Langage C

Introduction

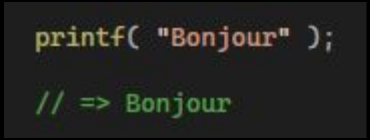
vs

Introduction

-Compilé
-Bas niveau
-Typé
-Gestion manuel de la mémoire

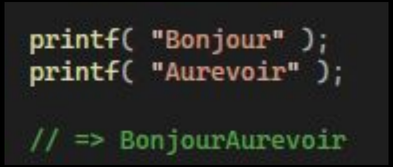
-Interpreté
-Haut niveau
-Dynamiquement typé
-Gestion automatique de la mémoire

Sortie



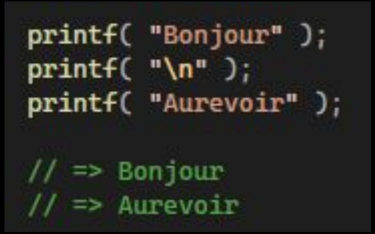
```c
printf( "Bonjour" );

// => Bonjour
```

Sortie

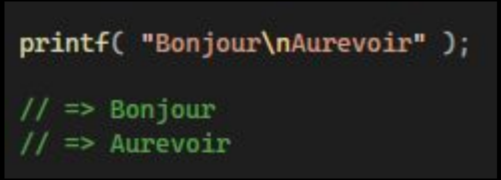


```
printf( "Bonjour" );
printf( "Aurevoir" );

// => BonjourAurevoir
```

Sortie

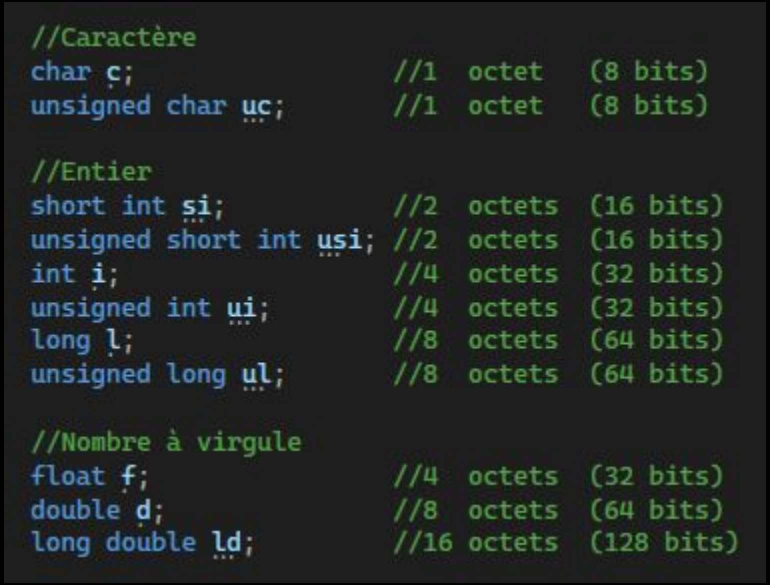


```c
printf( "Bonjour" );
printf( "\n" );
printf( "Aurevoir" );

// => Bonjour
// => Aurevoir
```



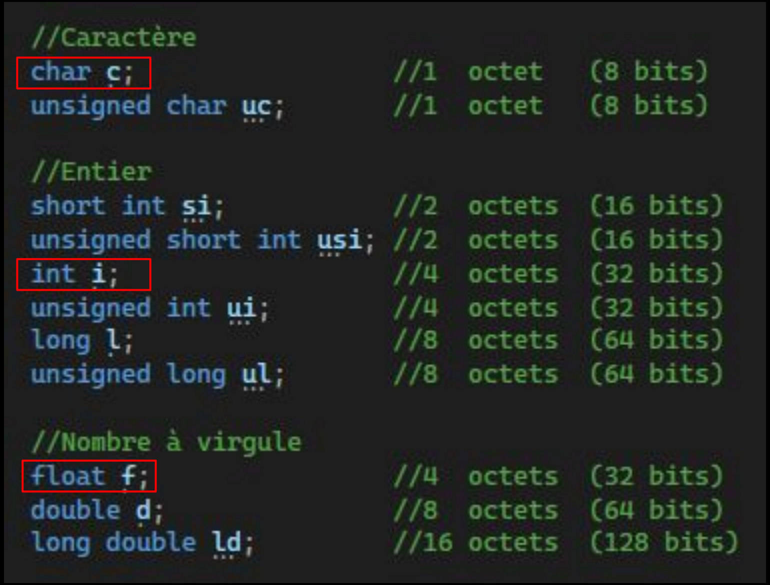
```c
printf( "Bonjour\nAurevoir" );

// => Bonjour
// => Aurevoir
```

Les types



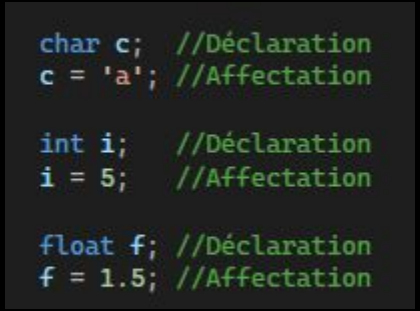
```
//Caractère
char c;                  //1  octet   (8 bits)
unsigned char uc;        //1  octet   (8 bits)

//Entier
short int si;            //2  octets  (16 bits)
unsigned short int usi;  //2  octets  (16 bits)
int i;                   //4  octets  (32 bits)
unsigned int ui;         //4  octets  (32 bits)
long l;                  //8  octets  (64 bits)
unsigned long ul;        //8  octets  (64 bits)

//Nombre à virgule
float f;                 //4  octets  (32 bits)
double d;                //8  octets  (64 bits)
long double ld;          //16 octets  (128 bits)
```

Les types

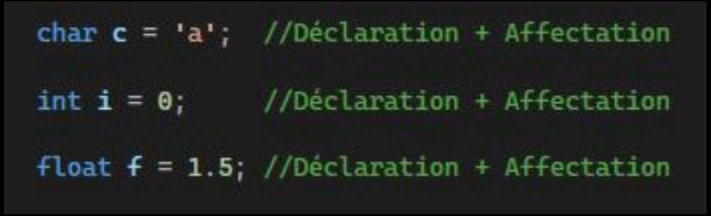


```
//Caractère
char c;                   //1  octet   (8 bits)
unsigned char uc;         //1  octet   (8 bits)

//Entier
short int si;             //2  octets  (16 bits)
unsigned short int usi;   //2  octets  (16 bits)
int i;                    //4  octets  (32 bits)
unsigned int ui;          //4  octets  (32 bits)
long l;                   //8  octets  (64 bits)
unsigned long ul;         //8  octets  (64 bits)

//Nombre à virgule
float f;                  //4  octets  (32 bits)
double d;                 //8  octets  (64 bits)
long double ld;           //16 octets  (128 bits)
```

Les types :
Déclaration et affectation

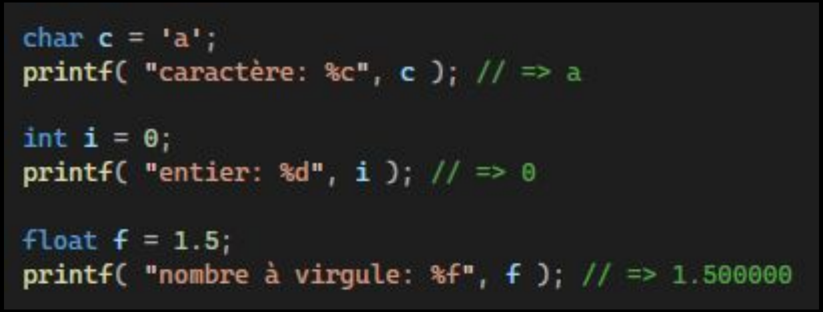


```c
char c;   //Déclaration
c = 'a'; //Affectation

int i;    //Déclaration
i = 5;    //Affectation

float f; //Déclaration
f = 1.5; //Affectation
```



```c
char c = 'a';   //Déclaration + Affectation

int i = 0;      //Déclaration + Affectation

float f = 1.5; //Déclaration + Affectation
```

Les types : Affichage



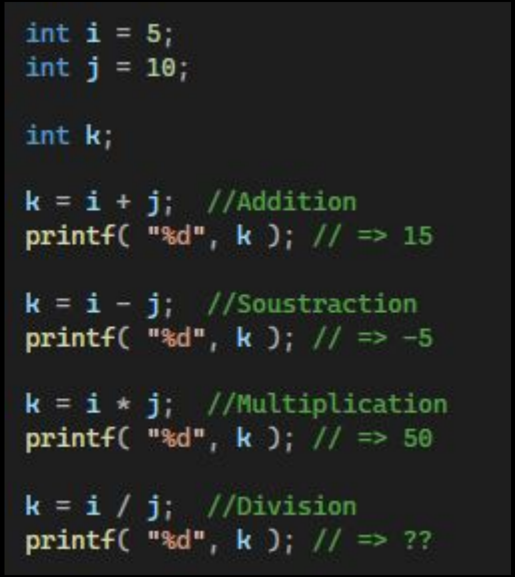
```
char c = 'a';
printf( "caractère: %c", c ); // => a

int i = 0;
printf( "entier: %d", i ); // => 0

float f = 1.5;
printf( "nombre à virgule: %f", f ); // => 1.500000
```

char : %c

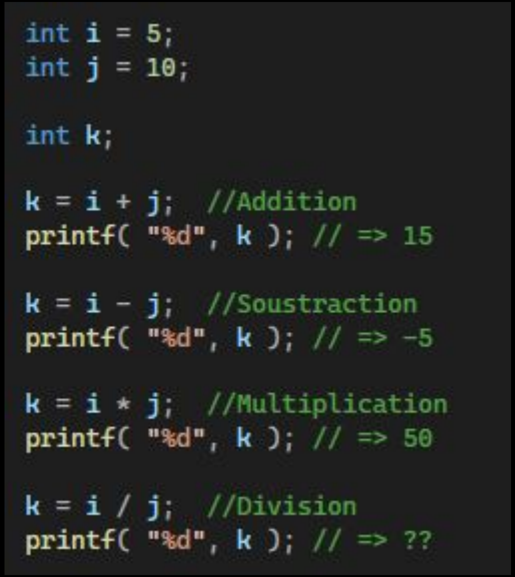
int : %d

float : %f

Les types :
Opérations sur int


```c
int i = 5;
int j = 10;

int k;

k = i + j;  //Addition
printf( "%d", k ); // => 15

k = i - j;  //Soustraction
printf( "%d", k ); // => -5

k = i * j;  //Multiplication
printf( "%d", k ); // => 50

k = i / j;  //Division
printf( "%d", k ); // => ??
```

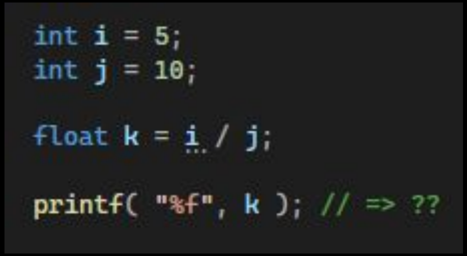
Les types :
Opérations sur int



```c
int i = 5;
int j = 10;

int k;

k = i + j;  //Addition
printf( "%d", k ); // => 15

k = i - j;  //Soustraction
printf( "%d", k ); // => -5

k = i * j;  //Multiplication
printf( "%d", k ); // => 50

k = i / j;  //Division
printf( "%d", k ); // => ??
```

