Analyste Programmeur en Automatisme, Robotique et Informatique Industrielle

TS ARII

Module MF 1.4

Analyser un projet informatique

Principes de programmation et algorithmes

Patrick MONASSIER – année 2019-2020

LA MACHE
- Supérieur -

Qualification des Services Intellectuels ISQ OPQF

Module 1.4 – Analyser un projet informatique

Compétences

Modéliser selon les modalités d'analyse orientées objet.

Objectifs

Décrire un traitement séquentiel
Définir un système de données.

Contenu

- **Algorithme**
  - **Type de données**
  - **Structure de contrôle**
  - **Procédures et fonctions**
- Méthode d'analyse UML
- Analyse MERISE.
  - Outil d'interface graphique
  - Langage SQL

Les principaux chapitres qui seront développés pendant le cours :

Introduction générale
Types de programmation
Les algorithmes
La Programmation Orientée Objet – POO
Les Bases de Données - SQL
La méthode d'analyse UML
L'analyse MERISE
Le langage SQL

de nombreux travaux pratiques et utilisations de logiciels

Sommaire du diaporama

L'homme calcule depuis 2000 ans avant Jésus-Christ **avec 10 chiffres** (0, 1, 2, 3, 4, 5, 6, 7, 8, 9), on parle alors de **base décimale** (ou base 10). Toutefois dans des civilisations plus anciennes ou pour certaines applications actuelles d'autres bases de calcul ont et sont toujours utilisées :

base sexagésimale (60) utilisée par les Sumériens. Cette base est également utilisée dans le système horaire actuel, pour les minutes et les secondes ;
base vicésimale (20) utilisée par les Mayas ;
base duodécimale (12) utilisée par les anglo-saxons dans leur système monétaire jusqu'en 1960 : un « pound » représentait vingt « shilling » et un « shilling » représentait douze « pences ». Le système d'heure actuel fonctionne également sur douze heures
base quinaire (5) utilisée par les Mayas ;

base binaire (2) utilisée par l'ensemble des technologies numériques.
Ne prend que deux valeurs : 0 et 1 *(fermé/ouvert – absent/présent…)*

2^3	2^2	2^1	2^0	
8	4	2	1	
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9
1	0	1	0	A
1	0	1	1	B
1	1	0	0	C
1	1	0	1	D
1	1	1	0	E
1	1	1	1	F

Nombre entier (*int*)

Un nombre entier, nombre sans virgule, peut être exprimé dans différentes bases :

- **Base décimale** : l'entier est représenté par une suite de chiffres unitaires (de 0 à 9) ne devant pas commencer par le chiffre 0
- **Base hexadécimale** : l'entier est représenté par une suite d'unités (de 0 à 9 ou de A à F (ou a à f)) devant commencer par 0x ou 0X – *exemple : OxFFE0*
- **Base octale** : l'entier est représenté par une suite d'unités (incluant uniquement des chiffres de 0 à 7) devant commencer par 0

- **Bit** *(Binary digit)* : ne prend que deux valeurs 0 et 1 *(ouvert/fermé – absent/présent…)*
- **Octet** : est un ensemble de 8 bits – *exemple : 00101101*
- ***N octets*** *: N fois des groupes de 8 bits*

- *Exemple : 1111 1111 1110 0000 binaire = 65504 décimal = FFE0 hexadécimal*

0	0	1	0	1	1	0	1
128	64	32	16	8	4	2	1
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

Unité	Symbole	Equivalence	Exemple
Bit	Bit	Unité de base	Le code ASCII à 7 bits
Octet	O	1 octet = 8 bits	Un caractère
Kilo octet	Ko	1 Ko = 2^{10} octets = 1024 octets	Un fichier texte de 10 Ko
Méga octet	Mo	1 Mo = 2^{20} = 1 048 576 octets = 1024 Ko	128 Mo de mémoire vive
Giga octet	Go	1 Go = 2^{30} octets = 1073741824 octets = 1024 Mo	Un disque dur de 40 Go
Téra octet	To	1 To = 2^{40} octets = 1099511627776 octets = 1024 Go	Un système RAID de 1 To

Caractère (*char*)

Le type **char** (provenant de l'anglais *character*) permet de stocker la valeur ASCII d'un caractère, c'est-à-dire un nombre entier ! – Voir la tables ASCII pages suivantes

Table ASCII

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	0	NUL (null)	32	20	Space	64	40	@	96	60	`
1	1	SOH (start of heading)	33	21	!	65	41	A	97	61	a
2	2	STX (start of text)	34	22	"	66	42	B	98	62	b
3	3	ETX (end of text)	35	23	#	67	43	C	99	63	c
4	4	EOT (end of transmission)	36	24	$	68	44	D	100	64	d
5	5	ENQ (enquiry)	37	25	%	69	45	E	101	65	e
6	6	ACK (acknowledge)	38	26	&	70	46	F	102	66	f
7	7	BEL (bell)	39	27	'	71	47	G	103	67	g
8	8	BS (backspace)	40	28	(	72	48	H	104	68	h
9	9	TAB (horizontal tab)	41	29	)	73	49	I	105	69	i
10	A	LF (NL line feed, new line)	42	2A	*	74	4A	J	106	6A	j
11	B	VT (vertical tab)	43	2B	+	75	4B	K	107	6B	k
12	C	FF (NP form feed, new page)	44	2C	,	76	4C	L	108	6C	l
13	D	CR (carriage return)	45	2D	-	77	4D	M	109	6D	m
14	E	SO (shift out)	46	2E	.	78	4E	N	110	6E	n
15	F	SI (shift in)	47	2F	/	79	4F	O	111	6F	o
16	10	DLE (data link escape)	48	30	0	80	50	P	112	70	p
17	11	DC1 (device control 1)	49	31	1	81	51	Q	113	71	q
18	12	DC2 (device control 2)	50	32	2	82	52	R	114	72	r
19	13	DC3 (device control 3)	51	33	3	83	53	S	115	73	s
20	14	DC4 (device control 4)	52	34	4	84	54	T	116	74	t
21	15	NAK (negative acknowledge)	53	35	5	85	55	U	117	75	u
22	16	SYN (synchronous idle)	54	36	6	86	56	V	118	76	v
23	17	ETB (end of trans. block)	55	37	7	87	57	W	119	77	w
24	18	CAN (cancel)	56	38	8	88	58	X	120	78	x
25	19	EM (end of medium)	57	39	9	89	59	Y	121	79	y
26	1A	SUB (substitute)	58	3A	:	90	5A	Z	122	7A	z
27	1B	ESC (escape)	59	3B	;	91	5B	[	123	7B	{
28	1C	FS (file separator)	60	3C	<	92	5C	\	124	7C	|
29	1D	GS (group separator)	61	3D	=	93	5D	]	125	7D	}
30	1E	RS (record separator)	62	3E	>	94	5E	^	126	7E	~
31	1F	US (unit separator)	63	3F	?	95	5F	_	127	7F	DEL

