Informatique Industrielle

Programmation - Langage C

1ère année – S1 (I1) – S2 (I2)

Documentation autorisée aux contrôles

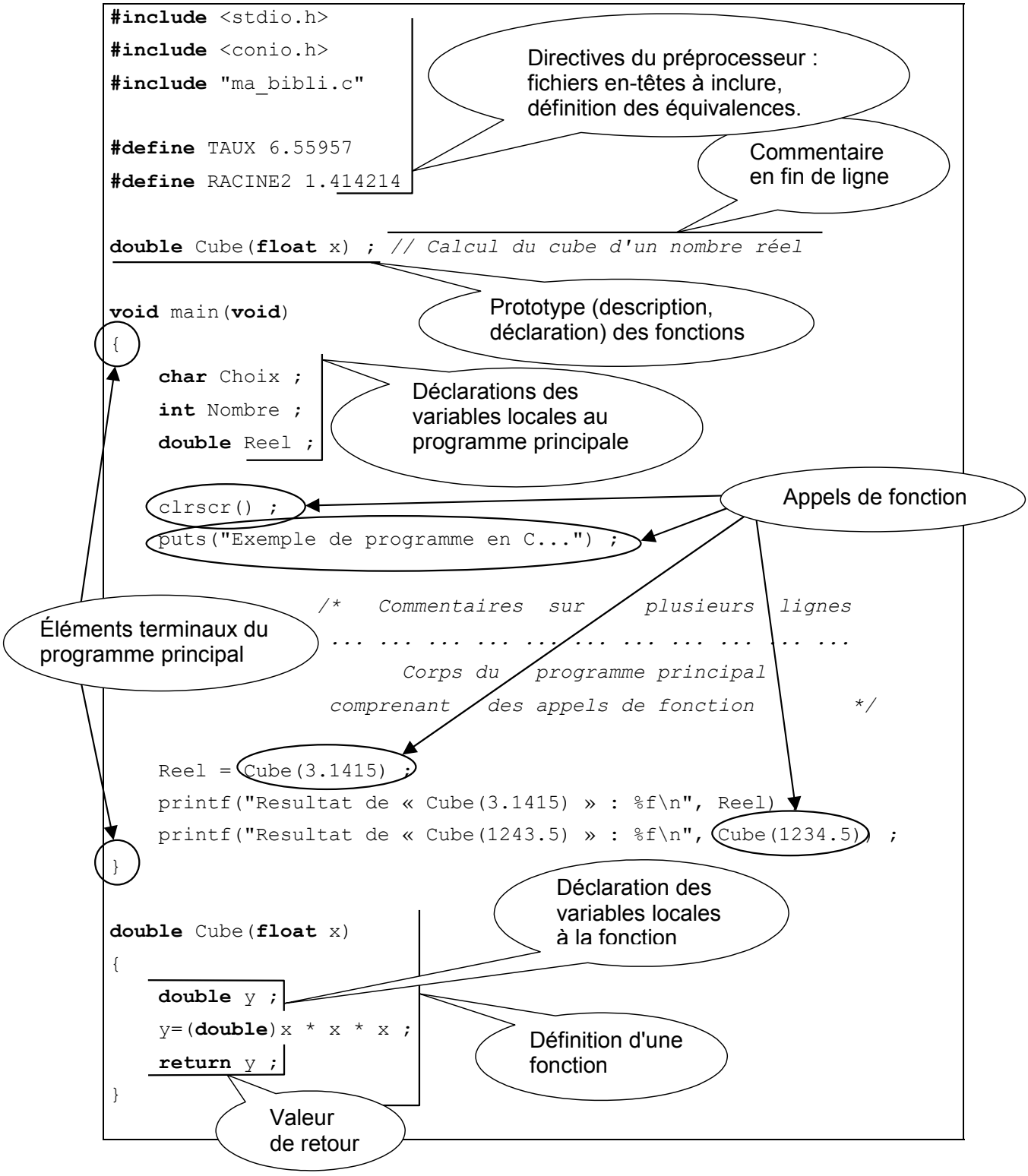
Responsable du module I1 : Pierre POISSON
Responsable du module I2 : Dominique BESSE

Équipe enseignante : Armel BRUNO, Dominique BESSE, Yann NEAU, Fabien NEBEL, Baudouin MARTIN, Pierre POISSON.

Version du document : 1.1.1. du 07/09/2006

SOMMAIRE

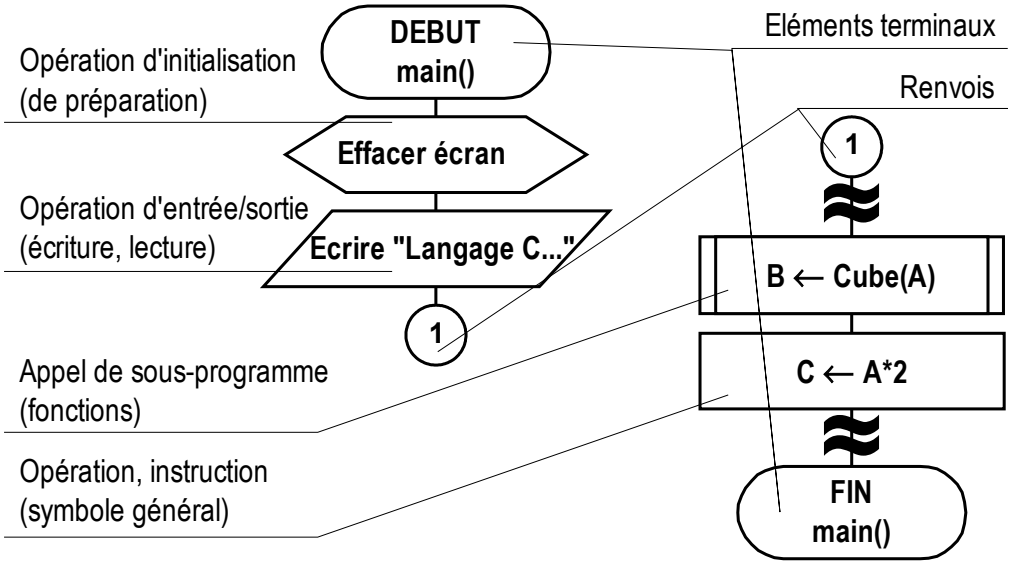
ORGANISATION D'UN PROGRAMME EN C



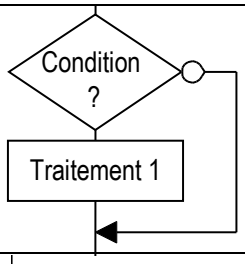
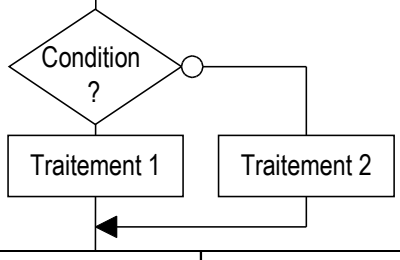
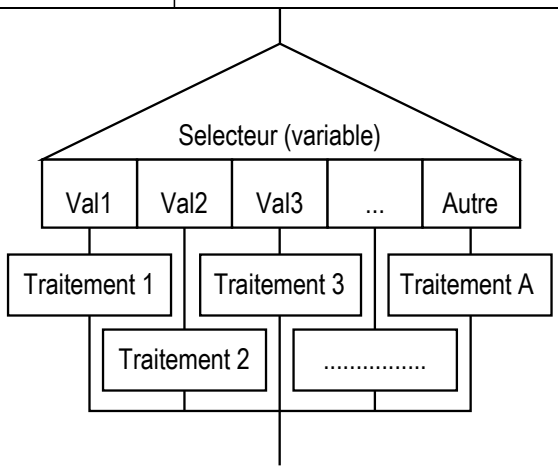
ORGANIGRAMMES DE PROGRAMMATION ET LANGAGE C