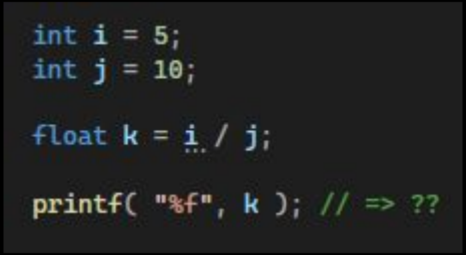
=> 0

Les types :
Opération sur float



```
int i = 5;
int j = 10;

float k = i / j;

printf( "%f", k ); // => ??
```

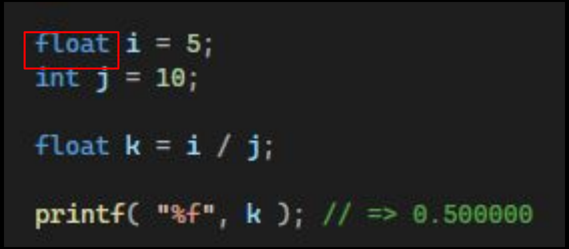
Les types :
Opération sur float



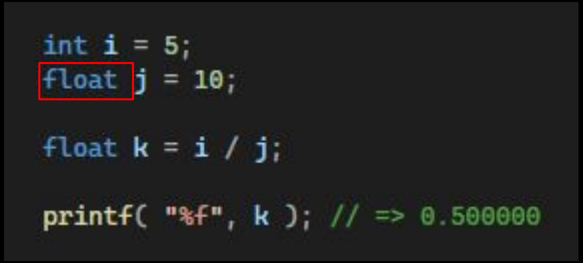
```
int i = 5;
int j = 10;

float k = i / j;

printf( "%f", k ); // => ??
```

=> 0.000000

Les types :
Opération sur float

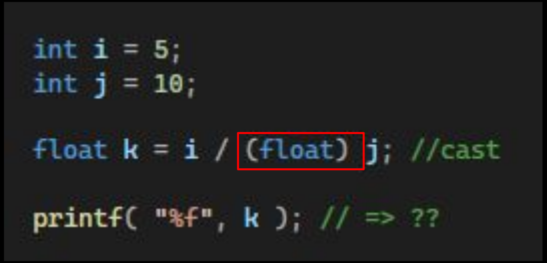


```c
float i = 5;
int j = 10;

float k = i / j;

printf( "%f", k ); // => 0.500000
```

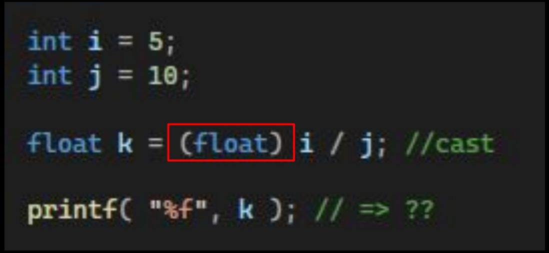


```c
int i = 5;
float j = 10;

float k = i / j;

printf( "%f", k ); // => 0.500000
```

Les types :
Opération sur float

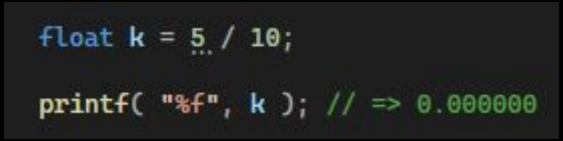


```c
int i = 5;
int j = 10;

float k = i / (float) j; //cast

printf( "%f", k ); // => ??
```

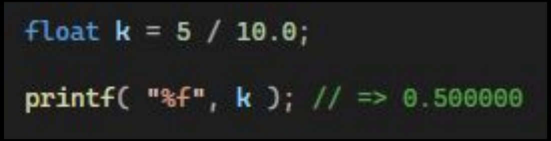


```c
int i = 5;
int j = 10;

float k = (float) i / j; //cast

printf( "%f", k ); // => ??
```

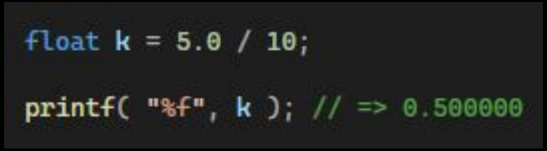
Les types :
Opération sur float



```c
float k = 5 / 10;

printf( "%f", k ); // => 0.000000
```

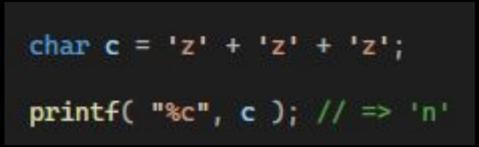


```c
float k = 5 / 10.0;

printf( "%f", k ); // => 0.500000
```



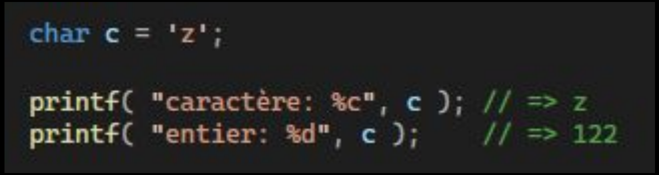
```c
float k = 5.0 / 10;

printf( "%f", k ); // => 0.500000
```

Les types : Opération sur char



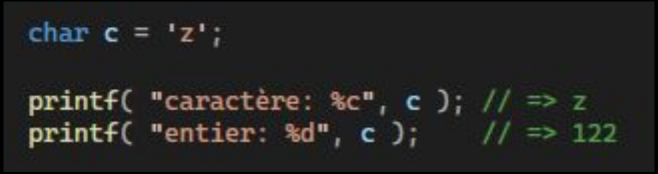
```c
char c = 'z' + 'z' + 'z';

printf( "%c", c ); // => 'n'
```

Les types :
Qu’est ce qu’un char ?

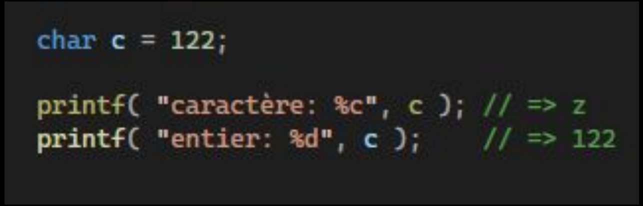


```
char c = 'z';

printf( "caractère: %c", c ); // => z
printf( "entier: %d", c );    // => 122
```

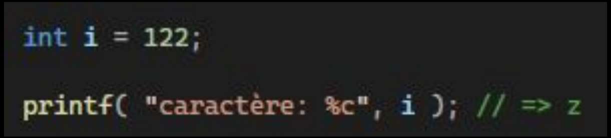
Les types :
Qu’est ce qu’un char ?



```c
char c = 'z';

printf( "caractère: %c", c ); // => z
printf( "entier: %d", c );    // => 122
```

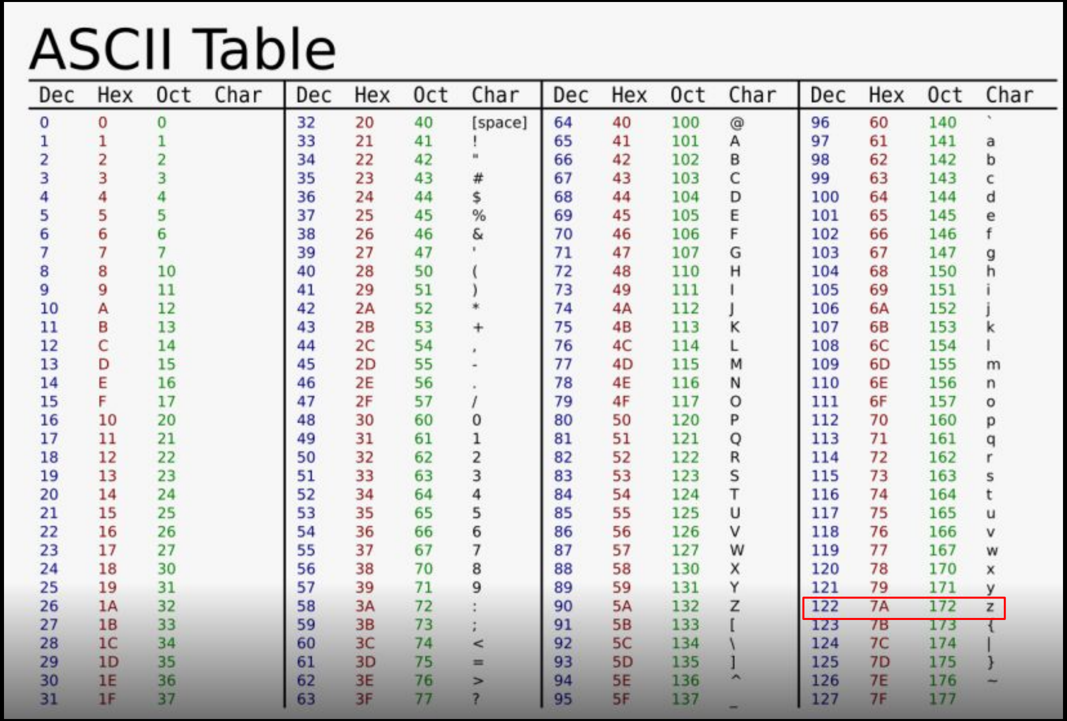


```c
char c = 122;

printf( "caractère: %c", c ); // => z
printf( "entier: %d", c );    // => 122
```



```c
int i = 122;

printf( "caractère: %c", i ); // => z
```

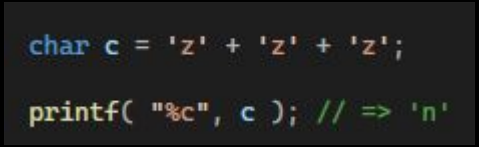
Les types:
Table ASCII



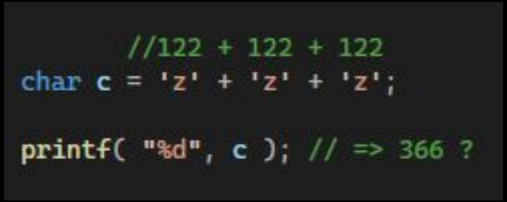
ASCII Table

Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char
0	0	0		32	20	40	[space]	64	40	100	@	96	60	140	`
1	1	1		33	21	41	!	65	41	101	A	97	61	141	a
2	2	2		34	22	42	"	66	42	102	B	98	62	142	b
3	3	3		35	23	43	#	67	43	103	C	99	63	143	c
4	4	4		36	24	44	$	68	44	104	D	100	64	144	d
5	5	5		37	25	45	%	69	45	105	E	101	65	145	e
6	6	6		38	26	46	&	70	46	106	F	102	66	146	f
7	7	7		39	27	47	'	71	47	107	G	103	67	147	g
8	8	10		40	28	50	(	72	48	110	H	104	68	150	h
9	9	11		41	29	51	)	73	49	111	I	105	69	151	i
10	A	12		42	2A	52	*	74	4A	112	J	106	6A	152	j
11	B	13		43	2B	53	+	75	4B	113	K	107	6B	153	k
12	C	14		44	2C	54	,	76	4C	114	L	108	6C	154	l
13	D	15		45	2D	55	-	77	4D	115	M	109	6D	155	m
14	E	16		46	2E	56	.	78	4E	116	N	110	6E	156	n
15	F	17		47	2F	57	/	79	4F	117	O	111	6F	157	o
16	10	20		48	30	60	0	80	50	120	P	112	70	160	p
17	11	21		49	31	61	1	81	51	121	Q	113	71	161	q
18	12	22		50	32	62	2	82	52	122	R	114	72	162	r
19	13	23		51	33	63	3	83	53	123	S	115	73	163	s
20	14	24		52	34	64	4	84	54	124	T	116	74	164	t
21	15	25		53	35	65	5	85	55	125	U	117	75	165	u
22	16	26		54	36	66	6	86	56	126	V	118	76	166	v
23	17	27		55	37	67	7	87	57	127	W	119	77	167	w
24	18	30		56	38	70	8	88	58	130	X	120	78	170	x
25	19	31		57	39	71	9	89	59	131	Y	121	79	171	y
26	1A	32		58	3A	72	:	90	5A	132	Z	122	7A	172	z
27	1B	33		59	3B	73	;	91	5B	133	[	123	7B	173	{
28	1C	34		60	3C	74	<	92	5C	134	\	124	7C	174	|
29	1D	35		61	3D	75	=	93	5D	135	]	125	7D	175	}
30	1E	36		62	3E	76	>	94	5E	136	^	126	7E	176	~
31	1F	37		63	3F	77	?	95	5F	137	_	127	7F	177	