Dec : base décimale
Hex : Base hexadécimale
Char : caractère ASCII

ASCII = " American Standard Code for Information Interchange"
"Code américain normalisé pour l'échange d'information".

Table ASCII étendue

Dec	Char	Dec	Char	Dec	Char	Dec	Char	Dec	Char	Dec	Char	Dec	Char	Dec	Char
128	Ç	144	É	161	í	177	▒	193	┴	209	╤	225	ß	241	±
129	ü	145	æ	162	ó	178	▓	194	┬	210	╥	226	Γ	242	≥
130	é	146	Æ	163	ú	179	│	195	├	211	╙	227	π	243	≤
131	â	147	ô	164	ñ	180	┤	196	─	212	╘	228	Σ	244	⌠
132	ä	148	ö	165	Ñ	181	╡	197	┼	213	╒	229	σ	245	⌡
133	à	149	ò	166	ª	182	╢	198	╞	214	╓	230	µ	246	÷
134	å	150	û	167	º	183	╖	199	╟	215	╫	231	τ	247	≈
135	ç	151	ù	168	¿	184	╕	200	╚	216	╪	232	Φ	248	°
136	ê	152	ÿ	169	⌐	185	╣	201	╔	217	┘	233	Θ	249	∙
137	ë	153	Ö	170	¬	186	║	202	╩	218	┌	234	Ω	250	·
138	è	154	Ü	171	½	187	╗	203	╦	219	█	235	δ	251	√
139	ï	156	£	172	¼	188	╝	204	╠	220	▄	236	∞	252	ⁿ
140	î	157	¥	173	¡	189	╜	205	═	221	▌	237	φ	253	²
141	ì	158	₧	174	«	190	╛	206	╬	222	▐	238	ε	254	■
142	Ä	159	ƒ	175	»	191	┐	207	╧	223	▀	239	∩	255	
143	Å	160	á	176	░	192	└	208	╨	224	α	240	≡		

Types de données - Norme IEEE-754

	Type de données	Signification	Taille (en octets)	Minimum	Maximum
Booléen	booléen	Vrai / faux	1 bit	0	1
Caractères	char	Caractère	1	-128	127
	unsigned char	Caractère non signé	1	0	255
Entiers	short int	Entier court	2	-32 768	32 767
	unsigned short int	Entier court non signé	2	0	65 535
	int	Entier	4	-2 147 483 648	2 147 483 647
	unsigned int	Entier non signé	4	0	4 294 967 295
	long int	Entier long	4	-2 147 483 648	2 147 483 647
	unsigned long int	Entier long non signé	4	0	4 294 967 295
Réels	float	Flottant (réel)	4	$3.4*10^{-38}$	$3.4*10^{38}$
	double	Flottant double	8	$1.7*10^{-308}$	$1.7*10^{308}$
	long double	Flottant double long	10	$3.4*10^{-4932}$	$3.4*10^{4932}$

Codage des réels - norme IEEE-754

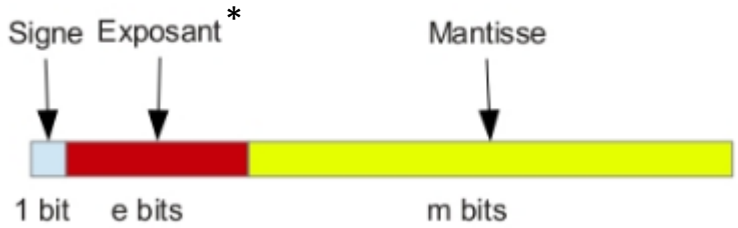
Les nombres de type **float** sont codés sur 32 bits dont :
- 23 bits pour la mantisse
- 8 bits pour l'exposant
- 1 bit pour le signe

Les nombres de type **double** sont codés sur 64 bits dont :
- 52 bits pour la mantisse
- 11 bits pour l'exposant
- 1 bit pour le signe

Les nombres de type **long double** sont codés sur 80 bits dont :
- 64 bits pour la mantisse
- 15 bits pour l'exposant
- 1 bit pour le signe

La précision des nombres réels est approchée. Elle dépend du nombre de positions décimales, elle sera au moins :
- de 6 chiffres pour le type **float**
- de 15 chiffres pour le type **double**
- de 17 chiffres pour le type **long double**

Exemple : $-1,3254 = \underbrace{-}_{\text{Signe}}\ \underbrace{13254}_{\text{Mantisse}}\ \times\ \underbrace{10^{-4}}_{\text{Exposant}}$



* *Pour des raisons pratiques, on ne stocke pas la vraie valeur de l'exposant mais l'exposant décalé c'est-à-dire un nombre entier compris entre* ***0*** *et* 2^{e-1} *plutôt qu'un nombre compris entre* $-2^{e-1}+1$ *et* 2^{e-1}. Le décalage (ou biais) est égal à $2^{e-1}-1$.Pour une valeur de **e=8**, l'exposant décalé sera donc compris entre **0** et **255** plutôt qu'entre **-127** et **128** et le décalage sera de **127**

Un programme écrit en langage C est composé de deux parties :

Partie 1 : Les déclarations

Déclaration des fonctions des bibliothèques (bibliothèque standard ou autre) par inclusion de fichiers fournis avec le langage et peut comprendre des déclarations des variables « globales ».

