection de contour ? $t'' = xor(t, t')$
- Comment obtenir le barycentre du contour ?
  La moyenne des positions pour les éléments à 1



Les flottants

- Les nombres réels sont représentés par le type "float"
  4o et interval +/-3.40282e+/-038
- Ou "double"
  8o et interval +/-1.79769e+/-308
- La représentation comporte deux entiers pour la valeur/mantisse et l'exposant[a]
- Les opérateurs possibles sont les mêmes que pour les entiers
- Et d'autres fonctions sont disponibles dans `math.h`

[a] Standard IEEE 754

Les flottants

Exercices.

- Soit un nombre a, la fonction $f(x) = (x - a/x)/2$. Proposer une fonction retournant la valeur de $u(n) = f(u(n-1))$ avec $u(0) = 0.1$
  Quel est la limite de u ?
- Soit une fonction f et un interval $[a, b]$. Proposer une fonction retournant la valeur x telle que $f(x) = 0$ (dichotomie)
- Soit un signal $s : \mathbb{N}_n \to \mathbb{R}$. Proposer une fonction pour normaliser s (: les valeurs $s(i)$ sont ramenées dans $[0, 1]$)

Les structures

- Une structure est définie par un nom est un ensemble de champs
- Un champ est défini par un nom et un type
- Ex. `struct Point { int x; int y; };`
- Une variable p de type Point est déclarée par `struct Point p;`
- Un exemple d'acces à un champ est `p.x = 12; printf("%i",p.x);`

Les structures

Exercice. Proposer un ensemble de fonctions pour:

- Afficher un point au format `(x,y)`
- Déplacer un point de `(dx,dy)`

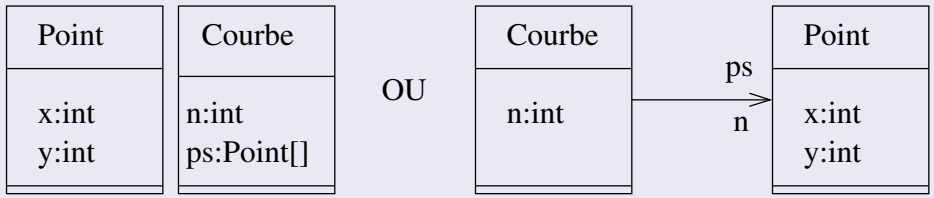
Une courbe est une liste de n points. Proposer des fonctions pour:

- Connaître le nombre de points d'une courbe ou ajouter un point à une courbe
- Afficher une courbe ou la déplacer une courbe de `(dx,dy)`;

Les structures

Remarques.

- Il existe une notation facilitant l'utilisation de pointeur de structures:
  `(*p).c` est équivalent à `p->c`
- Une structure est analogue à une "classe" en programmation objet
- Ainsi le dernier exemple peut se représenter



Les structures

Exemples

- Proposer un programme (et une bibliothèque de fonctions) pour manipuler des fractions (voir slide suivant)
  Même question mais pour les nombres complexes
- Comment représenter une adresse postale ? un gestionnaire de contacts (: pour trouver l'adresse d'une personne) ?

Exemple du type "Fraction"

```
struct Fraction {
  int num;
  int den; };
void affiche(struct Fraction f) {
  printf("%i/%i",f.num,f.den); }
void main() {
  struct Fraction f;
  f.num = 2;
  f.den = 8;
  affiche(f); }
```

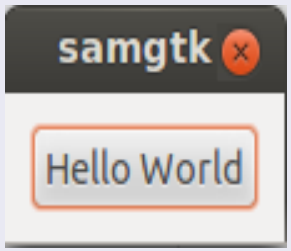
Exemple du type "Fraction" (2)

Compléter le code suivant ...

```
void main() {
  ...
  decimal(f);    // => 0.25
  simplifie(f); // => 1/4
  plus(f,f);     // => 1/2
  mult(f,f);     // => 1/4
```