Les types : Overflow

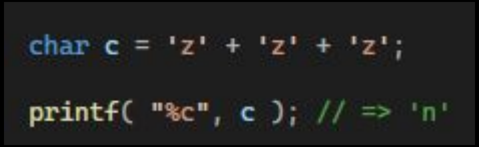


```c
char c = 'z' + 'z' + 'z';

printf( "%c", c ); // => 'n'
```

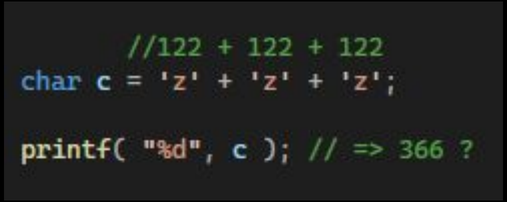


```c
        //122 + 122 + 122
char c = 'z' + 'z' + 'z';

printf( "%d", c ); // => 366 ?
```

Les types : Overflow



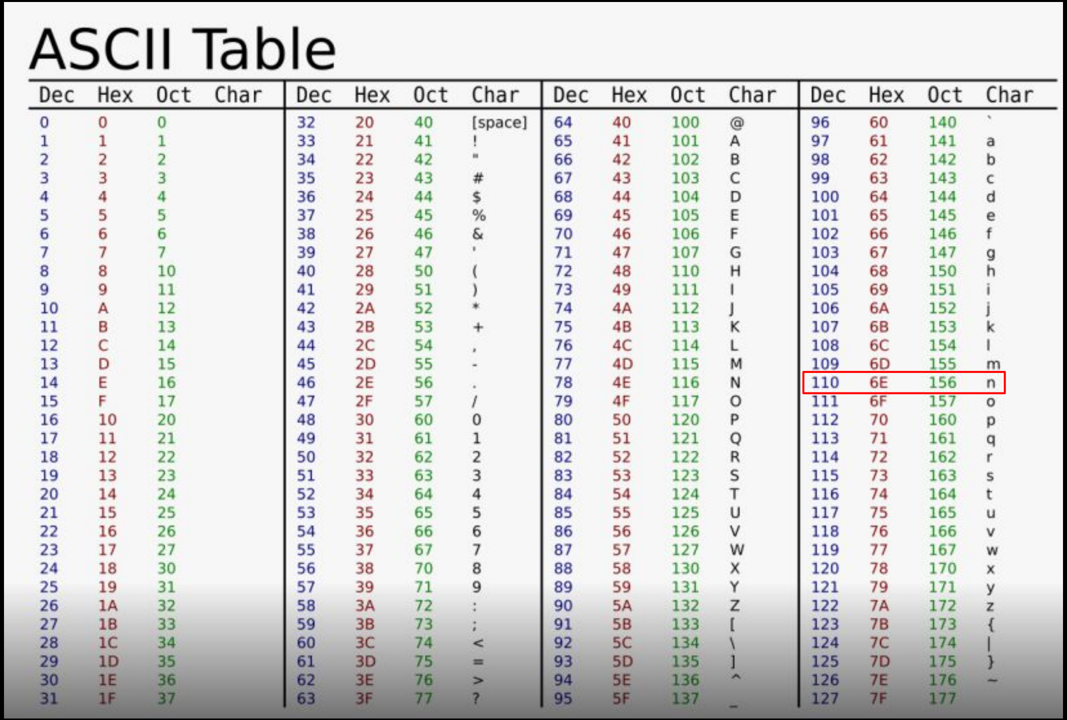
```c
char c = 'z' + 'z' + 'z';

printf( "%c", c ); // => 'n'
```



```c
        //122 + 122 + 122
char c = 'z' + 'z' + 'z';

printf( "%d", c ); // => 366 ?
```

=> 110

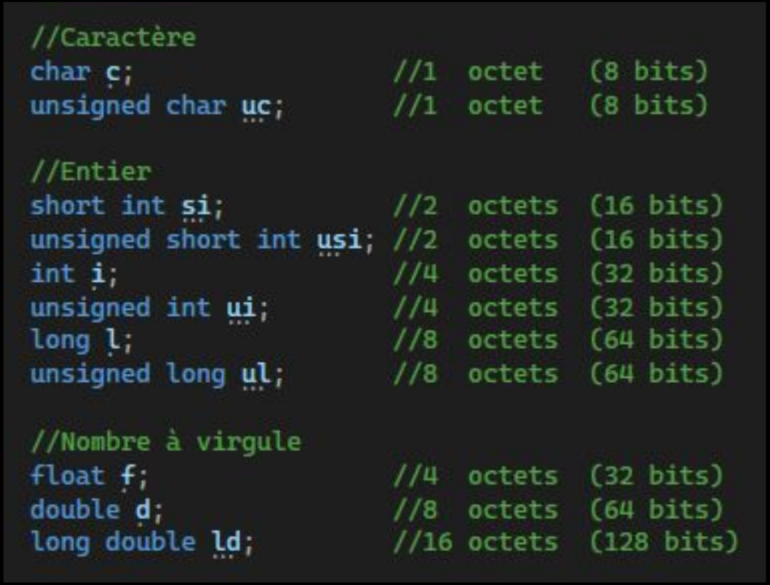
Les types : Overflow



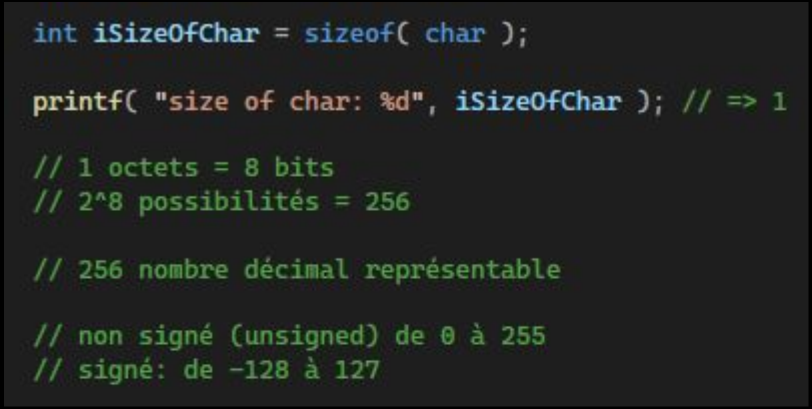
ASCII Table

Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char
0	0	0		32	20	40	[space]	64	40	100	@	96	60	140	`
1	1	1		33	21	41	!	65	41	101	A	97	61	141	a
2	2	2		34	22	42	"	66	42	102	B	98	62	142	b
3	3	3		35	23	43	#	67	43	103	C	99	63	143	c
4	4	4		36	24	44	$	68	44	104	D	100	64	144	d
5	5	5		37	25	45	%	69	45	105	E	101	65	145	e
6	6	6		38	26	46	&	70	46	106	F	102	66	146	f
7	7	7		39	27	47	'	71	47	107	G	103	67	147	g
8	8	10		40	28	50	(	72	48	110	H	104	68	150	h
9	9	11		41	29	51	)	73	49	111	I	105	69	151	i
10	A	12		42	2A	52	*	74	4A	112	J	106	6A	152	j
11	B	13		43	2B	53	+	75	4B	113	K	107	6B	153	k
12	C	14		44	2C	54	,	76	4C	114	L	108	6C	154	l
13	D	15		45	2D	55	-	77	4D	115	M	109	6D	155	m
14	E	16		46	2E	56	.	78	4E	116	N	110	6E	156	n
15	F	17		47	2F	57	/	79	4F	117	O	111	6F	157	o
16	10	20		48	30	60	0	80	50	120	P	112	70	160	p
17	11	21		49	31	61	1	81	51	121	Q	113	71	161	q
18	12	22		50	32	62	2	82	52	122	R	114	72	162	r
19	13	23		51	33	63	3	83	53	123	S	115	73	163	s
20	14	24		52	34	64	4	84	54	124	T	116	74	164	t
21	15	25		53	35	65	5	85	55	125	U	117	75	165	u
22	16	26		54	36	66	6	86	56	126	V	118	76	166	v
23	17	27		55	37	67	7	87	57	127	W	119	77	167	w
24	18	30		56	38	70	8	88	58	130	X	120	78	170	x
25	19	31		57	39	71	9	89	59	131	Y	121	79	171	y
26	1A	32		58	3A	72	:	90	5A	132	Z	122	7A	172	z
27	1B	33		59	3B	73	;	91	5B	133	[	123	7B	173	{
28	1C	34		60	3C	74	<	92	5C	134	\	124	7C	174	|
29	1D	35		61	3D	75	=	93	5D	135	]	125	7D	175	}
30	1E	36		62	3E	76	>	94	5E	136	^	126	7E	176	~
31	1F	37		63	3F	77	?	95	5F	137	_	127	7F	177	

Les types : Overflow

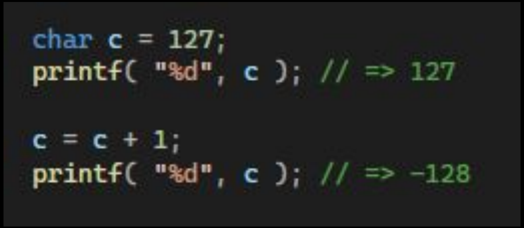


```c
//Caractère
char c;                  //1  octet   (8 bits)
unsigned char uc;        //1  octet   (8 bits)

//Entier
short int si;            //2  octets  (16 bits)
unsigned short int usi;  //2  octets  (16 bits)
int i;                   //4  octets  (32 bits)
unsigned int ui;         //4  octets  (32 bits)
long l;                  //8  octets  (64 bits)
unsigned long ul;        //8  octets  (64 bits)

//Nombre à virgule
float f;                 //4  octets  (32 bits)
double d;                //8  octets  (64 bits)
long double ld;          //16 octets  (128 bits)
```



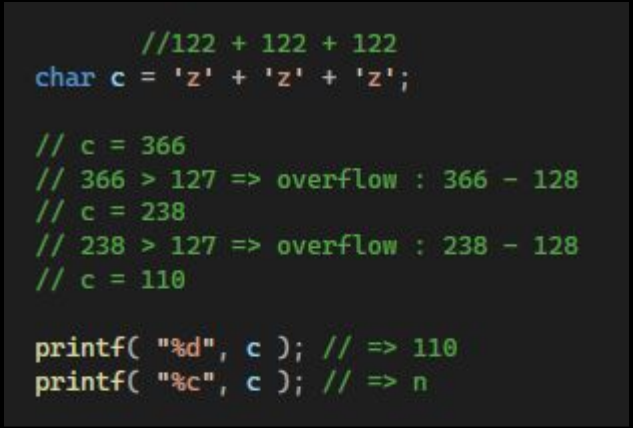
```c
int iSizeOfChar = sizeof( char );

printf( "size of char: %d", iSizeOfChar ); // => 1

// 1 octets = 8 bits
// 2^8 possibilités = 256

// 256 nombre décimal représentable

// non signé (unsigned) de 0 à 255
// signé: de -128 à 127
```