Partie 2 : Le corps du programme

Tout programme C doit comporter une fonction principale **main**. Cette fonction est celle utilisée par le système pour exécuter le programme.

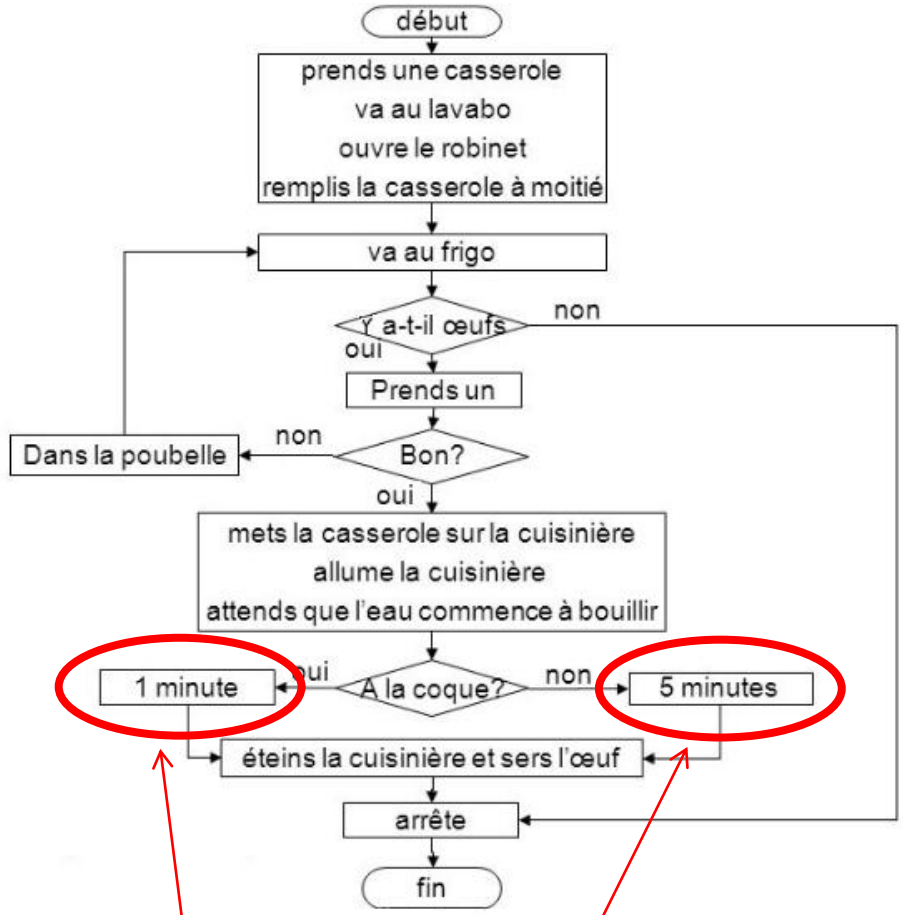
```
#include <stdio.h>

int main(void) {
  printf("hello, world\n") ;
  return 0 ;
}
```

Un programme important en taille peut nécessiter de reproduire le même traitement à différents moments et sur des données différentes, d'où l'introduction de « **fonctions** »

- Réduction de la taille du code.
- Facilité de programmation
- Structure du code plus simple
- Gain de temps pour le programmeur
- Un changement unique et bien localisé en cas de modification et d'évolution du code

Différents noms désignent ces portions de code réutilisées : **routines, sous-programmes, fonctions et procédures**.

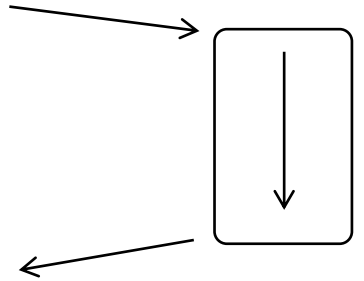


Fonction : bouillir (n minutes)

Structure d'un sous-programme

- ✓ Une **routine** *(ou sous-programme)* encapsule une portion de code
- ✓ Une **procédure** est une routine qui ne renvoie pas de valeur
- ✓ Une **fonction** est une routine qui renvoie une valeur

Appel de la **procédure** (avec ou sans valeur)



Exécution du code

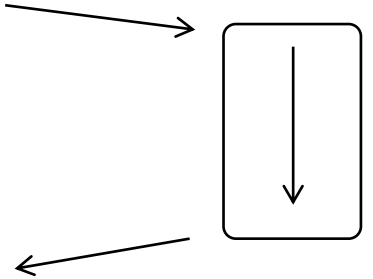
Retour de la procédure (sans valeur : void)

Exemple :

Résultat = moyenne *3 valeurs de températures*

Double moyenne_temperature(float A, float B, float C)

Appel de la **fonction** (avec ou sans valeurs)



Exécution du code

Retour de la fonction (avec valeur)

Exercices :
Organigrammes et codage

```
#include <stdio.h>

double moyenne_temperature(float A, float B, float C)
{
    Moy = (A+B+C) / 3;
    return Moy;
}

int main(void)
{
      MoyenneT = moyenne_temperature(ValT1, ValT2, ValT3);
      return 0 ;
}
```

La fonction moyenne_temperature est appelée à partir du programme principal en passant les valeurs des 3 températures et elle retourne la valeur moyenne des 3 températures. Les variables restent à déclarer…

Toutes les **variables** doivent être déclarées
Un variable est identifiée par son **nom** et par son **type**

Exemples : int Longueur; unsigned int Altitude; float ValT1; …

Les variables **caractères** et **chaines de caractères** :

Il n'existe pas de type spécial chaîne ou string en C. Une chaîne de caractères est traitée comme un tableau à une dimension de caractères. Il existe quand même des notations particulières et une bonne quantité de fonctions spéciales pour le traitement de ***tableaux de caractères***

Exemples : char chaine[]; char chaine[7] = "Bonjour"; char chaine[] = "Bonjour";
chaine[0] = 'B'; chaine[1] = 'o'; chaine[2] = 'n'; chaine[3] = 'j'; chaine[4] = 'o'; ….