D'après NF Z67-010 Août 1975

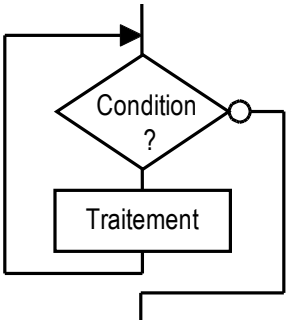
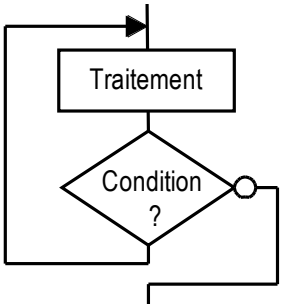
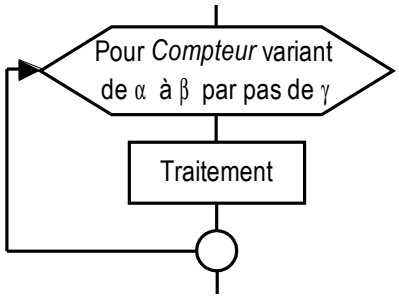
Éléments terminaux et opérations

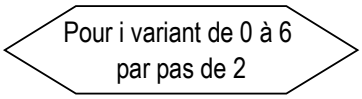
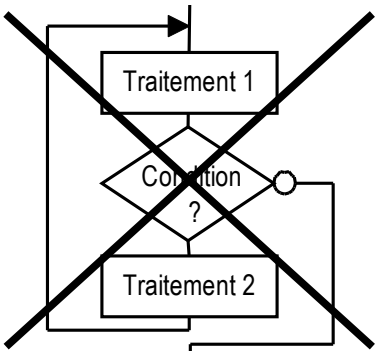


Structures alternatives

Alternative simple		`if (Condition)` `{` `  // Traitement1 ;` `}`
Alternative double		`if (Condition)` `{` `  // Traitement1 ;` `}` `else` `{` `  // Traitement2 ;` `}`
Sélecteur de choix		`switch(Selecteur)` `{` `  case Val1 :` `  // Traitement1 ;` `  break ;` `  case Val2 :` `  // Traitement2 ;` `  break ;` `  case Val3 :` `  // Traitement3 ;` `  break ;` `  /* … … … … … … */` `  default :` `  // TraitementA ;` `}`

Structures répétitives

<table>
<tr>
<td>Tant que…
(itération à condition d'entrée)</td>
<td> 
</td>
<td><pre>while (Condition)
{
    // Traitement ;
}</pre></td>
</tr>
<tr>
<td>Faire…
tant que
(itération à condition de sortie)</td>
<td> 
</td>
<td><pre>do
{
    // Traitement ;
}
while (Condition) ;</pre></td>
</tr>
<tr>
<td>Pour…
(itération avec compteur)</td>
<td> 

<u>Exemple :</u>
 
</td>
<td><pre>for (Init.; Cond.; Pas)
{
    // Traitement ;
}</pre>
<u>Exemple :</u>
<pre>for (i=0; i<=6; i=i+2)
{
    // Traitement ;
}</pre></td>
</tr>
<tr>
<td>Structure non programmable
(forme générale d'une structure d'itération)</td>
<td> 
</td>
<td>Cette structure n'existe ni en C, ni en Pascal !</td>
</tr>
</table>

Remarques

Toutes les extrémités d'un symbole doivent posséder un chemin.
Chaque structure de base n'a qu'une seule entrée et qu'une seule sortie.
Les structures peuvent être imbriquées les unes dans les autres (une opération de traitement peut être composée d'une ou de plusieurs structures).
Il n'existe qu'un seul point d'entrée et qu'un seul point de sortie pour l'ordinogramme. Pour le programme principale, il peut ne pas exister de point de sortie (boucle infinie).

Caractères spéciaux

Déc	Hex	Caract	Signification	En C
0	00	NUL	Null (caractère de fin de chaîne)	'\0'
1	01	SOH	Start of heading	
2	02	STX	Start of text	
3	03	ETX	End of text	
4	04	EOT	End of transmit	
5	05	ENQ	Enquiry	
6	06	ACK	Acknolege	
7	07	BEL	Audible bell (alarme)	'\a'
8	08	BS	Backspace (retour arrière)	'\b'
9	09	HT / TAB	Horizontal tab (tabulation horizontale)	'\t'
10	0A	LF	Line Feed (ligne suivante, nouvelle ligne)	'\n'
11	0B	VT	Vertical tab	
12	0C	FF	Form feed (saut de page)	'\f'
13	0D	CR	Cariage return (retour au début de la ligne)	'\r'
14	0E	SO	Shift out	
15	0F	SI	Shift in	
16	10	DLE	Data link espace	
17	11	DC1 / XON	Device control 1 (signale l'attente de données, autorise la transmission de données)	'\X11'
18	12	DC2	Device control 2	
19	13	DC3 / XOFF	Device control 3 (signale la fin de la réception des données, demande l'arrêt de la transmission)	'\X13'
20	14	DC4	Device control 4	
21	15	NAK	Neg. acknowledge	
22	16	SYN	Synchronous idle	
23	17	ETB	End trans. block	
24	18	CAN	Cancel	
25	19	EM	End of medium	
26	1A	SUB	Substitution	
27	1B	ESC	Escape	
28	1C	FS	File separator	
29	1D	GS	Group separator	
30	1E	RS	Record separator	
31	1F	US	Unit Separator	
34	22	"	Guillemets anglais	'\"'
39	27	'	Apostrophe	'\''
48	30	0		'0'
65	41	A		'A'
92	5C	\	"Back slash", barre de fraction inverse	'\\'

Codes ASCII étendus

(cf. page suivante)

DEC	HEX	CHAR
0	00	
1	01	☺
2	02	☻
3	03	♥
4	04	♦
5	05	♣
6	06	♠
7	07	•
8	08	◘
9	09	○
10	0A	◙
11	0B	♂
12	0C	♀
13	0D	♪
14	0E	♫
15	0F	☼
16	10	►
17	11	◄
18	12	↕
19	13	‼
20	14	¶
21	15	§
22	16	▬
23	17	↨
24	18	↑
25	19	↓
26	1A	→
27	1B	←
28	1C	∟
29	1D	↔
30	1E	▲
31	1F	▼

DEC	HEX	CHAR
32	20	
33	21	!
34	22	"
35	23	#
36	24	$
37	25	%
38	26	&
39	27	'
40	28	(
41	29	)
42	2A	*
43	2B	+
44	2C	,
45	2D	-
46	2E	.
47	2F	/
48	30	0
49	31	1
50	32	2
51	33	3
52	34	4
53	35	5
54	36	6
55	37	7
56	38	8
57	39	9
58	3A	:
59	3B	;
60	3C	<
61	3D	=
62	3E	>
63	3F	?