Les types : Overflow



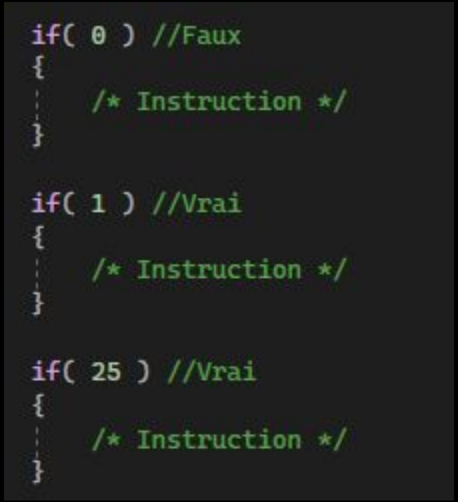
```
char c = 127;
printf( "%d", c ); // => 127

c = c + 1;
printf( "%d", c ); // => -128
```

Les types : Overflow



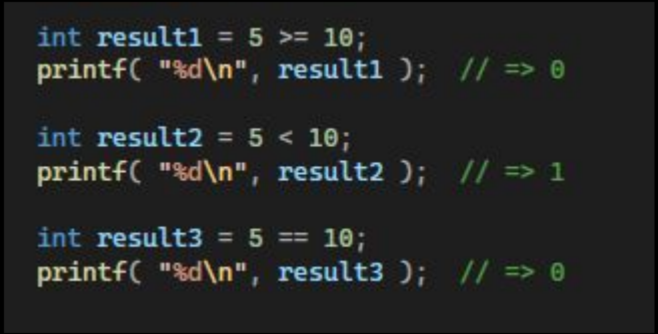
```
       //122 + 122 + 122
char c = 'z' + 'z' + 'z';

// c = 366
// 366 > 127 => overflow : 366 - 128
// c = 238
// 238 > 127 => overflow : 238 - 128
// c = 110

printf( "%d", c ); // => 110
printf( "%c", c ); // => n
```

Les conditions



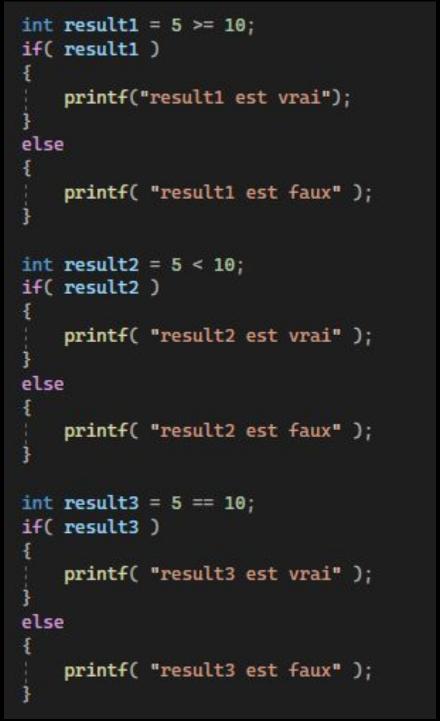
```
if( 0 ) //Faux
{
    /* Instruction */
}

if( 1 ) //Vrai
{
    /* Instruction */
}

if( 25 ) //Vrai
{
    /* Instruction */
}
```

Les conditions



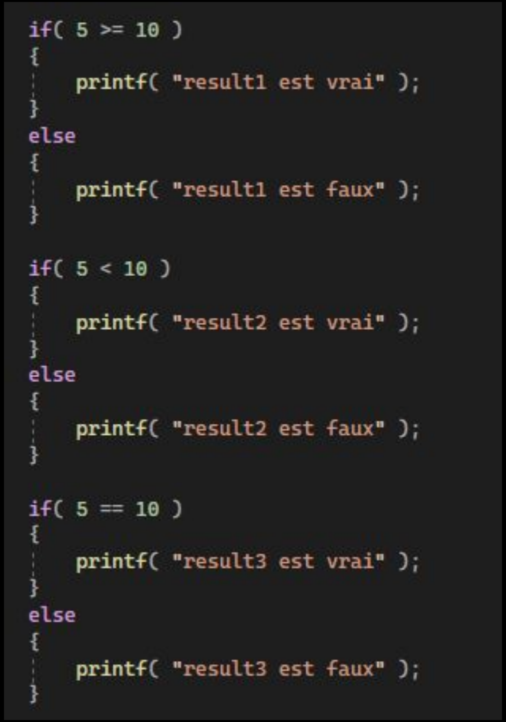
```
int result1 = 5 >= 10;
printf( "%d\n", result1 );  // => 0

int result2 = 5 < 10;
printf( "%d\n", result2 );  // => 1

int result3 = 5 == 10;
printf( "%d\n", result3 );  // => 0
```

Les conditions



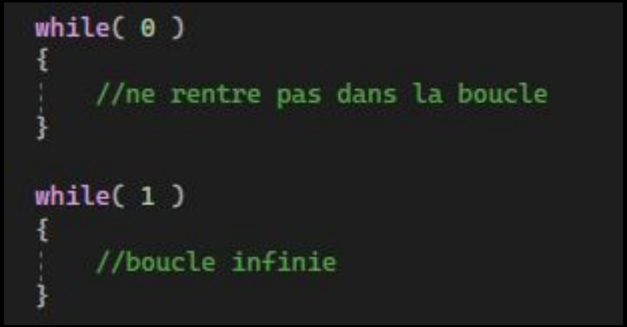
```c
int result1 = 5 >= 10;
if( result1 )
{
    printf("result1 est vrai");
}
else
{
    printf( "result1 est faux" );
}

int result2 = 5 < 10;
if( result2 )
{
    printf( "result2 est vrai" );
}
else
{
    printf( "result2 est faux" );
}

int result3 = 5 == 10;
if( result3 )
{
    printf( "result3 est vrai" );
}
else
{
    printf( "result3 est faux" );
}
```

Les conditions



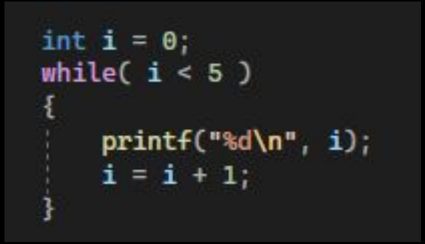
```c
if( 5 >= 10 )
{
    printf( "result1 est vrai" );
}
else
{
    printf( "result1 est faux" );
}

if( 5 < 10 )
{
    printf( "result2 est vrai" );
}
else
{
    printf( "result2 est faux" );
}

if( 5 == 10 )
{
    printf( "result3 est vrai" );
}
else
{
    printf( "result3 est faux" );
}
```

Les boucles

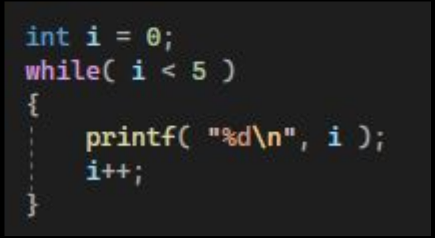


```
while( 0 )
{
    //ne rentre pas dans la boucle
}

while( 1 )
{
    //boucle infinie
}
```

Les boucles

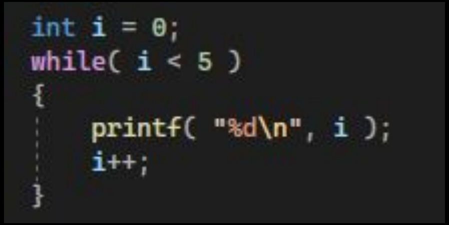


```c
int i = 0;
while( i < 5 )
{
    printf("%d\n", i);
    i = i + 1;
}
```

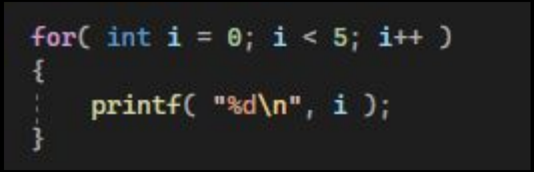


```c
int i = 0;
while( i < 5 )
{
    printf( "%d\n", i );
    i++;
}
```

Les boucles

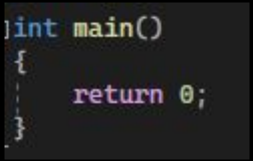


```c
int i = 0;
while( i < 5 )
{
    printf( "%d\n", i );
    i++;
}
```



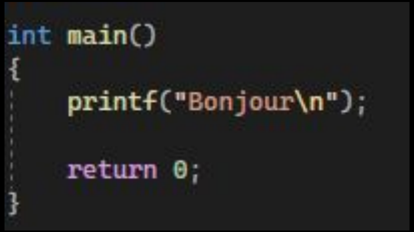
```c
for( int i = 0; i < 5; i++ )
{
    printf( "%d\n", i );
}
```

Les fonctions : main



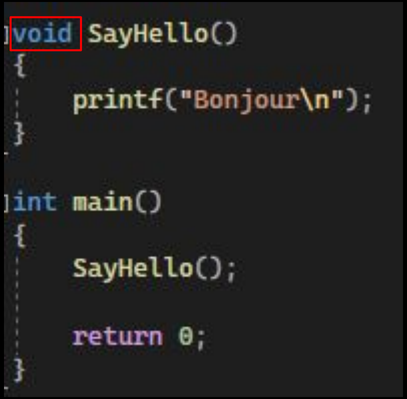
```
int main()
{
    return 0;
}
```

Les fonctions : main



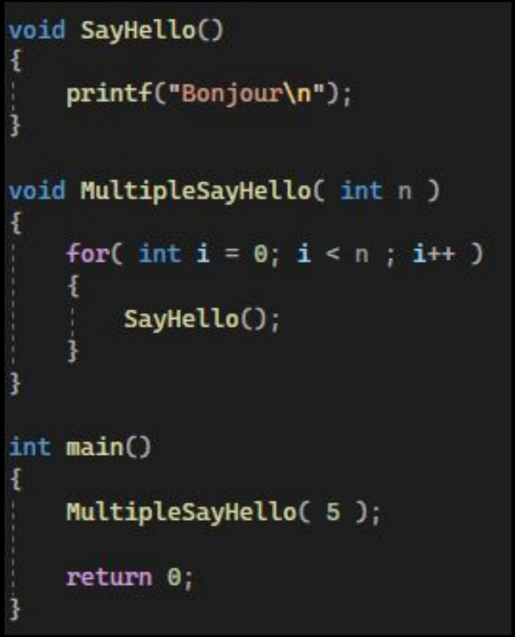
```c
int main()
{
    printf("Bonjour\n");

    return 0;
}
```

Les fonctions :
Sans retour



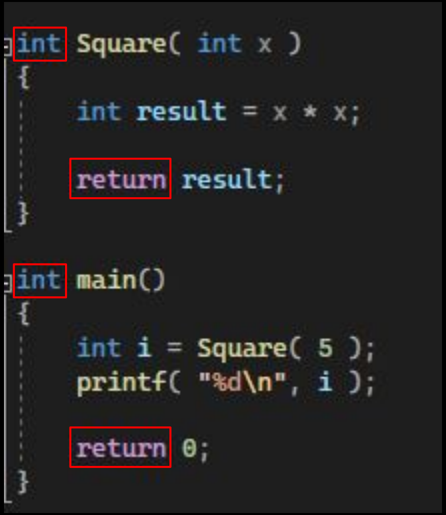
```c
void SayHello()
{
    printf("Bonjour\n");
}

int main()
{
    SayHello();

    return 0;
}
```