Exercice : déclarations et types de variables d'une cellule robotique, en tenant compte des précisions souhaitées et/ou imposées : La présence d'une pièce, une température, la mesure d'une pièce, la position d'un support mobile, l'angle du support …

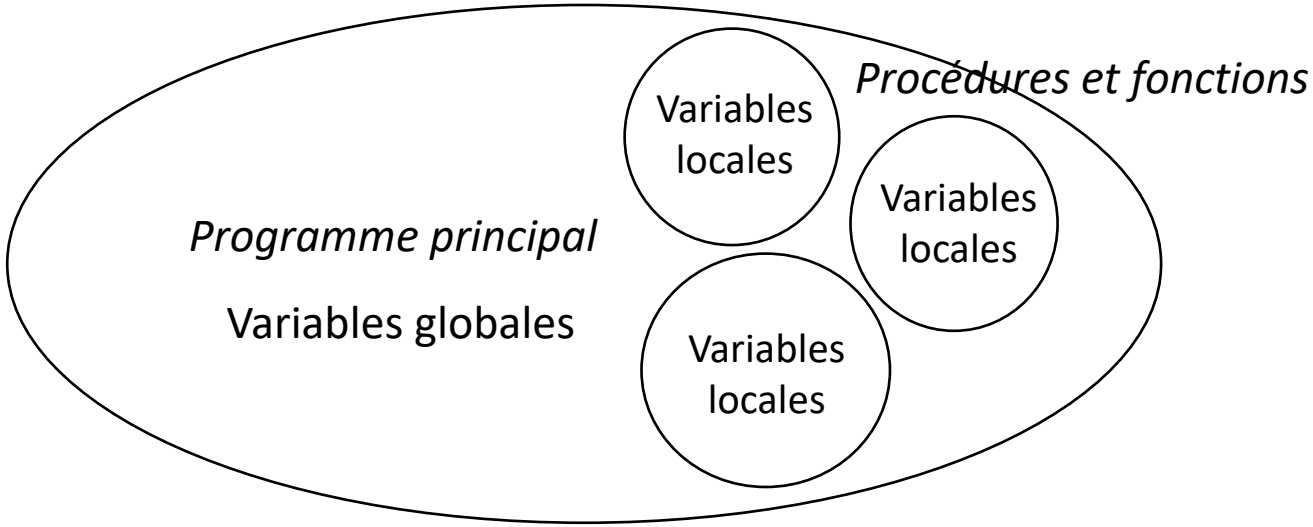
Une **variable locale** est une **variable** qui ne peut être utilisée que dans la fonction ou le bloc où elle est définie.

La **variable locale** s'oppose à la **variable globale** qui peut être utilisée dans tout le programme.
une **variable globale** est une variable déclarée à l'extérieur du corps de toute fonction.

Programme principal
Déclarations des variables globales

Fonction {
Déclarations des variables locales
……
}

Procédure {
Déclarations des variables locales
……
}



Des variables globales et locales peuvent porter un nom identique …. mais elles seront différentes !

Les variables déclarées au début du fichier, à l'extérieur de toutes les fonctions *sont disponibles à toutes les fonctions du programme*. Ce sont alors des ***variables globales***. En général, les variables globales sont déclarées immédiatement derrière les instructions **#include** au début du programme.

> ***Attention !*** *Les variables déclarées au début de la fonction principale* **main** ***ne sont pas*** *des variables globales, mais elles sont locales à* **main** ***!***

Les variables globales sont à utiliser avec précaution, puisqu'elles créent des *liens invisibles entre les fonctions*. La modularité d'un programme peut en souffrir et le programmeur risque de perdre la vue d'ensemble - Il faut faire attention à ne pas cacher involontairement des variables globales par des variables locales du même nom – La programmation défensive nous conseille ***d'écrire nos programmes aussi 'localement' que possible***. L'utilisation de ces variables rend plus difficile la compréhension d'un programme ainsi que son débuggage et sa modification ultérieure.

L'utilisation de variables globales devient inévitable, si plusieurs fonctions qui ne s'appellent pas ont besoin des mêmes variables ou si plusieurs fonctions d'un programme ont besoin du même ensemble de variables. Ce serait alors trop encombrant de passer toutes les variables comme paramètres d'une fonction à l'autre

Exemple variable Globale :

La variable tmp est déclarée localement

```
void echange(int a, int b)
{
        int tmp;  /* variable locale tmp */
        tmp = a;
        a = b;
        b = tmp;
}
```

Toute **variable** a une durée de vie bornée au bloc où elle est déclarée.
Ce bloc définit la **portée** de la **variable**

Exemple variable Globale :

La variable status est déclarée globalement pour pouvoir être utilisée dans les procédures A et B.

```
#include <stdio.h>
int status;

void A(...) {
     if (status)
      status -;
     else ...
        }

void B(...) {
     status ++;
      ... }
```

Tableaux simples, *à 1 dimension* :

Un **ensemble de valeurs** portant le même nom de variable et repérées par un nombre, s'appelle un **tableau**, ou encore une variable indicée. Le nombre qui, au sein d'un tableau, sert à repérer chaque valeur s'appelle **l'indice**. Chaque fois que l'on doit désigner un élément du tableau, on fait figurer le nom du tableau, suivi de l'indice de l'élément, entre parenthèses. L'indice qui sert à désigner les éléments d'un tableau peut être exprimé directement comme un nombre en clair, mais il peut être aussi une variable, ou une expression calculée.

```
int mon_tableau[4] = { 1, 2, 3,4 };
int i = 0;

  // Initialisation du tableau
  for (i = 0 ; i < 4 ; i++)  {
     mon_tableau[i] = 0;
  }
```

On peut en revanche décider ou avoir besoin de gérer *soi-même* la taille du tableau alors que le programme est en cours d'exécution. Faire cela s'appelle « allouer dynamiquement » de la mémoire au programme (tableau dynamique).