Les structures

- La plupart des bibliothèques définissent des structures et des opérations sur celles-ci. Ex. d'interface graphique:




```c
#include <gtk/gtk.h>

static void hello(GtkWidget *widget, gpointer data) {
    g_print ("Hello World\n"); }

static gboolean delete_event(GtkWidget *widget, GdkEvent *event, gpointe
    g_print ("delete event occurred\n");
    return TRUE; }

static void destroy(GtkWidget *widget, gpointer data) {
    gtk_main_quit (); }

int main(int argc, char *argv[]) {
    GtkWidget *window;
    GtkWidget *button;

    gtk_init (&argc, &argv);
    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
    g_signal_connect (window, "delete-event", G_CALLBACK (delete_event),
    g_signal_connect (window, "destroy", G_CALLBACK (destroy), NULL);
    gtk_container_set_border_width (GTK_CONTAINER (window), 10);
    button = gtk_button_new_with_label ("Hello World");
    g_signal_connect (button, "clicked", G_CALLBACK (hello), NULL);
    g_signal_connect_swapped (button, "clicked", G_CALLBACK (gtk_widget_
    gtk_container_add (GTK_CONTAINER (window), button);
    gtk_widget_show (button);
    gtk_widget_show (window);
    gtk_main ();
    return 0; }
```

Les classes/objets

- Une évolution du langage c++ est la possibilité d'intégrer les fonctions aux structures (pour former une classe)
- Exemple.

```
struct Point {
  int x;
  int y;

  void affiche() { printf("(%i,%i)",x, y); }
  void deplace(int dx,int dy) { x=x+dx; y=y+dy; }
};
```

Les classes/objets

- Exemple d'utilisation.

```
int main() {
  struct Point p; p.x = 2; p.y = 3;
  p.affiche();
  p.deplace(1,2);
  p.affiche();

  return 1;
}
```

- p est appelé "instance de" Point, et affiche/déplace sont appelés "messages"

Les classes/objets

- Les structures sont alors remplacée par des "class"
- Avec une fonction particulière appelée "contructeur" pour initialiser les données membres

```
class Point {
  private:
    int x;
    int y;
  public:
    Point(int _x, int _y) { x=_x; y=_y; }
    void affiche() { printf("(%i,%i)",x, y); }
    void deplace(int dx,int dy) { x=x+dx; y=y+dy; }
};
```

Les classes/objets

- L'ensemble des opérations proposées (appelée "interface" .h) est alors séparée de la réalisation (.cpp)
- Seul la spécification est nécessaire pour pouvoir utiliser le type Point

```
class Point { // Point.h
  private:
    int x;
    int y;
  public:
    Point(int _x, int _y);
    void affiche();
    void deplace(int dx,int dy);
};
```

Les classes/objets

- Et le fichier cpp

```
#include "Point.h" // Point.cpp
Point::Point(int _x, int _y) {
  x=_x; y=_y;
}
void Point::affiche() {
  printf("(%i,%i)",x, y);
}
void Point::deplace(int dx,int dy) {
  x=x+dx; y=y+dy;
}
```

Les classes/objets

- Et pour finir, le Main

```
#include "Point.h"

int main() {
  Point p = Point(2,3);
  p.affiche();
  p.deplace(1,2);
  p.affiche();
  return 1;
}
```

- Pour la compilation

```
gcc -c *.cpp
gcc Point.o Main.o -o Main.exe
```

Les classes/objets

- Il existe un grand nombre d'objets réutilisables (ex. std::vector)

```
#include <vector>
int main() {
  std::vector<Point> courbe;
  courbe.push_back(Point(1,2));
  courbe.push_back(Point(2,3));
  for(int i=0;i<courbe.size();i++)
    courbe[i].affiche();

  return 1; }
```

Les classes/objets

Exercice. Proposer un programme pour gérer des transactions bancaires

- Une transaction à un compte émetteur, un compte récepteur, une date, un montant et un libellé
- On doit pouvoir ajouter une transaction, obtenir toutes les transactions à partir d'un compte x (et pour une période donnée)

Compilation séparée

Rappel

- Pour faciliter le développement d'applications, on sépare celles-ci en plusieurs fichiers .c (chacun définissant des types/fonctions pour un besion particulier)
- On sépare la réalisation des fonctions de leur spécification dans un fichier .h
- Seul ce dernier est nécessaire pour utiliser un type/fonction particulier
- Chaque fichier .c est compilé avec gcc -c pour obtenir les fichiers binaires .o
- Gcc avec l'ensemble des .o va réaliser le "linkage" entre les différents symboles, et générer un exécutable

Les fichiers

- Un fichier est représenté par le type `FILE*`
- La fonction `fopen(nom,acces)` retourne le pointeur sur le fichier "nom" avec un acces
  - "r" pour read (:lecture seule)
  - "w" pour write (:ecriture)
- La fonction `fclose(ptr)` permet de libérer l'acces à un fichier
- En considérant un buffer `char b[512]`
  - La fonction `fgets(buf,512,ptr)` lit une ligne et place les caractères dans "buf"
  - La fonction `fputs(buf,ptr)` écrit les caractères de "buf" dans le fichier (ptr)