Les fonctions : Sans retour



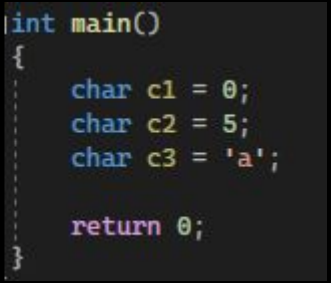
```c
void SayHello()
{
    printf("Bonjour\n");
}

void MultipleSayHello( int n )
{
    for( int i = 0; i < n ; i++ )
    {
        SayHello();
    }
}

int main()
{
    MultipleSayHello( 5 );

    return 0;
}
```

Les fonctions : Avec retour

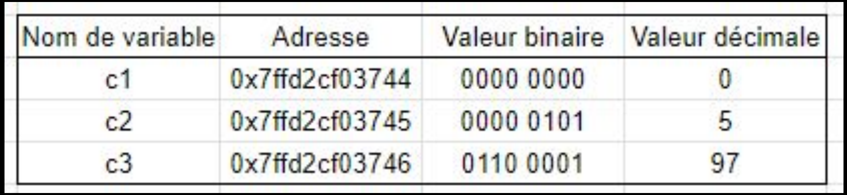


```
int Square( int x )
{
    int result = x * x;

    return result;
}

int main()
{
    int i = Square( 5 );
    printf( "%d\n", i );

    return 0;
}
```

Les adresses

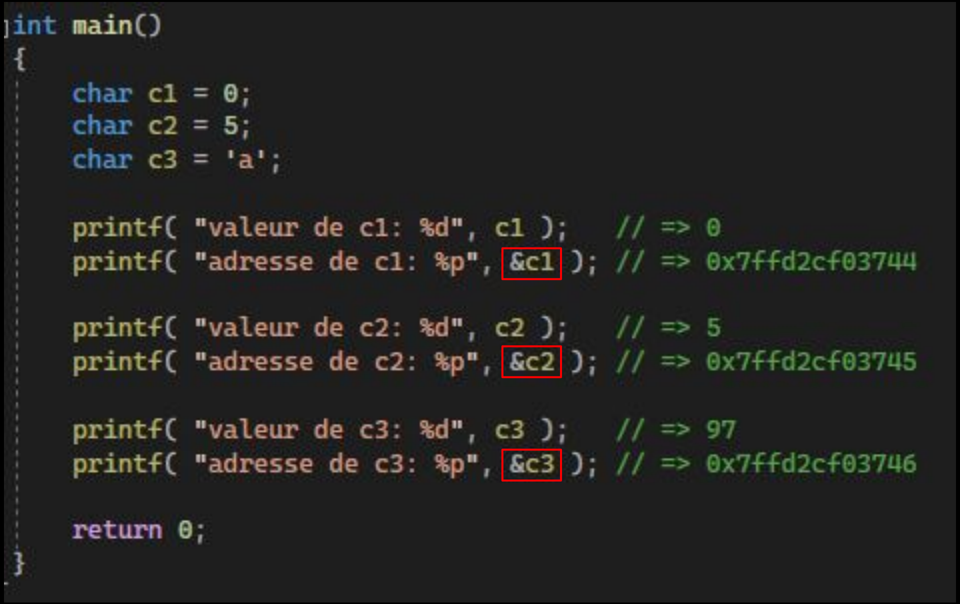


```
int main()
{
    char c1 = 0;
    char c2 = 5;
    char c3 = 'a';

    return 0;
}
```

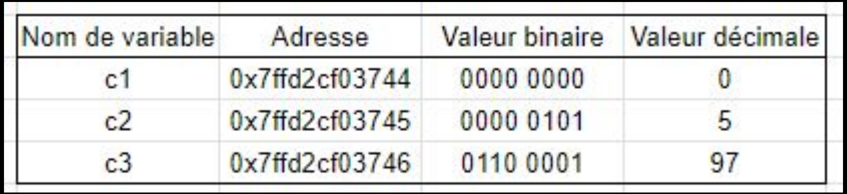


Nom de variable	Adresse	Valeur binaire	Valeur décimale
c1	0x7ffd2cf03744	0000 0000	0
c2	0x7ffd2cf03745	0000 0101	5
c3	0x7ffd2cf03746	0110 0001	97

Les adresses :
&

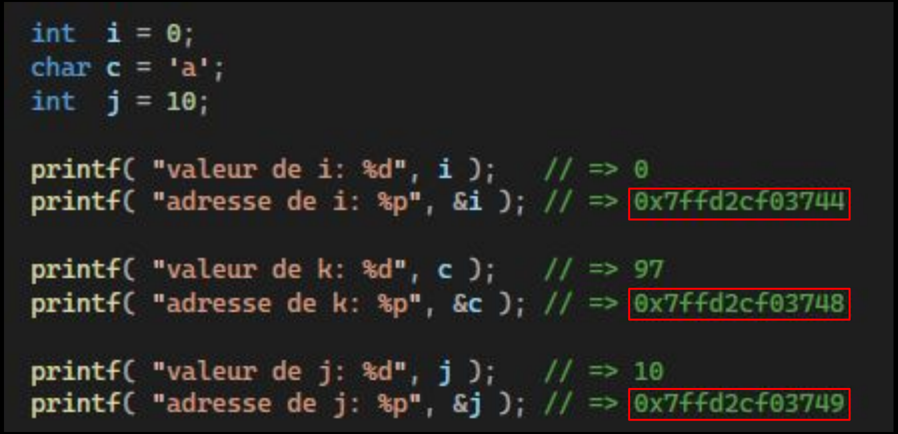


```c
int main()
{
    char c1 = 0;
    char c2 = 5;
    char c3 = 'a';

    printf( "valeur de c1: %d", c1 );   // => 0
    printf( "adresse de c1: %p", &c1 ); // => 0x7ffd2cf03744

    printf( "valeur de c2: %d", c2 );   // => 5
    printf( "adresse de c2: %p", &c2 ); // => 0x7ffd2cf03745

    printf( "valeur de c3: %d", c3 );   // => 97
    printf( "adresse de c3: %p", &c3 ); // => 0x7ffd2cf03746

    return 0;
}
```

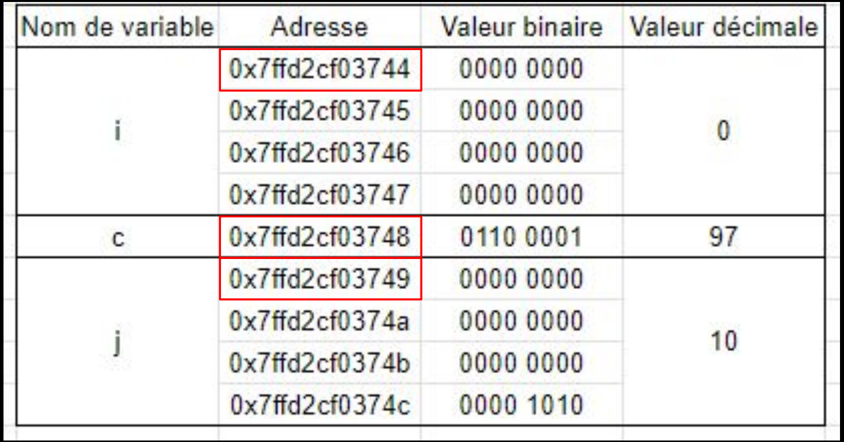


Nom de variable	Adresse	Valeur binaire	Valeur décimale
c1	0x7ffd2cf03744	0000 0000	0
c2	0x7ffd2cf03745	0000 0101	5
c3	0x7ffd2cf03746	0110 0001	97

Les adresses

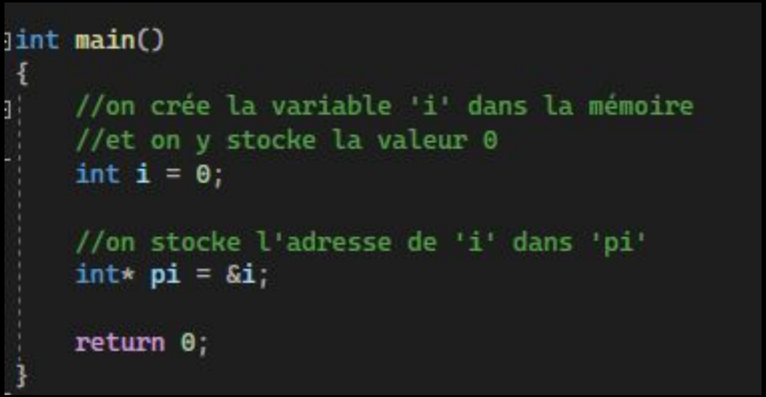


```
int  i = 0;
char c = 'a';
int  j = 10;

printf( "valeur de i: %d", i );   // => 0
printf( "adresse de i: %p", &i ); // => 0x7ffd2cf03744

printf( "valeur de k: %d", c );   // => 97
printf( "adresse de k: %p", &c ); // => 0x7ffd2cf03748

printf( "valeur de j: %d", j );   // => 10
printf( "adresse de j: %p", &j ); // => 0x7ffd2cf03749
```

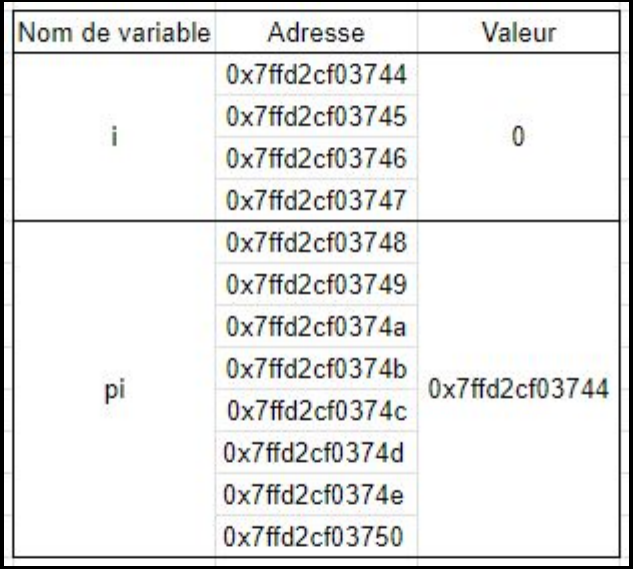


Nom de variable	Adresse	Valeur binaire	Valeur décimale
i	0x7ffd2cf03744	0000 0000	0
	0x7ffd2cf03745	0000 0000	
	0x7ffd2cf03746	0000 0000	
	0x7ffd2cf03747	0000 0000	
c	0x7ffd2cf03748	0110 0001	97
j	0x7ffd2cf03749	0000 0000	10
	0x7ffd2cf0374a	0000 0000	
	0x7ffd2cf0374b	0000 0000	
	0x7ffd2cf0374c	0000 1010	

Les pointeurs

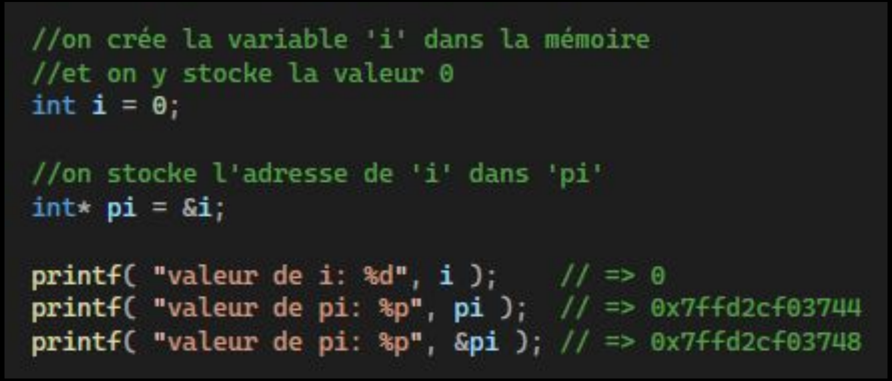


```c
int main()
{
    //on crée la variable 'i' dans la mémoire
    //et on y stocke la valeur 0
    int i = 0;

    //on stocke l'adresse de 'i' dans 'pi'
    int* pi = &i;

    return 0;
}
```