Tableaux multidimensionnels, *à plusieurs dimensions* :

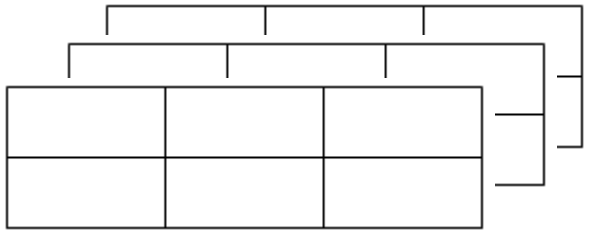
Techniquement, un tableau multidimensionnel est un tableau dont les éléments sont eux-mêmes des tableaux. Dès lors, vous avez besoin d'autant d'indices qu'il y a de dimensions. Par exemple, pour un tableau à deux dimensions, vous avez besoin d'un premier indice pour accéder à l'élément souhaité du premier tableau, mais comme cet élément est lui-même un tableau, vous devez utiliser un second indice pour sélectionner un élément de celui-ci.

```
#include <stdio.h>

int main(void) {
    int tab[2][3] = { { 1, 2,3 }, { 4, 5,6 } };
    for (unsigned i = 0; i < 2; ++i) {
        for (unsigned j = 0; j < 3; ++j) {
            tab[i][j] = 0;
            }
        }
}
```

Tableau 2 dimensions

Tableau n dimensions



Les structures :

Différence entre une **structure** et un tableau : un **tableau** permet de regrouper et manipuler des éléments de mêmes types, c'est-à-dire codés sur le même nombre d'octets et de la même façon. Les **structures** regroupent des champs (ou attributs) **de natures différentes** au sein d'une entité repérée par un seul nom de variable.

```
struct  NomDeMaStructure
{
   char variable1;
   int variable2;
   int autreVariable;
   double nombreDecimal;
};
```

```
NomDeMaStructure.variable1 = ''D '';
NomDeMaStructure.variable2 = 5 ;
NomDeMaStructure. nombreDecimal = 4.12 ;
```

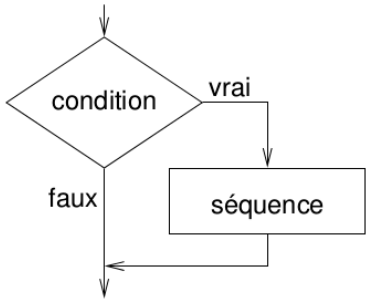
Une **structure de contrôle** est une instruction particulière d'un langage de programmation impératif pouvant dévier le flot de **contrôle** du programme la contenant lorsqu'elle est exécutée. Elles permettent de réaliser des tests, et suivant le résultat de ces tests, d'exécuter des parties de code différentes

Les opérateurs logiques

Opérateur	Signification
==	c'est le test d'équivalence
!=	différent (non équivalence)
>	supérieur
>=	supérieur ou égal
<	inférieur
<=	inférieur ou égal
&&	ET logique
||	OU logique
!	NOT logique
^	XOR logique

If…then…else

```
If  (condition)  {
   instruction1 ;
}
```




```
If (condition)  {
   instruction1 ;
 } else {
   instruction2 ;
 }
```

```
if (condition1) {
   instruction1 ;
} else if (condition2) {
    instruction2 ;
} else if (condition3) {
    instruction_par_defaut;
}
```

```
if (a > b) {
   printf(" a est supérieur à b \n");
} else if(a == b) {
   printf(" a est égal à b \n");
} else {
    printf("a est inférieur à b \n");
 }
```

```
if (a == b) { printf ("les variables a et b sont égales\n"); }
```

Switch

Pour éviter les imbrications d'instructions **if**, le C possède une instruction qui permet d'explorer plusieurs cas en même temps : c'est l'instruction **switch**. La syntaxe du switch est la suivante :

```
Switch (variable) {
  case valeur1 :
     instruction10 ;
     instruction11 ;
     break;
  case valeur2 :
     instruction12 ;
     instruction13 ;
break;
….
  default :
     instruction par défaut ;
}
```

Si *variable* prend la valeur *valeur1* alors on exécute les instructions *instruction10* et *instruction11*, si elle prend la valeur *valeur2* on exécute les instructions *instruction12* et *instruction13*, etc. Par défaut (c'est à dire, si aucune des valeurs ci-dessus ne correspond à la variable), alors on exécute l'instruction *instruction par défaut.*

ATTENTION : Une fois l'exécution des instructions commencée à partir d'un "case" les autres instructions sont exécutées séquentiellement y compris celles des "case" suivants jusqu'à rencontrer l'instruction **break** qu'il ne faut donc surtout pas oublier. *(spécifique au langage C)*

Les structures de boucle :

Les **structures de boucles** (ou structures répétitives) constituent un élément important de la programmation : elles permettent d'exécuter des instructions en boucles soit pour un nombre d'itérations fixées à l'avance, soit jusqu'à ce qu'une condition soit remplie

La boucle POUR : « for »

```
for (initialisation ; condition ; incrémentation) {
      instructions à répéter
}
```

```
for (i = 0 ; i < 10 ; i = i + 1) {
      printf ("iteration %d\", i) ;
 }
```

La boucle FAIRE ... TANT QUE : « do ... While »

```
do {
   bloc d'instructions à répéter
} while (condition de rebouclage) ;
```

```
do {
   printf ("iteration %d \n", i) ;
   i = i + 1 ;
} while ( i < 10 ) ;
```

```
while (condition de boucle) {
bloc d'instructions répéter
}
```

```
while ( i < 10) {
   printf ("iteration %d \n", i) ;
   i = i + 1 ;
}
```

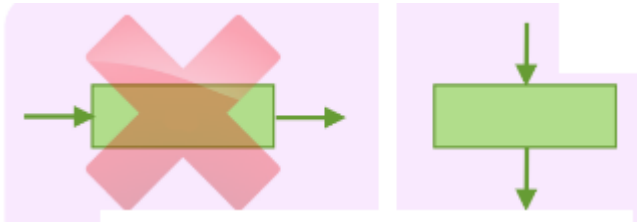
Diagramme de flux – organigramme informatique :

Le but d'un diagramme de flux est d'éliminer l'ambiguïté dans la compréhension du flux de données à travers un processus, donc il doit être clair, net et facile à suivre . La direction habituelle du débit est **de haut en bas** et de **gauche à droite** et de l'organigramme doit avoir un **départ** logique et une **fin** logique.