DEC	HEX	CHAR
64	40	@
65	41	A
66	42	B
67	43	C
68	44	D
69	45	E
70	46	F
71	47	G
72	48	H
73	49	I
74	4A	J
75	4B	K
76	4C	L
77	4D	M
78	4E	N
79	4F	O
80	50	P
81	51	Q
82	52	R
83	53	S
84	54	T
85	55	U
86	56	V
87	57	W
88	58	X
89	59	Y
90	5A	Z
91	5B	[
92	5C	\
93	5D	]
94	5E	^
95	5F	_

DEC	HEX	CHAR
96	60	`
97	61	a
98	62	b
99	63	c
100	64	d
101	65	e
102	66	f
103	67	g
104	68	h
105	69	i
106	6A	j
107	6B	k
108	6C	l
109	6D	m
110	6E	n
111	6F	o
112	70	p
113	71	q
114	72	r
115	73	s
116	74	t
117	75	u
118	76	v
119	77	w
120	78	x
121	79	y
122	7A	z
123	7B	{
124	7C	|
125	7D	}
126	7E	~
127	7F	⌂

DEC	HEX	CHAR OEM	CHAR ANSI
128	80	Ç	€
129	81	ü	
130	82	é	‚
131	83	â	*f*
132	84	ä	„
133	85	à	…
134	86	å	†
135	87	ç	‡
136	88	ê	ˆ
137	89	ë	‰
138	8A	è	Š
139	8B	ï	‹
140	8C	î	Œ
141	8D	ì	
142	8E	Ä	Ž
143	8F	Å	
144	90	É	
145	91	æ	‘
146	92	Æ	’
147	93	ô	“
148	94	ö	”
149	95	ò	•
150	96	û	–
151	97	ù	—
152	98	ÿ	˜
153	99	Ö	™
154	9A	Ü	š
155	9B	ø	›
156	9C	£	œ
157	9D	Ø	
158	9E	×	ž
159	9F	*f*	Ÿ

DEC	HEX	CHAR OEM	CHAR ANSI
160	A0	á	
161	A1	í	¡
162	A2	ó	¢
163	A3	ú	£
164	A4	ñ	¤
165	A5	Ñ	¥
166	A6	ª	¦
167	A7	º	§
168	A8	¿	¨
169	A9	⌐	©
170	AA	¬	ª
171	AB	½	«
172	AC	¼	¬
173	AD	¡	-
174	AE	«	®
175	AF	»	¯
176	B0	░	°
177	B1	▒	±
178	B2	▓	²
179	B3	│	³
180	B4	┤	´
181	B5	╡	µ
182	B6	╢	¶
183	B7	╖	·
184	B8	╕	¸
185	B9	╣	¹
186	BA	║	º
187	BB	╗	»
188	BC	╝	¼
189	BD	╜	½
190	BE	╛	¾
191	BF	┐	¿

DEC	HEX	CHAR OEM	CHAR ANSI
192	C0	└	À
193	C1	┴	Á
194	C2	┬	Â
195	C3	├	Ã
196	C4	─	Ä
197	C5	┼	Å
198	C6	╞	Æ
199	C7	╟	Ç
200	C8	╚	È
201	C9	╔	É
202	CA	╩	Ê
203	CB	╦	Ë
204	CC	╠	Ì
205	CD	═	Í
206	CE	╬	Î
207	CF	╧	Ï
208	D0	╨	Đ
209	D1	╤	Ñ
210	D2	╥	Ò
211	D3	╙	Ó
212	D4	╘	Ô
213	D5	╒	Õ
214	D6	╓	Ö
215	D7	╫	×
216	D8	╪	Ø
217	D9	┘	Ù
218	DA	┌	Ú
219	DB	█	Û
220	DC	▄	Ü
221	DD	▌	Ý
222	DE	▐	Þ
223	DF	▀	ß

DEC	HEX	CHAR OEM	CHAR ANSI
224	E0	α	à
225	E1	ß	á
226	E2	Γ	â
227	E3	∏	ã
228	E4	∑	ä
229	E5	σ	å
230	E6	µ	æ
231	E7	τ	ç
232	E8	Φ	è
233	E9	θ	é
234	EA	Ω	ê
235	EB	δ	ë
236	EC	∞	ì
237	ED	Ø	í
238	EE	∈	î
239	EF	∩	ï
240	F0	≡	ð
241	F1	±	ñ
242	F2	≥	ò
243	F3	≤	ó
244	F4	⌠	ô
245	F5	⌡	õ
246	F6	÷	ö
247	F7	≈	÷
248	F8	▫	ø
249	F9	¨	ù
250	FA	·	ú
251	FB	√	û
252	FC	ⁿ	ü
253	FD	²	ý
254	FE	■	þ
255	FF		ÿ

LES OPÉRATEURS (EN C)

Opérateurs arithmétiques

+	Addition	`A + 5`
-	Soustraction Opérateur unaire d'opposition	`B - C` `-3`
*	Multiplication	`A * 2.5`
/	Division	`B / 5` `C / 6.2`
%	Modulo (reste de la division de 2 entiers)	`B % 5`

Opérateurs de manipulation de bits

(par exemple, sur un octet)

&	ET logique (bit à bit)	`A & 0xA5`
|	OU logique inclusif (bit à bit)	`B | 0x2F`
^	OU logique exlusif (bit à bit)	`A ^ 0xD6`
~	Complémentation à un (opérateur unaire)	`~B`
<<	Décalage à gauche	`A << 2`
>>	Décalage à droite	`B >> 2`

Opérateurs d'affectation

`=`	Opérateur d'affectation ordinaire
`--` `++`	Décrémentation et incrémentation
`+=` `-=` `*=` `/=` `%=` `&=` `|=` `^=` `<<=` `>>=`	Opérateurs d'affectation composés

Exemples (et comportement équivalent)

```
i++ ;
++i ;
```
```
i = i+1 ;
```

```
i-- ;
--i ;
```
```
i = i-1 ;
```

```
Tab[i++] = A ;
```
```
Tab[i] = A ;
i=i+1 ;
```

```
Tab[++i] = A ;
```
```
i=i+1 ;
Tab[i] = A ;
```

```
B = A-- + 5 ;
```
```
B = A + 5 ;
A = A - 1 ;
```

```
A += 3 ;
```
```
A = A + 3 ;
```

```
B %= 10 ;
```
```
B = B % 10 ;
```

```
A &= 0x0F ;
```
```
A = A & 0x0F ;
```

```
A <<= 2 ;
```
```
A = A << 2 ;
```

L'usage des opérateurs d'affectation composés (y compris l'incrémentation et la décrémentation) est déconseillé aux programmeurs ne maîtrisant pas ces notations.