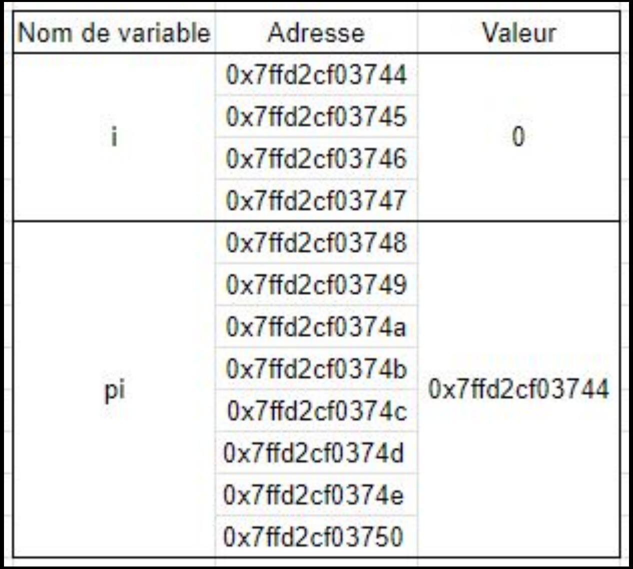


Nom de variable	Adresse	Valeur
i	0x7ffd2cf03744	0
	0x7ffd2cf03745	
	0x7ffd2cf03746	
	0x7ffd2cf03747	
pi	0x7ffd2cf03748	0x7ffd2cf03744
	0x7ffd2cf03749	
	0x7ffd2cf0374a	
	0x7ffd2cf0374b	
	0x7ffd2cf0374c	
	0x7ffd2cf0374d	
	0x7ffd2cf0374e	
	0x7ffd2cf03750	

Les pointeurs

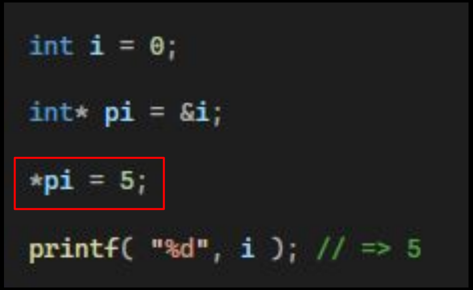


```c
//on crée la variable 'i' dans la mémoire
//et on y stocke la valeur 0
int i = 0;

//on stocke l'adresse de 'i' dans 'pi'
int* pi = &i;

printf( "valeur de i: %d", i );    // => 0
printf( "valeur de pi: %p", pi );  // => 0x7ffd2cf03744
printf( "valeur de pi: %p", &pi ); // => 0x7ffd2cf03748
```



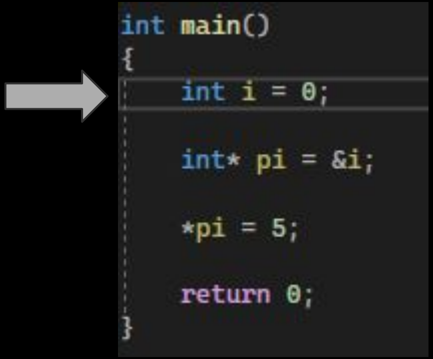
Nom de variable	Adresse	Valeur
i	0x7ffd2cf03744	0
	0x7ffd2cf03745	
	0x7ffd2cf03746	
	0x7ffd2cf03747	
pi	0x7ffd2cf03748	0x7ffd2cf03744
	0x7ffd2cf03749	
	0x7ffd2cf0374a	
	0x7ffd2cf0374b	
	0x7ffd2cf0374c	
	0x7ffd2cf0374d	
	0x7ffd2cf0374e	
	0x7ffd2cf03750	

Les pointeurs : déréférencement

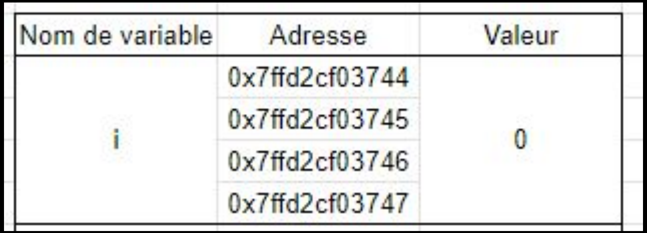


```c
int i = 0;

int* pi = &i;

*pi = 5;

printf( "%d", i ); // => 5
```

Les pointeurs : déréférencement

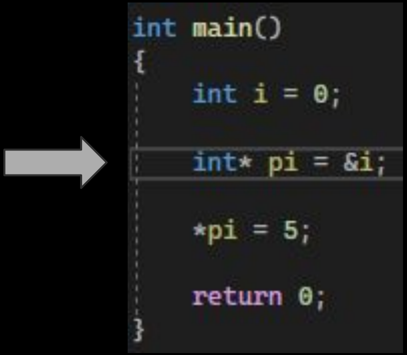


```
int main()
{
    int i = 0;

    int* pi = &i;

    *pi = 5;

    return 0;
}
```

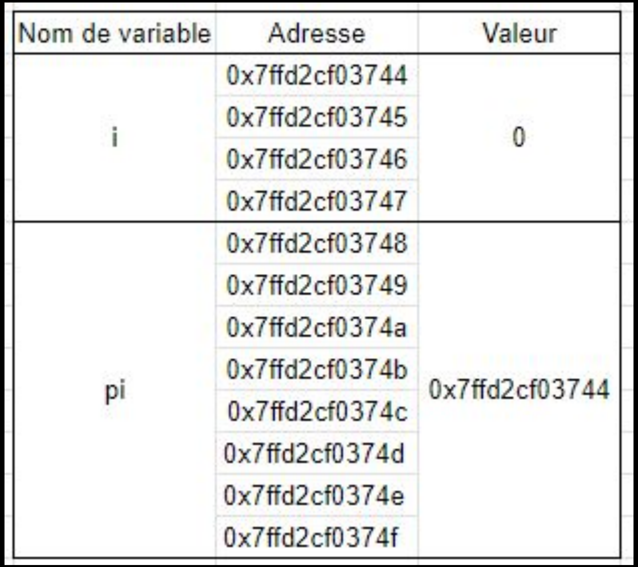


Nom de variable	Adresse	Valeur
i	0x7ffd2cf03744	0
	0x7ffd2cf03745	
	0x7ffd2cf03746	
	0x7ffd2cf03747	

Les pointeurs : déréférencement

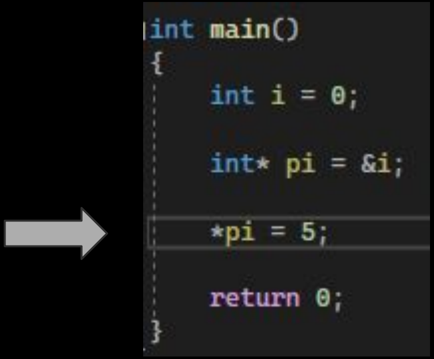


```
int main()
{
    int i = 0;

    int* pi = &i;

    *pi = 5;

    return 0;
}
```

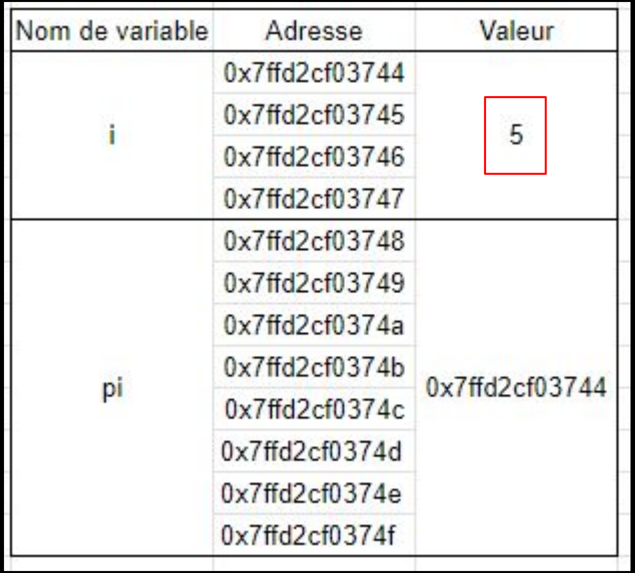


Nom de variable	Adresse	Valeur
i	0x7ffd2cf03744	0
	0x7ffd2cf03745	
	0x7ffd2cf03746	
	0x7ffd2cf03747	
pi	0x7ffd2cf03748	0x7ffd2cf03744
	0x7ffd2cf03749	
	0x7ffd2cf0374a	
	0x7ffd2cf0374b	
	0x7ffd2cf0374c	
	0x7ffd2cf0374d	
	0x7ffd2cf0374e	
	0x7ffd2cf0374f	

Les pointeurs : déréférencement

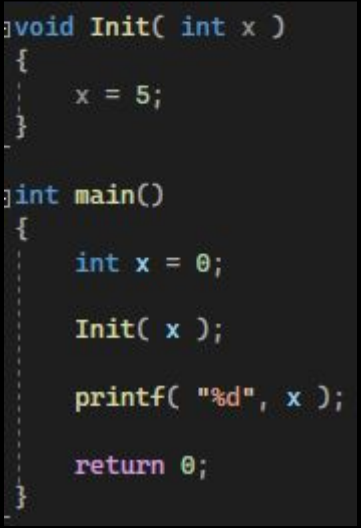


```
int main()
{
    int i = 0;

    int* pi = &i;

    *pi = 5;

    return 0;
}
```



Nom de variable	Adresse	Valeur
i	0x7ffd2cf03744	5
	0x7ffd2cf03745	
	0x7ffd2cf03746	
	0x7ffd2cf03747	
pi	0x7ffd2cf03748	0x7ffd2cf03744
	0x7ffd2cf03749	
	0x7ffd2cf0374a	
	0x7ffd2cf0374b	
	0x7ffd2cf0374c	
	0x7ffd2cf0374d	
	0x7ffd2cf0374e	
	0x7ffd2cf0374f	

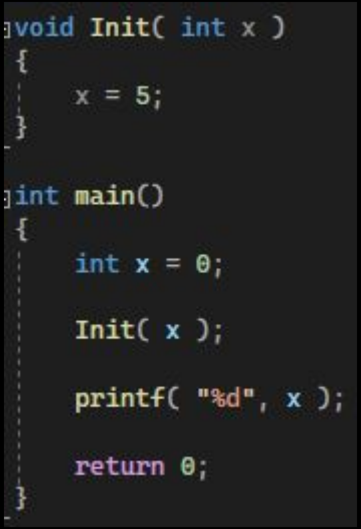
Les pointeurs



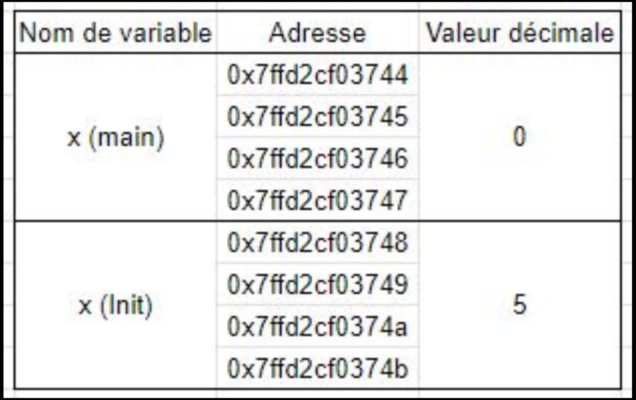
```c
void Init( int x )
{
    x = 5;
}

int main()
{
    int x = 0;

    Init( x );

    printf( "%d", x );

    return 0;
}
```

=> ??