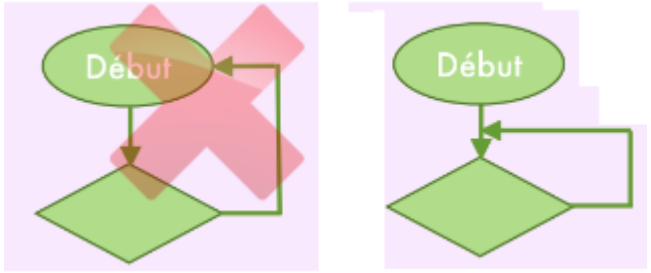
Six symboles de base de diagramme de flux

Début et Fin | Processus | Décision | Document | sous programme | Base de données

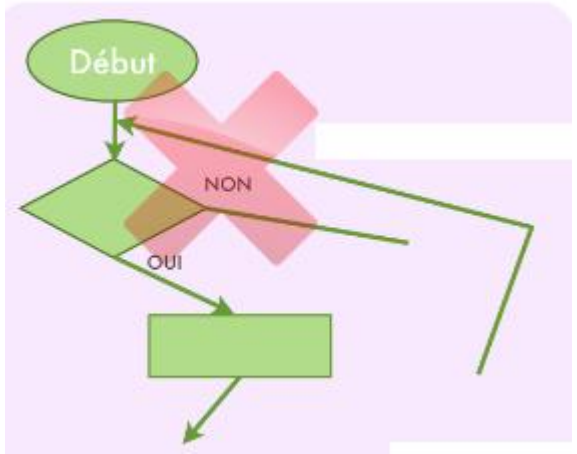
Quelques règles de conception



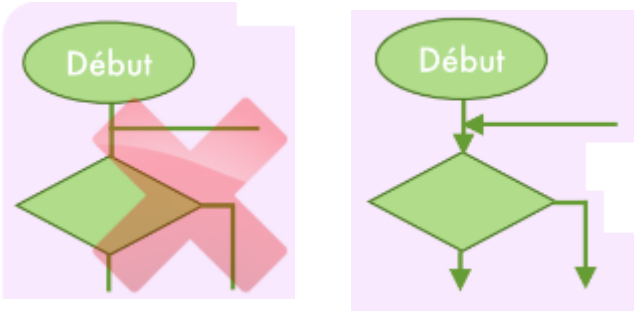
On arrive par dessus, on sort par dessous



D'une flèche, on va et on retourne à une autre flèche

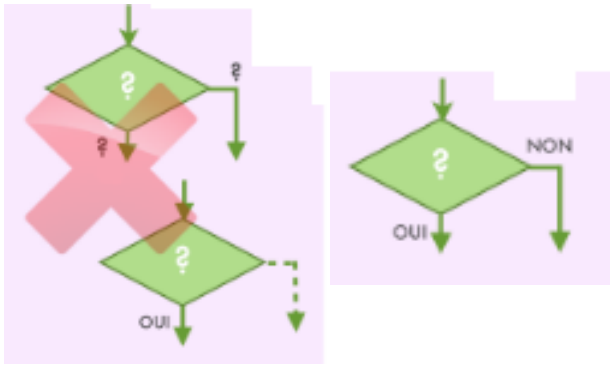


On utilise uniquement des trais horizontaux et de traits verticaux



On indique le sens des flèches, sinon on ne sait pas où on va !

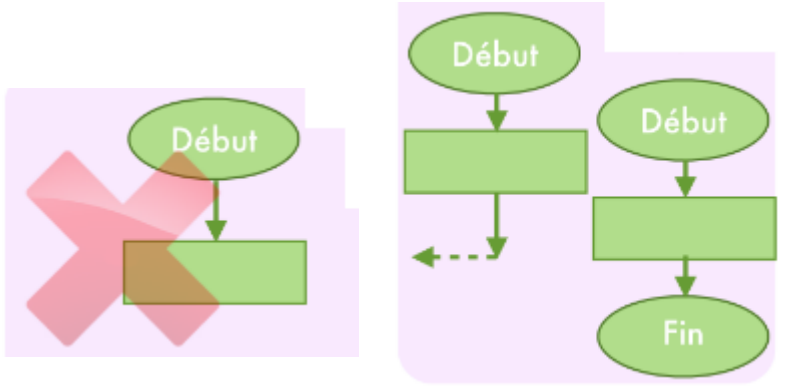
Quelques règles de conception



On n'oublie pas toutes les possibilités :

Si oui alors….

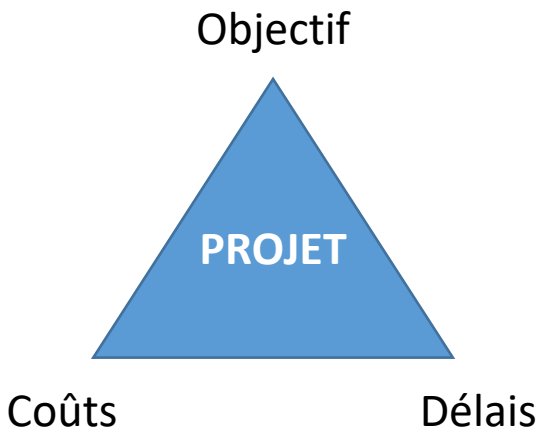
Si non alors…



La programmation doit se terminer par une fin ou revenir sur une boucle. Impossible de ne pas finir !