Opérateurs relationnels (tests de comparaison)

>	Supérieur à	`(A>5)`
>=	Supérieur ou égal à	`(B>=3)`
<	Inférieur à	`(A<0x0F)`
<=	Inférieur ou égal à	`(C<=B)`
==	Identique à (égalité)	`(A==B)`
!=	Différent de (inégalité)	`(A!=B)`

Opérateurs propositionnels

!	Opérateur unaire de négation	`!(A>5)`
&&	ET logique	`(B>=3)&&(A<5)`
||	OU logique	`(B>=3)||(A<5)`

Application du théorème de DE MORGAN :

`!((B>=3)&&(A<5))`
est équivalent à
`(B<3)||(A>=5)`

Exemples de syntaxe

```
do
{
   /* --- */
   i = i + 1 ;
}
while((A<=5)&&(B!=3)) ;
```

```
if((A != 0)||(B == 0xF0))
{
   /* --- */
   C = A & 0x0F ;
}
```

Opérateurs d'accès aux objets

`&`	Obtention de l'adresse d'un objet (opérateur unaire)	`&A /* Adresse de A */`
`*`	Adressage indirect. Contenu d'un pointeur. (opérateur unaire)	`*pB /* Contenu de l'espace mémoire dont l'adresse est pB */`
	Déclaration d'un pointeur (opérateur unaire)	`int * pB ;` `float * pC ;`
`[ ]`	Adressage indexé. Accès à l'élément d'un tableau. `Tab[i]` est équivalent à `*(Tab+i)` `Tab` est équivalent à `&Tab[0]`	`Tab[i] = A ;`

Autres opérateurs

`sizeof` — Obtention de la taille d'une variable (en octets)
(opérateur unaire sur un type ou sur une variable)

```
A = sizeof(Tableau) ;
printf("Taille d'un \"int\" : %d\n", sizeof(int)) ;
```

`(char)` `(int)` `(float)` `(double)` ***etc.*** — Opérateurs de conversion de type (Opérateurs de coercition)
Opérateurs de "cast"

```
Y = (float) J / 2 ; /* X et Y sont des réels de type float */
J = (int) X ;       /* J est un entier de type int */
A = (char) J ;      /* A est un entier de type char */
```

Priorité des opérateurs (classement par ordre de priorité décroissante)

`( )  [ ]`
Opérateurs unaires : `!  ~  ++  --  *  -  &  (type)  sizeof( )`
`*  /  %`
`+  -`
`<<  >>`
`<  <=  >  >=`
`==  !=`
`&`
`^`
`|`
`&&`
`||`
`=  +=  -=  *=  /=  %=  >>=  <<=  |=  ^=`

Il est fortement conseillé d'utiliser des paires de parenthèses pour clarifier une expression il ne faut pas abuser de la règle sur la priorité des opérateurs.

Il faut écrire `A = (((A-0x05)<<2) | ((B>>4)&0x0F)) ;` plutôt que `A = A-0x05<<2 | B>>4&0x0F ;`

Nombres

Type	*Taille Étendue*	*Exemple de déclaration*	*Format d'affichage Exemple d'écriture (écran)*	*Exemple de lecture (clavier)*
unsigned char nombre entier non signé codé sur 1 octet	1 octet 0 à + 255	`unsigned char Positif ;`	**%d - %x** `printf("Le nombre : %d", Positif) ;` `printf("En hexa. : %x", Positif);`	`unsigned char Temp[81] ;` `/* ... ... ... */` `gets (Temp) ;` `Positif = (unsigned char) atoi(Temp) ;`
char nombre entier signé codé sur 1 octet	1 octet - 128 à + 127	`char Entier`	**%d** `printf("Le nombre : %d", Entier) ;`	`unsigned char Temp[81] ;` `/* ... ... ... */` `gets (Temp) ;` `Entier = (char) atoi (Temp) ;`
int "Integer", nombre entier	2 octets − 32 768 à + 32 767	`int Nombre ;` `int Nb, Nb2 ;`	**%d - %x** `printf("Le nombre : %d", Nombre) ;` `printf("En hexa. :%x",Nb);`	`unsigned char Temp[81] ;` `/* ... ... ... */` `gets (Temp) ;` `Nombre = atoi(Temp) ;`
long (int) nombre entier de grande capacité	4 octets −2 147 483 648 à +2 147 483 647	`long int NbLg;` `long Entier;`	**%ld** `printf("Le nombre : %ld", Entier) ;`	`unsigned char Temp[81] ;` `/* ... ... ... */` `gets (Temp) ;` `Entier = atol(Temp) ;`
float "Floating point", réel	4 octets (±) 3,4 10^{-38} à 3,4 10^{+38} 7 ou 8 chiffres significatifs	`float R, NbR ;`	**%f** `printf("Nombre : %f",R) ;` `printf("Montant dû %.2f EUR", NbR) ;`	`unsigned char Temp[81] ;` `/* ... ... ... */` `gets (Temp) ;` `NbR = atof(Temp) ;`
double "Double", réel de grande capacité	8 octets (±) 1,7 10^{-308} à 1,7 10^{+308} 15 ou 16 chiffres significatifs	`double GdReel ;`	**%lf** `printf("Le nombre : %lf", GdReel) ;`	`unsigned char Temp[81] ;` `/* ... ... ... */` `gets (Temp) ;` `GdReel = atof(Temp) ;`

Caractères et chaînes

Type	*Taille*	*Exemple de déclaration*	*Format d'affichage* Exemple d'écriture (écran)*	*Exemple de lecture (clavier)*
unsigned char "Character", caractère isolé	1 octet Cf. page	**unsigned char** MonChoix ;	**%c** `printf("Le choix est : %c", MonChoix) ;` `putch(Choix);` **%d - %x** *(Pour afficher la valeur du code ASCII)* `printf("Code ASCII :\n Valeur décimale : %d\n Valeur héxa. : %x", MonChoix, MonChoix) ;`	`MonChoix = getch() ;` **ou** `MonChoix = getche() ;`
unsigned char[] tableau d'octets chaîne de caractères	(Nb+1) octets *Nb* représente le nombre de caractères composant la chaîne.	**unsigned char** Nom[21] ; /* chaîne de 20 caractères */	**%s** `printf("M/Mme %s est né(e) à ...", Nom) ;` `printf("M/Mme %20s est né(e) à ...", Nom) ;` `puts(Nom) ;`	`gets (Nom) ;`