Les pointeurs

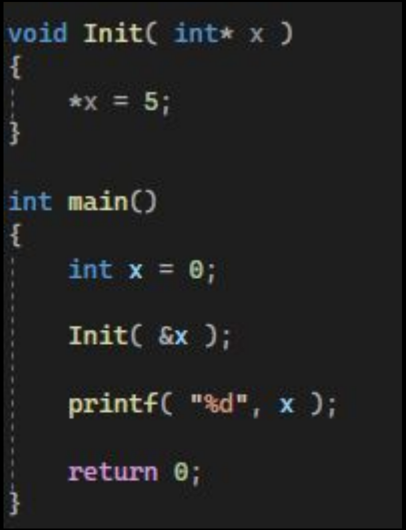


```c
void Init( int x )
{
    x = 5;
}

int main()
{
    int x = 0;

    Init( x );

    printf( "%d", x );

    return 0;
}
```

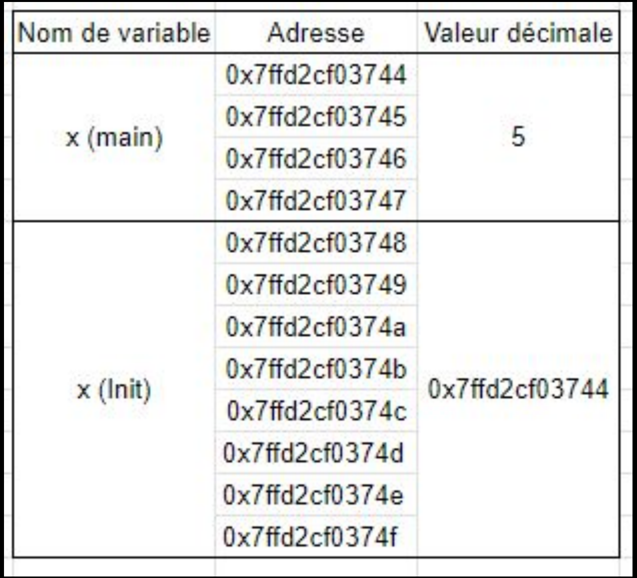


Nom de variable	Adresse	Valeur décimale
x (main)	0x7ffd2cf03744	0
	0x7ffd2cf03745	
	0x7ffd2cf03746	
	0x7ffd2cf03747	
x (Init)	0x7ffd2cf03748	5
	0x7ffd2cf03749	
	0x7ffd2cf0374a	
	0x7ffd2cf0374b	

Les pointeurs

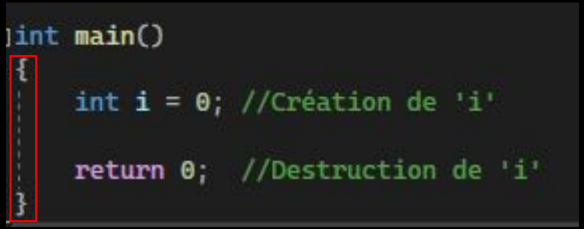


```
void Init( int* x )
{
    *x = 5;
}

int main()
{
    int x = 0;

    Init( &x );

    printf( "%d", x );

    return 0;
}
```



Nom de variable	Adresse	Valeur décimale
x (main)	0x7ffd2cf03744	5
	0x7ffd2cf03745	
	0x7ffd2cf03746	
	0x7ffd2cf03747	
x (Init)	0x7ffd2cf03748	0x7ffd2cf03744
	0x7ffd2cf03749	
	0x7ffd2cf0374a	
	0x7ffd2cf0374b	
	0x7ffd2cf0374c	
	0x7ffd2cf0374d	
	0x7ffd2cf0374e	
	0x7ffd2cf0374f	

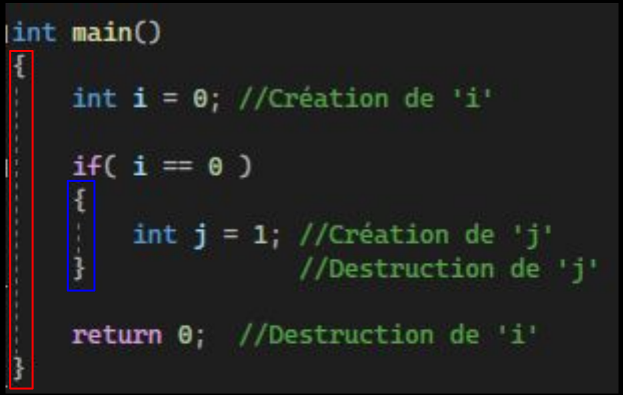
La pile :
Allocation de mémoire automatique



```
int main()
{
    int i = 0; //Création de 'i'

    return 0;  //Destruction de 'i'
}
```

La pile :
Allocation de mémoire automatique

scope du main
scope du if

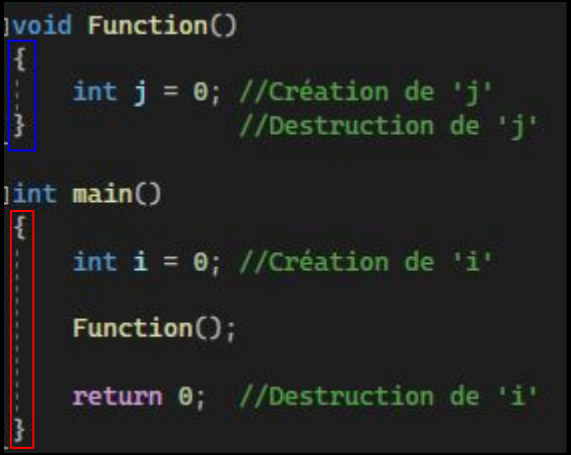


```
int main()
{
    int i = 0; //Création de 'i'

    if( i == 0 )
    {
        int j = 1; //Création de 'j'
    }              //Destruction de 'j'

    return 0;  //Destruction de 'i'
}
```

La pile :
Allocation de mémoire automatique

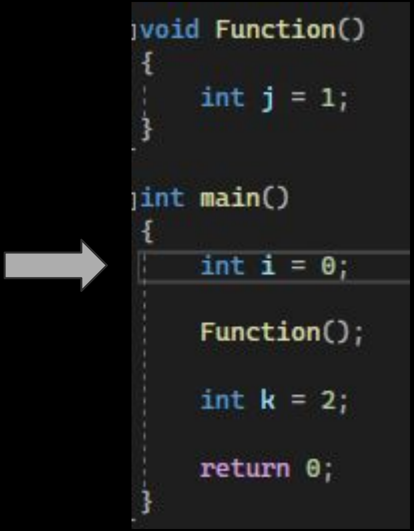
scope de Function

scope du main

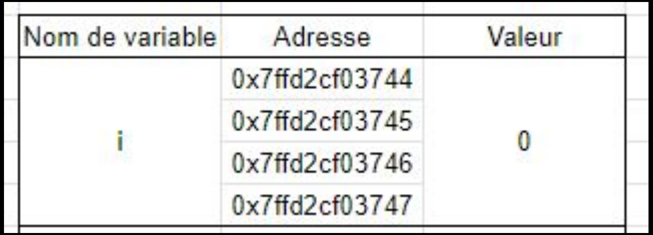


```
void Function()
{
    int j = 0; //Création de 'j'
}              //Destruction de 'j'

int main()
{
    int i = 0; //Création de 'i'

    Function();

    return 0;  //Destruction de 'i'
}
```

La pile :
Gestion de la mémoire

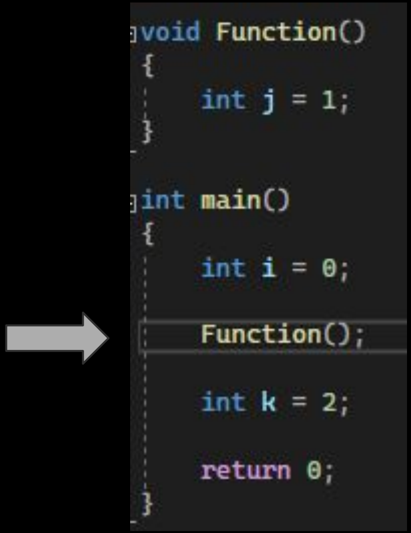


```c
void Function()
{
    int j = 1;
}

int main()
{
    int i = 0;

    Function();

    int k = 2;

    return 0;
}
```

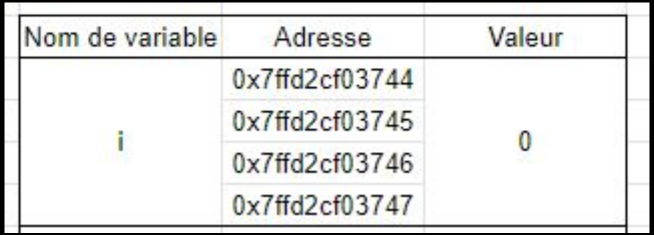


Nom de variable	Adresse	Valeur
i	0x7ffd2cf03744	0
	0x7ffd2cf03745	
	0x7ffd2cf03746	
	0x7ffd2cf03747	

La pile :
Gestion de la mémoire

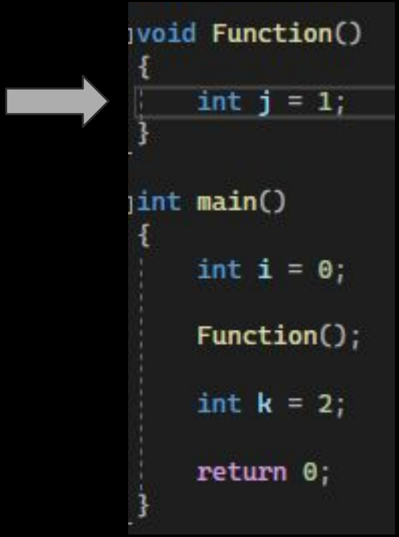


```
void Function()
{
    int j = 1;
}

int main()
{
    int i = 0;

    Function();

    int k = 2;

    return 0;
}
```

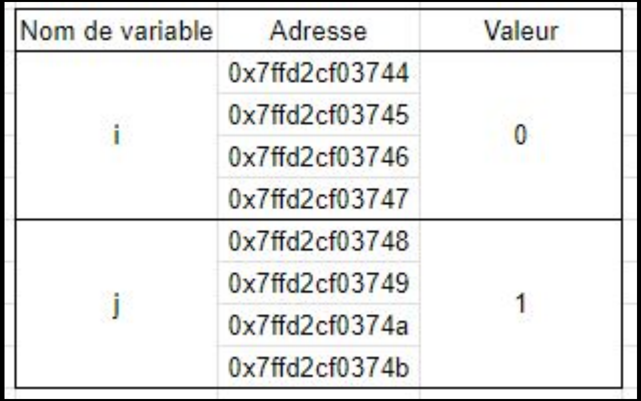


Nom de variable	Adresse	Valeur
i	0x7ffd2cf03744	0
	0x7ffd2cf03745	
	0x7ffd2cf03746	
	0x7ffd2cf03747	