Le triangle O-C-D *(Objectif – Coûts - Délais)*

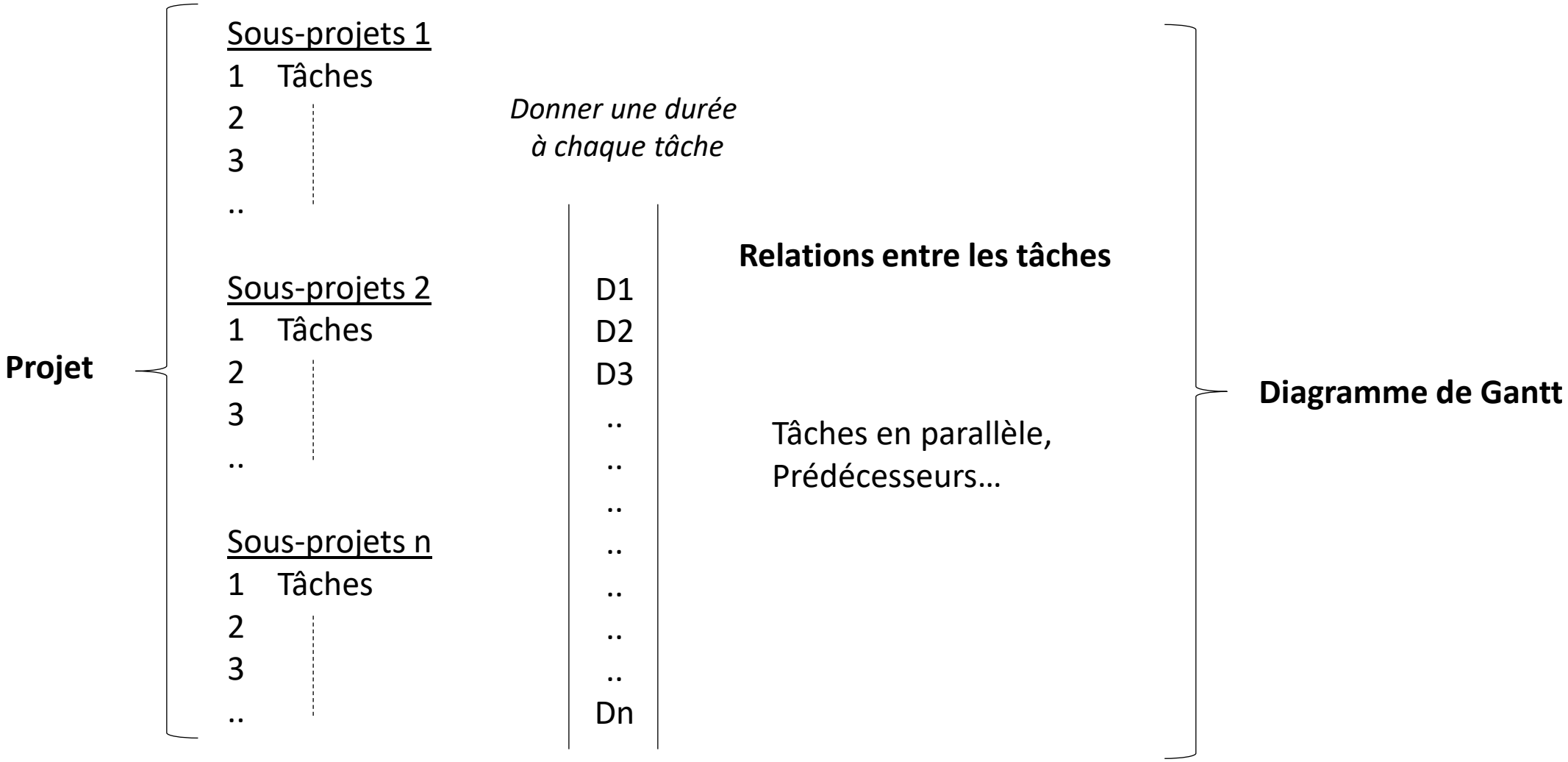
- Un projet comprend un objectif défini devant être livré dans un délai et à un coût convenu
- Un système dynamique à maintenir en équilibre : Chaque changement déséquilibre le projet



Le Projet

- Le décomposer en sous-projets : *n* étapes
- Les **étapes** ou **sous-projet** sont composées de **tâches**
- Donner une **durée** à chaque étape
- **Relations** qui existent entre les tâches
  Tâches en même temps, parallèle ou consécutives
  Prédécesseurs
- Organiser sous forme d'un diagramme **visuel**
  Diagramme de Gant par exemple