*Cf. Exemples de formatage (utilisation de printf et sprintf) page 15

	Tableaux à 1 dimension	*Tableaux à 2 dimensions*
Définition des équivalences Définition des dimensions du tableau (facultatif).	`#define TAILLE 15`	`#define NB_BLOCS 5` `#define NB_ELEM 12`
Déclaration et initialisation Exemples d'initialisation lors de la déclaration.	`float TabReel[15] ;` `float TabReel[TAILLE] ;` `/* Tableau de 15 réels. */` `float TabReel[15]={1.5, 2.5, 3} ;` `/* ... est complété par des 0.0*/`	`int MonTab[5][12] ;` `int MonTab[NB_BLOCS][NB_ELEM] ;` `/* Tableau à 2 dimensions. :` `5 blocs de 12 entiers. */` `float TabData[2][3] =` `{{2.1, 3.5, 5.2},` `{5.3, 6.4, 3.8}} ;`
Affectation	`TabReel[0] = 5.5 ;` `/* 5.5 est affecté au premier élément de TabReel. */`	`MonTab[2][9] = 12 ;` `/* 12 est affecté au 10ème élément du 3ème bloc de MonTab */`
Lecture au clavier	`gets(Temp) ;` `TabReel[14] = atof(Temp) ;` `/* Lecture de données au clavier et affectation au dernier élément de TabReel. */`	`gets(Temp) ;` `MonTab[0][0] = atoi(Temp) ;` `/* Lecture de données au clavier et affectation... */`
Écriture à l'écran	`printf("Premier et dernier élément du tableau : %.2f - %.2f\n", TabReel[0], TabReel[TAILLE-1]) ;`	`printf("Bloc n°%d - Element n°%d : %f\n", i, j, TabData[i][j]) ;` `/* i et j sont des variables de type int. */`
Prototypes de fonction (passage de tableaux par adresse)	`void InitTab(float Tableau[]) ;` `void AffichTab(const float TabR[]) ;` `float RechercheMax(const float TableauDonnees[]) ;`	`void AfficherTab(const int Tableau[][NB_ELEM]) ;` `float CalculMoy(const float Treels[], int NumLigne) ;`
Appels de fonction (passage de tableaux par adresse)	`InitTab(TabReel) ;` `AffichTab(TabReel) ;` `Maximum = RechercheMax(TabReel) ;` `/* Maximum est une variable de type float. */`	`AfficherTab(MonTab) ;` `Moyenne = CalculMoy(TabData, 1) ;` `/* Moyenne est une variable de type float. */`

CHAÎNES DE CARACTÈRES (ET TABLEAUX DE CHAÎNES)

	Chaînes de caractères	*Tableaux de chaînes*
Définition des équivalences Définition des dimensions du tableau (facultatif).	`#define LONGUEUR 20+1`	`#define NB_CHAINES 5` `#define NB_CAR 30+1`
Déclaration et initialisation Exemples d'initialisation lors de la déclaration.	`unsigned char Nom[21] ;` `unsigned char Nom[LONGUEUR] ;` `/* Chaîne de 20 caractères. */` `unsigned char Nom[21] = "Ritchie" ;` `unsigned char Nom[] = "Kernighan & Ritchie" ;` `unsigned char Nom[21] = {'R','i','t','c','h','i','e','\0'} ;`	`unsigned char MonCarnet[5][31] ;` `unsigned char MonCarnet[NB_CHAINES][NB_CAR] ;` `/* Tableau de 5 chaînes de 30 caractères. */` `unsigned char MonCarnet[][NB_CAR] = {"Kernighan", "Ritchie", "Meyer", "Hejlsberg", "Stroustrup" } ;`
Affectation	`strcpy(Nom, "Ritchie") ;` `/* Affectation à une chaîne de caractères. */`	`strcpy(MonCarnet[4], "Kernighan") ;`
Lecture au clavier	`gets(Nom) ;` `/* Lecture d'une chaîne de caractères. */`	`gets(MonCarnet[0]) ;` `/* Lecture d'une chaîne et stockage dans un tableau de chaînes. */`
Écriture à l'écran	`puts(Nom) ;` `printf("L'initiale de %s est %c", Nom, Nom[0]) ;`	`printf("L'initiale de %s est %c", MonCarnet[4], MonCarnet[4][0]) ;`
Prototypes de fonction (passage de tableaux par adresse)	`void SaisieChaine(unsigned char Chaine[], int Longueur) ;` `int LongueurChaine(const unsigned char Chaine[]) ;`	`void TrierCarnet(unsigned char TabChaine[][31], int TailleTab) ;` `void AffichCarnet(const unsigned char TabCh[][Nb_CAR], int NbCh) ;`
Appels de fonction (passage de tableaux par adresse)	`SaisieChaine(Nom, 20) ;` `Lng = LongueurChaine(Nom) ;` `/* Lng est une variable de type int. */`	`TrierCarnet(MonCarnet, 5) ;` `AffichCarnet(MonCarnet, NB_CHAINES);`

LES FONCTIONS STANDARDS

AVERTISSEMENT

Pour certaines fonctions, deux prototypes sont cités : *celui en italique correspond à l'utilisation habituelle de la fonction*, l'autre correspond à la définition réelle de la fonction

Fonctions mathématiques

`int abs(int x);` `double fabs(double x);`	**<math.h>** Retourne la valeur absolue du nombre passé en paramètre.
`double sin(double x);` `double cos(double x);` `double tan(double x);`	Retourne le sinus, le cosinus, la tangente de l'angle spécifié en radians.
`double ceil(double x);` `double floor(double x);`	Retourne l'arrondi par excès/défaut du nombre passé en paramètre
`double exp(double x);` `double log(double x);` `double log10(double x);` `double pow(double x, double y);` `double pow10(int p);` `double sqrt(double x);`	Calcule respectivement e^x, ln(x), $Log_{10}(x)$, x^y, 10^p, $\sqrt{x}$
`void randomize(void);` `int random(int num) ;`	**<stdlib.h> et <time.h>** `random` renvoie un nombre aléatoire compris entre 0 et (`num1`). Le générateur de nombre aléatoire doit être initialisé par `randomize`.

Conversion de type, lecture des données

`double atof(const char s[]);` `double atof(const char *s);` *`int atoi(const char s[]);`* `int atoi(const char *s);` *`long atol(const char s[]);`* `long atol(const char *s);`	**<stdlib.h>** Convertit la chaîne désignée par `s` en réel (double), entier ou entier de grande capacité.
`char getch(void);` `int getch(void);` *`char getche(void);`* `int getche(void);`	**<conio.h>** Attend la frappe d'un caractère et retourne celui-ci (avec affichage si on utilise `getche`)
`void gets(char s[]);` `char *gets(char *s);`	**<stdio.h>** Modifie une chaîne de caractères : lecture d'une suite de caractères (saisie au clavier)

Gestion d'écran texte, affichage des données

`void clrscr(void);` `void clreol(void);`	**<conio.h>** Efface la totalité de l'écran Efface la fin de la ligne à partir de la position du curseur
`void gotoxy(int x, int y);` `int wherex(void);` `int wherey(void);`	**<conio.h>** Déplace le curseur à la colonne `x` (1 à 80), et à la ligne `y` (1 à 25) Retourne le numéro de colonne, de ligne de la position du curseur.
`void putch(char c);` `int putch(int c);`	**<conio.h>** Affiche le caractère `c`.
`void putchar(char c);` `int putchar(int c);` *`void puts(const char s[]);`* `int puts(const char *s);` *`void printf("Texte, Chaîne formatée avec %..." [, liste de variables]);`* `int printf(const char *format[, argument, ...]);`	**<stdio.h>** Affiche le caractère `c`. Affiche la chaîne `s` suivi d'un saut de ligne. Permet l'affichage de tout type de variable (cf. ci-dessous).