Les fichiers

Exemple: Que fait le programme suivant ?

```
FILE* f;
FILE* f2;
f = fopen("files.txt", "r");
f2= fopen("files2.txt","w");
char buf[512];
while(fgets(buf,512,f) != NULL) {
  printf("%s",buf);
  fputs(buf,f2);
}
fclose(f);
fclose(f2);
```

Exercices

- En utilisant la fonction `strlen` (string.h), proposer un programme qui retourne la taille d'un fichier
  Note. Utiliser les paramètres de la ligne de commande:
  `void main(int argsn, char** args)`
- Proposer deux programmes pour crypter/decrypter un fichier avec la méthode de César

La récursivité

- Une fonction est "récursive" si elle est définie en s'utilisant elle-même.
  Ex. $fact(n+1) = (n+1) * fact(n)$
- Il faut noter que l'ensemble des entiers est défini récursivement par 0 et $n \in \mathbb{N} \Rightarrow (n+1) \in \mathbb{N}$
- Une fonction récursive est définie alors par: 1) un cas de base, ex. $fact(0) = 1$, et 2) un cas général comme ci-dessus.
- En C

```
int fact(int n) {
  if (n==0) { return 1; }
  else { return n*fact(n-1); } }
```

La récursivité

- La récursivité permet de réaliser autrement des boucles
- Par contre, certains algorithmes (sur les arbres par ex.) ne sont pas réalisables qu'avec la récursivité
- La récursivité est aussi liée au concept "d'induction" utilisée en preuve de programmes (cf. cours SD)
- Exercices
  Comment calculer la somme des valeurs d'un tableau t de taille n en utilisant une fonction récursive ?
  Comment obtenir le nombre de 'l' dans une chaine "hello ..." ?

La récursivité

- La récursivité s'applique aussi aux types. Par exemple, une liste est soit vide, soit obtenue en ajoutant un élément à une autre liste
- Ceci correspond à deux fonctions:

```
Liste vide();
Liste ajoute(int x,Liste y);
```

Comment définir la liste l=(1,(2,(3,()))) - soit plus simplement (1,2,3) ?

- Aux fonctions précédentes, on ajoute généralement les fonctions suivantes:

```
int   estVide(Liste l);
int   premier(Liste l);
Liste suivant(Liste l);
```

- Donner une réalisation possible du type "Liste"

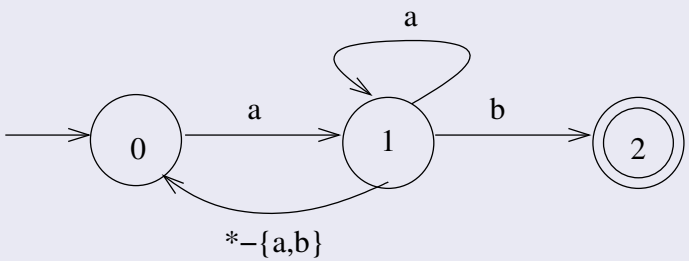
La récursivité

Exercices. Proposer des fonctions récursives pour:

- Calculer la longueur d'une liste
- Concaténer deux listes. Ex (1 2)+(3 4)=(1 2 3 4)
  Comment démontrer que
  long(concat(l1,l2))=long(l1)+long(l2) ?
- Inverser l'ordre des éléments d'une liste
- Extraire la liste des éléments qui sont divisibles par deux
- Ajouter n à tous les éléments d'une liste

Les automates

- Un automate (accepteur) est une structure composée de
  - Un ensemble d'états X
  - Un état initial $x \in X$
  - Un ensemble d'états finaux $X_f \subseteq X$
  - Un alphabet Σ
  - Un ensemble de transitions $T \subseteq X \times \Sigma \times X$
- Un exemple est donné ci-dessous



- Identifier X, x, X_f, etc.

Les automates

- Une trace t est une suite d'éléments de Σ (notée aussi Σ^*)
- Un automate accepte une trace t si la suite des transitions de indiquée par t conduisent a un état final
- Quels sont les traces acceptées par l'automate précédent ?
- Exercices.
  - Proposer un automate acceptant les nombres à virgule.
  - Soit un fichier f, proposer un programme qui affiche tous les nombres contenus dans f