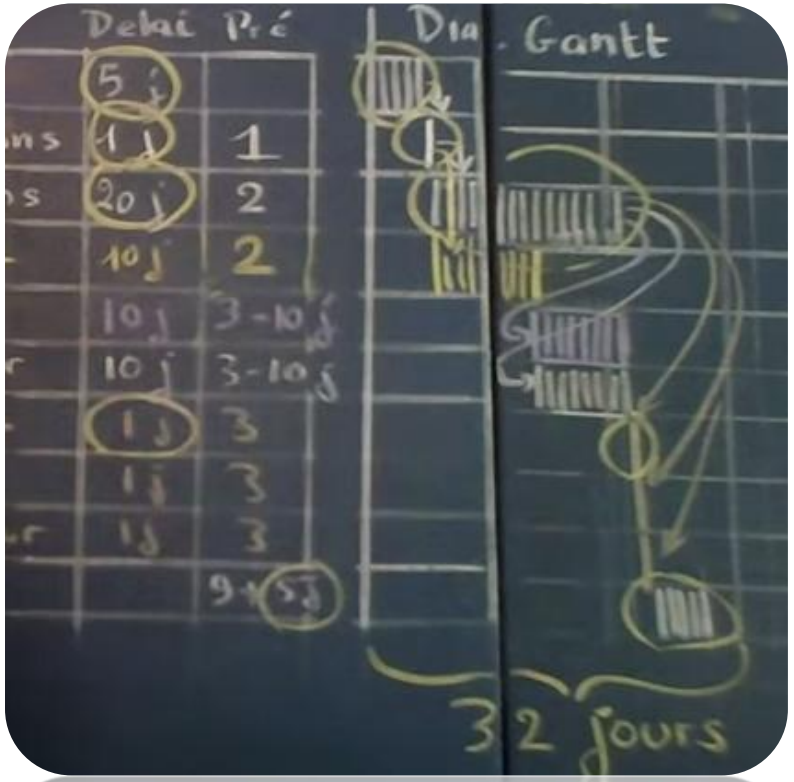
Décomposer en *n* sous-projets



N°	Tâches	Délai	Préd.
1	Sélection des invités	5 j	
2	Envoi postal	1 j	1
3	Collecte des retours	20 j	2
4	Recherche des animations	10 j	2
5	Recherche de la salle	10 j	3 – 10 j
6	Recherche du traiteur	10 j	3 – 10 j
7	Réserver animation	1 j	3
8	Réserver la salle	1 j	3
9	Réserver restaurateur	1 j	3
10	Soirée 10 juin		9 + 5 j

Chemin critique

Le planning

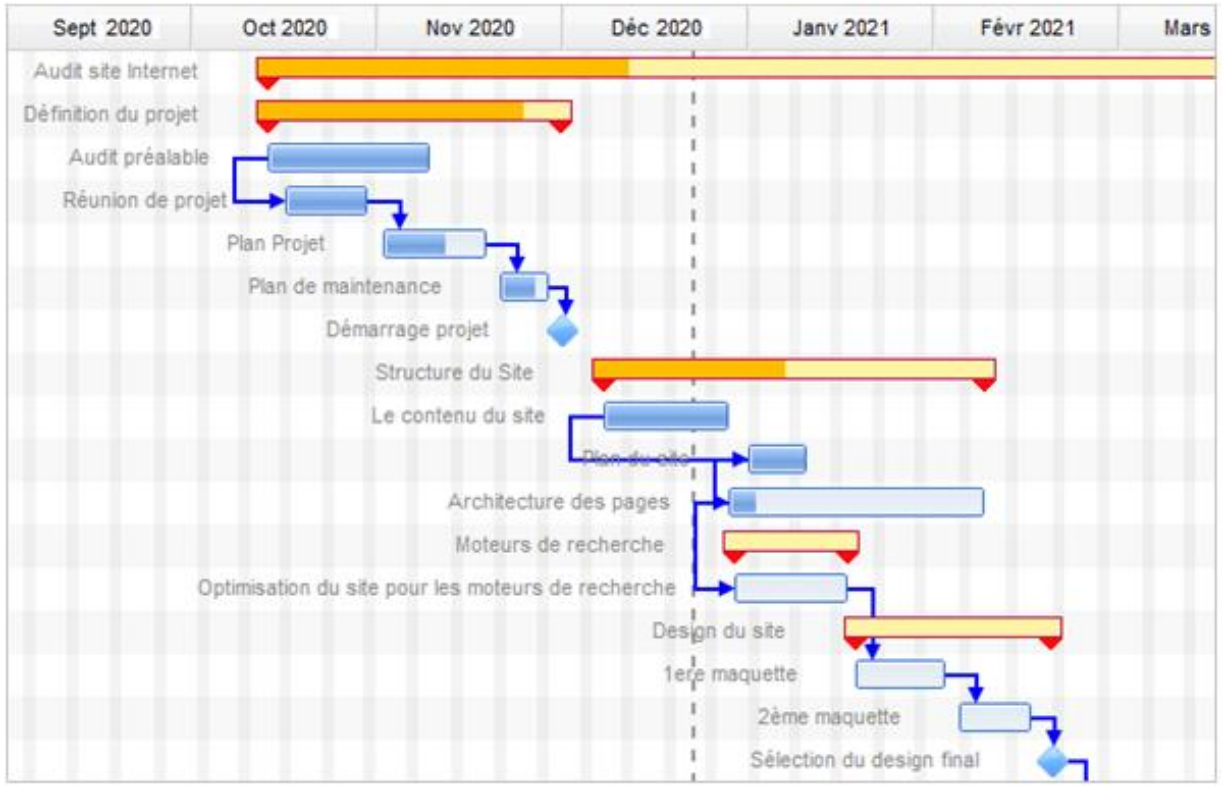


On parle aussi d'objectif **SMART**

- **S**pécifique (dans le sens personnalisé)
- **M**esurable (quels indicateurs ?)
- **A**mbitieux
- **R**éaliste (dans le sens accessible : pouvons-nous l'atteindre ?)
- **D**élimité dans le Temps (combien de temps pour atteindre l'objectif ?)

Logiciel gratuit « GanttProject » *http://ganttproject.biz/*

Diagramme de Gantt

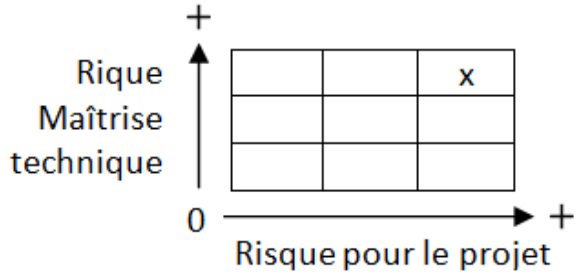


Evaluer les risques

Mettre des indicateurs d'évaluation

Noter tous les évènements

Le facteur humain est très important !



Analyste Programmeur en Automatisme, Robotique et Informatique Industrielle
TS ARII

Module MF 1.4 - Cours N° 1

Analyser un projet informatique

Principes de programmation et algorithmes

Fin de Présentation