Exemples de formatage (utilisation de `printf` et `sprintf`)

%d : notation décimale

%u : notation décimale non signée

%x, %X : notation hexadécimale (en minuscule, en majuscule)

%c : afficher le caractère correspondant (cf. codage ASCII)

%4d : afficher un entier sur 4 caractères au minimum, cadré à droite

%06d : afficher un entier sur 6 caractères au minimum, cadré à droite, complété par des 0 à gauche

%-4d : afficher un entier sur 4 caractères au minimum, cadré à gauche

%f : notation normale des réels

%e : notation scientifique des réels

%6f : afficher un réel sur 6 caractères au minimum, cadré à droite

%.2f : afficher un réel avec 2 chiffres significatifs après le séparateur décimal, cadré à droite

%+f : afficher un réel avec le signe ('+' ou '-').

%-+8.3f : afficher un réel avec le signe, sur 8 caractères avec 3 chiffres après le point décimal, cadré à gauche.

%20s : afficher une chaîne sur 20 caractères, cadrée à droite.

%-20s : afficher une chaîne sur 20 caractères, cadrée à gauche.

Manipulation des caractères et des chaînes de caractères

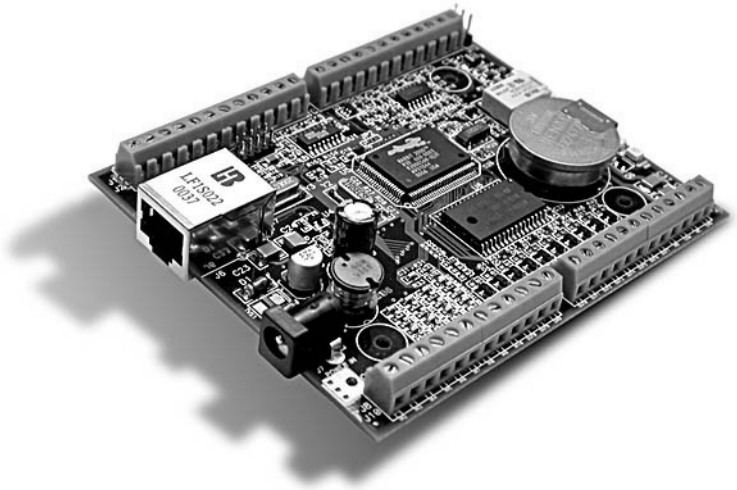
`void strcpy(char dest[], const char src[]);` `char *strcpy(char *dest, const char *src);` `void strncpy(char dest[], const char src[], int maxlen);` `char *strncpy(char *dest, const char *src, size_t maxlen);`	**<string.h>** Recopie le contenu de la chaîne `src` dans la chaîne `dest`. `strncpy` limite la copie à `maxlen` caractères.
`void strcat(char dest[], const char src[]) ;` `char *strcat(char *dest, const char *src);`	Ajoute le contenu de la chaîne `src` à la fin de la chaîne `dest` (concaténation).
`int strlen(const char s[]) ;` `size_t strlen(const char *s);`	Retourne la longueur de la chaîne `s`.
`int strcmp(const char s1[], const char s2[]) ;` `int strcmp(const char *s1, const char *s2);` *`int strncmp(const char s1[], const char s2[], int maxlen) ;`* `int strncmp(const char *s1, const char *s2, size_t maxlen);`	Effectuent une comparaison des deux chaînes `s1` et `s2`. `strncmp` limite la comparaison à `maxlen` caractères. Retourne 0 si les chaînes sont identiques, < 0 si `s1` doit être classé (selon l'ordre du code ASCII) avant `s2` et > 0 dans le cas contraire.
`void strlwr(char s[]);` `char *strlwr(char *s);` *`void strupr(char s[]);`* `char *strupr(char *s);`	Mise en minuscules (ou en majuscules) des caractères alphabétiques de la chaîne.
`void sprintf(char buffer[], "Texte, Chaîne formatée avec %..." [, liste de variables]);` `int sprintf(char *buffer, const char *format[, argument, ...]);`	**<stdio.h>** Fonctionnement identique à `printf`. La chaîne formatée n'est pas affichée à l'écran mais est stockée dans la chaîne `Buffer`.
`char *strchr(const char *s, int c);` `char *strrchr(const char *s, int c);`	**<string.h>** Recherchent le caractère `c` au sein de la chaîne `s` et retournent un pointeur sur le caractère trouvé. Si le caractère n'a pas été trouvé la valeur retournée est 0. `strchr` recherche la première occurrence, `strrchr` recherche la dernière.
`char *strstr(const char *s1, const char *s2) ;`	Recherche la sous-chaîne `s2` au sein de `s1` et retourne un pointeur sur la première chaîne `s2` trouvée dans `s1`. Si `s2` n'a pas été trouvée la valeur retournée est 0.
`void strrev(char s[]);` `char *strrev(char *s);`	Permute tous les caractères de la chaîne `s` afin d'en inverser l'ordre ; seul le caractère de fin de chaîne n'est pas concerné.

char tolower(char ch); int tolower(int ch); *char toupper(char ch);* int toupper(int ch);	**<ctype.h>** Convertit un caractère en minuscule. Convertit un caractère en majuscule.
int islower(char c); int islower(int c); int isupper(int c); int isalpha(int c); int isdigit(int c); int isxdigit(int c); int iscntrl(int c); int ispunct(int c); int isspace(int c); int isprint(int c); int isascii(int c);	**<ctype.h>** Cette famille de fonction sert à tester un caractère (test par rapport au code ASCII) Par exemple, `islower` retourne `VRAI` si `c` est une minuscule et `FAUX` (0) dans le cas contraire. `isalpha` test les lettres (a-z et A-Z) `isdigit` test les chiffres (0-9) `isxdigit` test les chiffres en base hexadécimale (0-9, a-f, A-F)

Divers

void delay(unsigned milliseconds); void sleep(unsigned seconds);	**<dos.h>** Suspend l'exécution du programme pendant un intervalle de temps (en milli-secondes, en secondes)
int kbhit(void);	**<conio.h>** Teste une éventuelle frappe de touche. Permet de faire une boucle d'attente avec *while (!kbhit());*
void textmode(int newmode);	**<conio.h>** Sélectionne un mode texte spécifique. Le mode choisi peut être spécifié dans l'argument `newmode` grâce à une équivalence symbolique (définie dans CONIO.H). Par exemple : *textmode(BW80) ;* permet de passer en mode 80 colonnes monochrome (noir et blanc)

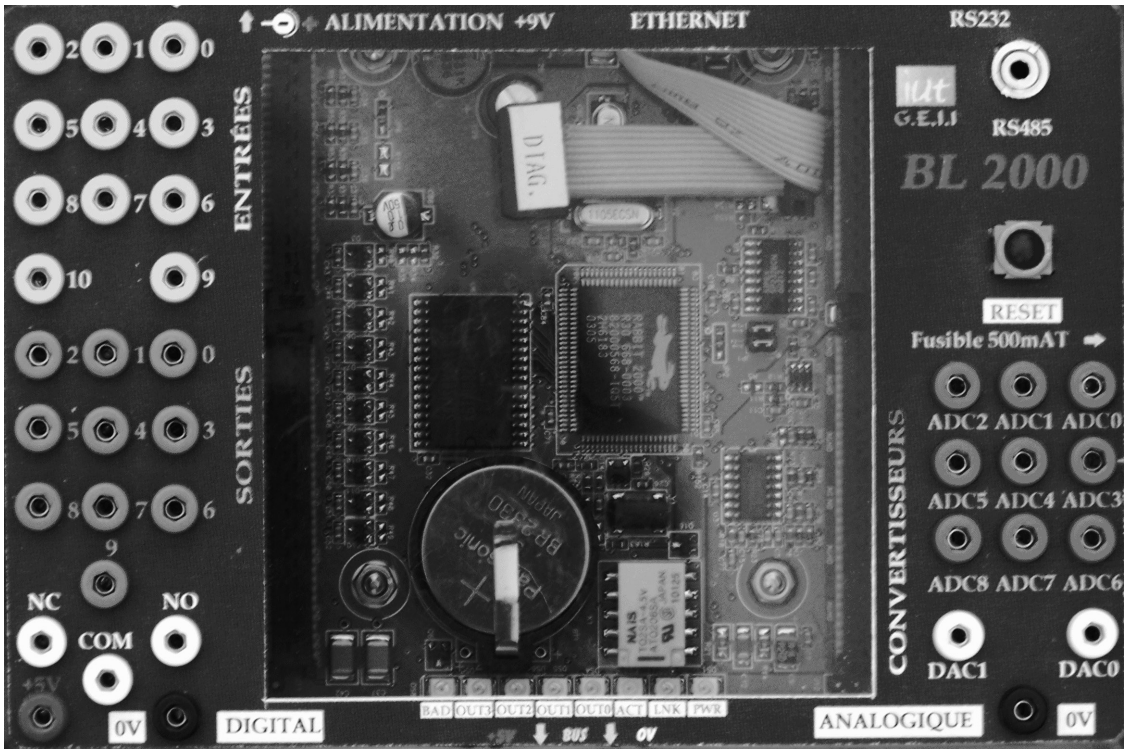
LA CARTE BL2000



La carte BL2000 est idéale pour le contrôle embarqué et les applications de surveillance. Sa taille réduite (87 x 105 x 21 mm) permet une intégration aisée dans les endroits les plus exigus.

Cette carte inclue notamment 11 entrées numériques, 10 sorties numériques (et 1 relais piloté en parallèle avec une sortie numérique), 9 entrées analogiques 12 bits, 2 sorties analogiques 12 bits et 1 port série RS232.

Le développement des programmes s'effectue en Dynamic C, atelier de développement très répandu dans l'industrie.



ORGANISATION D'UN PROGRAMME SOUS DYNAMIC C

```
#use "geii_bl2000.lib"

// Définition des équivalences
#define BPMA IN0
#define MOTEUR OUT2

// Prototype des fonctions
void AttendreAppuiBPMA(void) ;
void InitLiaisonSerie(void) ;
void ConfigInt(void) ;

// Déclaration des variables globales
int Compteur ;

void main(void)
{
    // Déclaration et initialisation des constantes locales
    const int TabAffichage[]={0x00, 0xA5, 0xB3, 0xFF} ;

    // Déclaration des variables locales
    char Octet ;
    int i ;

    // Initialisation des variables locales
    i = 0 ;

    // Initialisation de la carte BL2000 et de ses périphériques
    brdInit() ;
    InitLiaisonSerie() ;
    ConfigInt() ;

    // Boucle infinie
    while(1)
    {
         /* Traitement cyclique normal */
    }
}

// Définition des fonctions
void AttendreAppuiBPMA(void)
{
        /* ... */
}
/* ... */
```

Directives du préprocesseur :
seul fichier à inclure : `geii_bl2000.lib`,
définition des équivalences.

L'usage des variables globales doit être limité aux cas où il n'est pas possible d'utiliser les variables locales (mise en œuvre des interruptions).
Il faut **toujours** préférer l'usage des variables locales

Facultatif !

Facultatif !

Non obligatoire !
Un programme peut ne pas comporter de boucle infinie.

LES FONCTIONS D'ENTRÉE-SORTIE

Les fonctions d'E/S booléennes

Type	*Sens*	*Lignes*	*Ordre*	*Action*	*Fonction*
Bit	Sortie	OUT0 à OUT9	Écriture	Mise à 0	`void ClrBit(int AdrsBitOUT) ;`
				Mise à 1	`void SetBit(int AdrsBitOUT) ;`
				Complémenter	`void CplBit(int AdrsBitOUT) ;`
			Lecture	Lire état bit	`int TestBit(int AdrsBitOUT) ;`
	Entrée	IN0 à IN10	Lecture	Lire état bit	`int TestBit(int AdrsBitIN) ;`
Octet	Sortie	OUT0 à OUT7	Écriture		`void WritePortOUT7_0(char octet) ;`
		OUT0 à OUT9			`void WritePortOUT (int data) ;`
		OUT0 à OUT9	Lecture		`int ReadPortOUT (void) ;`
	Entrée	IN0 à IN10	Lecture		`int ReadPortIN (void) ;`

Les fonctions d'E/S analogiques

Sens	*Lignes*	*Ordre*	*Action*	*Fonction*
Sortie	DAC0 et DAC1	Écriture	Sort Tension	`void anaOutVolts (unsigned int Voie, float Tension) ;`
Entrée	ADC0 à ADC8	Lecture	Lit Tension	`float anaInVolts (unsigned int Voie) ;`

Dynamique des convertisseurs de tension

Convertisseurs analogiques – numériques

⇨ Entrées ADC0 à ADC3 : -10,24 V à +10,24 V

⇨ Entrées ADC4 à ADC8 : 0 à 48 V

Convertisseurs numériques – analogiques

⇨ Sorties DAC0 et DAC1 : 0 à 4,095 V

Taille des tampons (2^n-1)	`#define BINBUFSIZE 127 /* Tampon de réception (entrée) */` `#define BOUTBUFSIZE 127 /* Tampon d'émission (sortie) */`	
Ouverture	`int serBopen(long baud) ;`	*Paramètre :* ▪ *baud: vitesse de transmission* *Valeur retournée :* ▪ *1 : vitesse compatible* ▪ *0 : vitesse incompatible*
Taille données	`void serBdatabits(state) ;` `/* à effectuer après l'ouverture du port série */`	*Paramètre :* ▪ *state : PARAM_7BIT ou PARAM_8BIT* *Valeur retournée :* ▪ *aucune*
Parité	`void serBparity(int parity_mode) ;` `/* à effectuer après l'ouverture du port série */`	*Paramètre :* ▪ *parity_mode : PARAM_NOPARITY ou PARAM_OPARITY ou PARAM_EPARITY ou PARAM_2STOP* *Valeur retournée :* ▪ *aucune*
Fermeture	`void serBclose( ) ;`	
Vider tampons	`serBrdFlush( ) ; /* Tampon de réception (entrée) */` `serBwrFlush( ) ; /* Tampon d'émission (sortie) */`	

	Caractère ou octet	***Chaîne de caractères ou tableau d'octets***
Réception	`int serBgetc( ) ;` *Paramètre :* ▪ *aucun* *Valeur retournée :* ▪ *premier caractère disponible (codé sur 2 octets)* ▪ *si pas de caractère disponible : -1*	`int serBread(void *data, int length, unsigned long tmount) ;` *Paramètres:* ▪ *data: emplacement de la zone de données à stocker* ▪ *length : nombre d'octets à lire* ▪ *tmount : durée d'attente max en ms entre 2 réceptions* *Valeur retournée :* ▪ *nombre d'octets lus ou 0 si aucun caractère disponible*
État du tampon de Réception	`int serBrdUsed();` *Paramètre : aucun* *Valeur retournée : le nombre de caractères disponibles dans le tampon de réception*	
Émission	`int serBputc(char c) ;` *Paramètre :* ▪ *c : caractère à envoyer* *Valeur retournée :* ▪ *si caractère émis (succès) : 1* ▪ *sinon (échec) : 0*	`int serBwrite(void *data, int length);` *Paramètres:* ▪ *data: emplacement de la zone de données à envoyer* ▪ *length : nombre d'octets à envoyer* *Valeur retournée :* ▪ *nombre d'octets envoyés*

LES INTERRUPTIONS EXTERNES

Il existe 2 lignes d'entrées d'interruptions externes :

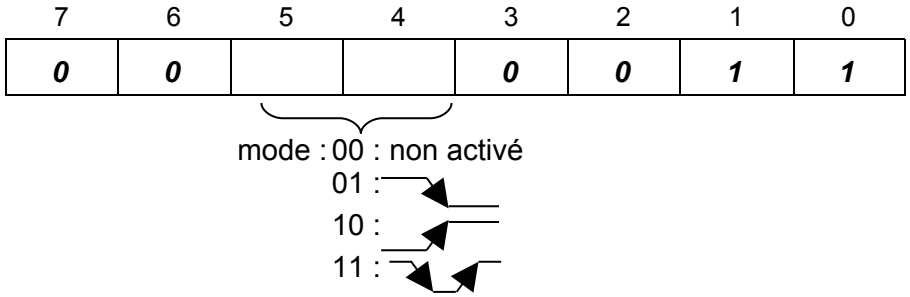
⇨ **IN2** à laquelle correspond l'interruption dite **INT0** (N° 0) et le registre **I0CR**,

⇨ **IN3** à laquelle correspond l'interruption dite **INT1** (N° 1) et le registre **I1CR**

Il est possible de choisir le mode de déclenchement de la ligne d'interruption (non actif, front montant, front descendant, front montant et descendant) recevant le signal qui va générer une demande d'interruption.

Pour cela on doit configurer le Registre de Contrôle de l'Interruption Externe concernée : I0CR pour INT0 et I1CR pour INT1.

Le registre IxCR



Étapes de mise en œuvre d'une interruption externe

1. Déclaration et définition de la fonction d'interruption

⇨ Déclarer le prototype de la fonction :

```
interrupt void Nom_de_la_fonction (void) ;
```

⇨ Définir la fonction (écrire la tâche réalisée par la fonction)

```
interrupt void Nom_de_la_fonction (void)
{


}
```

2. Installation

Mettre en place le vecteur d'interruption en indiquant au numéro d'interruption choisi (0 pour INT0 entrée IN2 et 1 pour INT1 entrée IN3) l'adresse de la routine d'interruption (son nom).

```
SetVectExtern3000 (Num_Interrupt, Nom_de_la_fonction) ;
```

3. Configuration du registre IxCR

Écrire dans le registre IxCR le mode de déclenchement et la priorité choisis :

```
WrPortI(IxCR, &IxCRShadow, octet) ;
          // octet représente la valeur que doit contenir IxCR
```

L'INTERRUPTION SYSTÈME 2KHZ

La carte BL2000 a la possibilité d'être périodiquement interrompue à la fréquence de 2 kHz. Toutes les 500 µs le microprocesseur Rabbit 2000 exécute la routine d'interruption dont l'adresse de début est mentionnée dans une table dite « table des vecteurs d'interruptions ».

L'adresse de la routine d'interruption système périodique (2 kHz) est rangée dans une zone de la table des vecteurs correspondant au numéro 0. Le constructeur a mentionné dans cette zone l'adresse d'une routine d'interruption qu'il exécute périodiquement.

Afin de pouvoir exécuter une autre routine que celle du constructeur, il faut mentionner dans cette zone l'adresse de la nouvelle routine. Cependant pour que le système fonctionne correctement, il est nécessaire de sauvegarder l'ancienne adresse avant d'installer la nouvelle afin d'exécuter la routine du constructeur à la fin de la nouvelle routine.

Si nécessaire on restitue l'ancien vecteur en quittant le programme.

Étapes de mise en œuvre d'une interruption système 2 kHz

Numéro d'interruption associé : 0

1. Déclarer une variable globale pouvant contenir l'adresse de l'ancien vecteur :

```
interrupt void (*p_Old)( ) ;
```

2. Déclarer et définir la fonction d'interruption à installer

⇨ Déclarer le prototype de la fonction :

```
interrupt void Nom_de_la_fonction (void) ;
```

⇨ Définir la fonction (écrire la tâche réalisée par la fonction)

```
interrupt void Nom_de_la_fonction (void)
{
       ||
  p_Old( ) ; // exécute l'ancienne int système pour éviter plantage
}
```

3. Lire puis sauver l'ancien vecteur :

```
p_Old = GetVectIntern ( Num_Interrupt ) ;
           // Num_Interrupt vaut 0 pour l'interruption système 2 kHz
```

4. Mettre en place le nouveau vecteur

```
SetVectIntern (Num_Interrupt, Nom_de_la_fonction) ;
           // Num_Interrupt vaut 0 pour l'interruption système 2 kHz
```

5. Désinstallation

Si nécessaire mettre en place en fin de programme l'ancienne interruption.

```
SetVectIntern (Num_Interrupt, p_Old) ;
           // Num_Interrupt vaut 0 pour l'interruption système 2 kHz
```