La pile :
Gestion de la mémoire

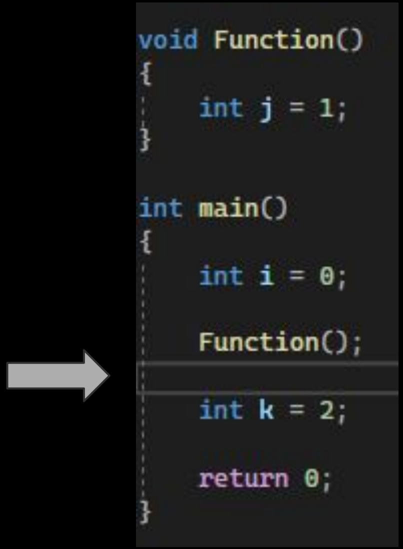


```c
void Function()
{
    int j = 1;
}

int main()
{
    int i = 0;

    Function();

    int k = 2;

    return 0;
}
```

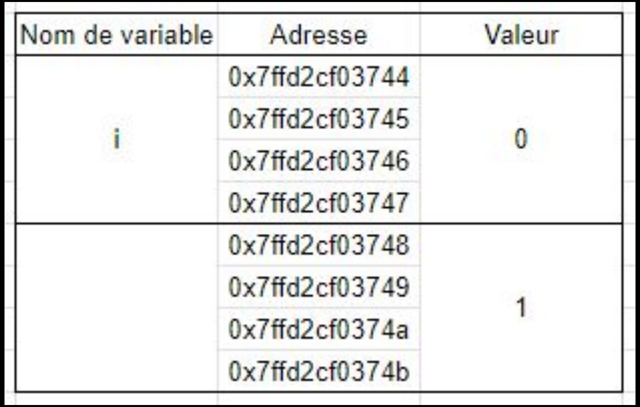


Nom de variable	Adresse	Valeur
i	0x7ffd2cf03744	0
	0x7ffd2cf03745	
	0x7ffd2cf03746	
	0x7ffd2cf03747	
j	0x7ffd2cf03748	1
	0x7ffd2cf03749	
	0x7ffd2cf0374a	
	0x7ffd2cf0374b	

La pile :
Gestion de la mémoire

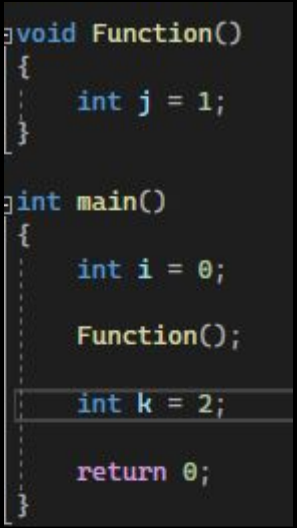


```c
void Function()
{
    int j = 1;
}

int main()
{
    int i = 0;

    Function();

    int k = 2;

    return 0;
}
```

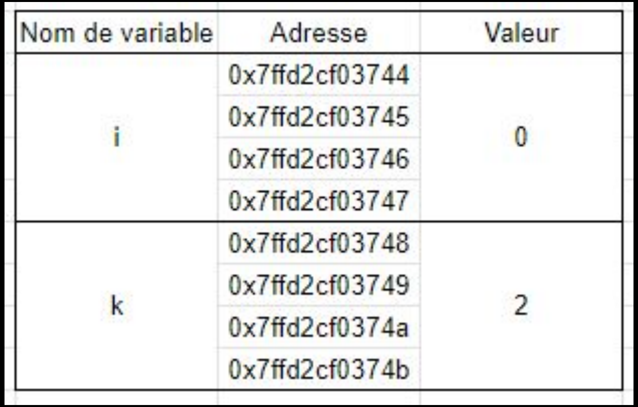


Nom de variable	Adresse	Valeur
i	0x7ffd2cf03744	0
	0x7ffd2cf03745	
	0x7ffd2cf03746	
	0x7ffd2cf03747	
	0x7ffd2cf03748	1
	0x7ffd2cf03749	
	0x7ffd2cf0374a	
	0x7ffd2cf0374b	

La pile :
Gestion de la mémoire

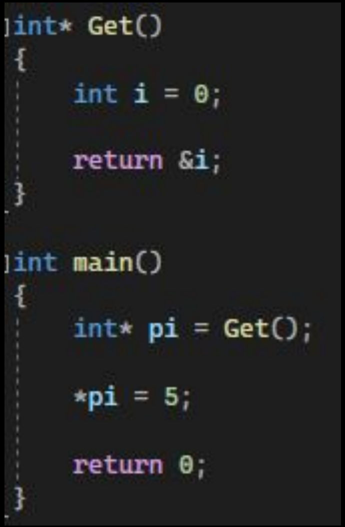


```
void Function()
{
    int j = 1;
}

int main()
{
    int i = 0;

    Function();

    int k = 2;

    return 0;
}
```



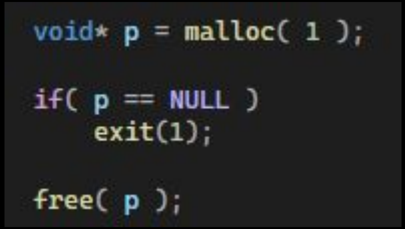
Nom de variable	Adresse	Valeur
i	0x7ffd2cf03744	0
	0x7ffd2cf03745	
	0x7ffd2cf03746	
	0x7ffd2cf03747	
k	0x7ffd2cf03748	2
	0x7ffd2cf03749	
	0x7ffd2cf0374a	
	0x7ffd2cf0374b	

La pile :
Gestion de la mémoire

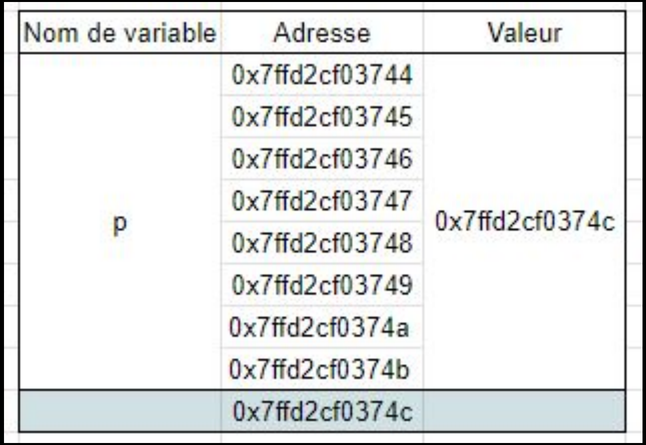


```
int* Get()
{
    int i = 0;

    return &i;
}

int main()
{
    int* pi = Get();

    *pi = 5;

    return 0;
}
```

Le tas :
Allocation dynamique

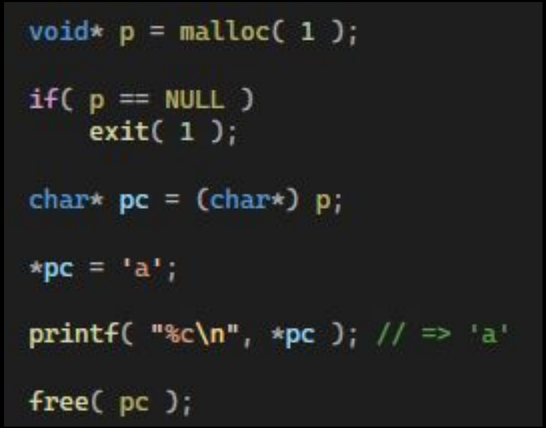


```c
void* p = malloc( 1 );

if( p == NULL )
    exit(1);

free( p );
```

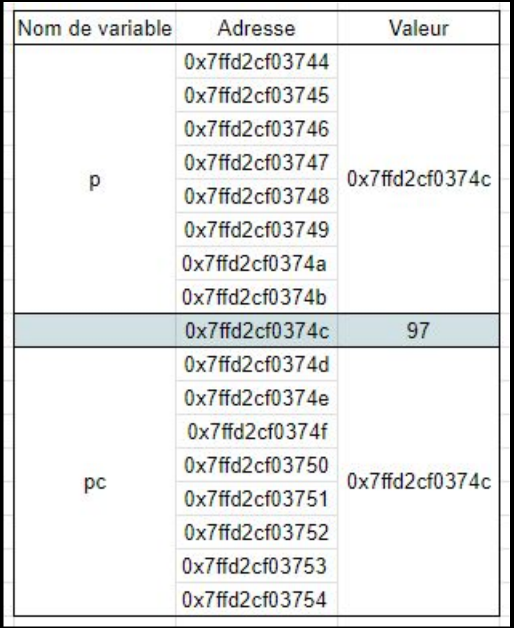


Nom de variable	Adresse	Valeur
p	0x7ffd2cf03744	0x7ffd2cf0374c
	0x7ffd2cf03745	
	0x7ffd2cf03746	
	0x7ffd2cf03747	
	0x7ffd2cf03748	
	0x7ffd2cf03749	
	0x7ffd2cf0374a	
	0x7ffd2cf0374b	
	0x7ffd2cf0374c	

Le tas :
Allocation dynamique

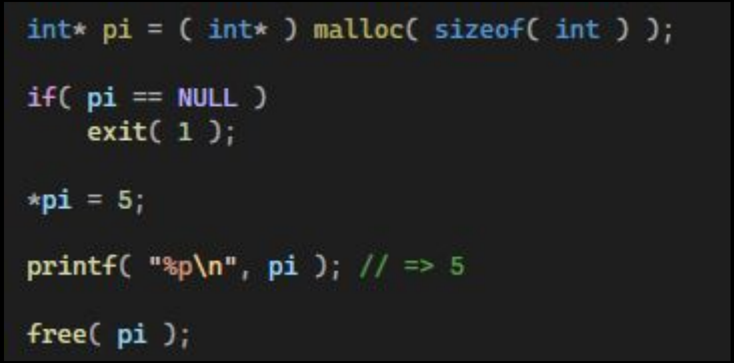


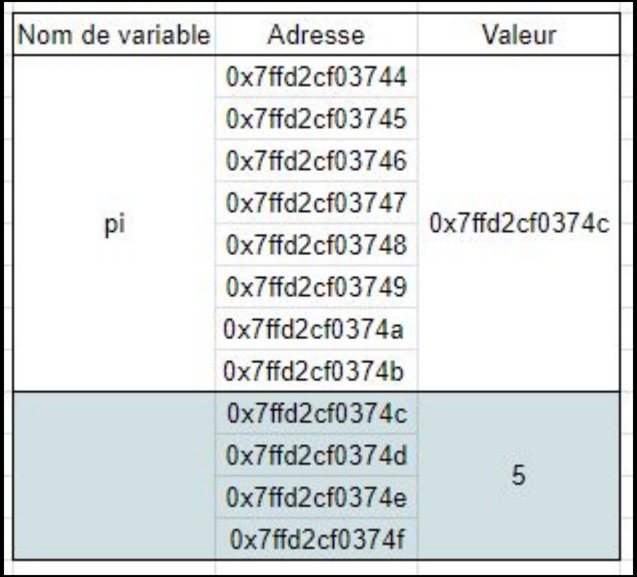
```c
void* p = malloc( 1 );

if( p == NULL )
    exit( 1 );

char* pc = (char*) p;

*pc = 'a';

printf( "%c\n", *pc ); // => 'a'

free( pc );
```



Nom de variable	Adresse	Valeur
p	0x7ffd2cf03744	0x7ffd2cf0374c
	0x7ffd2cf03745	
	0x7ffd2cf03746	
	0x7ffd2cf03747	
	0x7ffd2cf03748	
	0x7ffd2cf03749	
	0x7ffd2cf0374a	
	0x7ffd2cf0374b	
	0x7ffd2cf0374c	97
pc	0x7ffd2cf0374d	0x7ffd2cf0374c
	0x7ffd2cf0374e	
	0x7ffd2cf0374f	
	0x7ffd2cf03750	
	0x7ffd2cf03751	
	0x7ffd2cf03752	
	0x7ffd2cf03753	
	0x7ffd2cf03754	

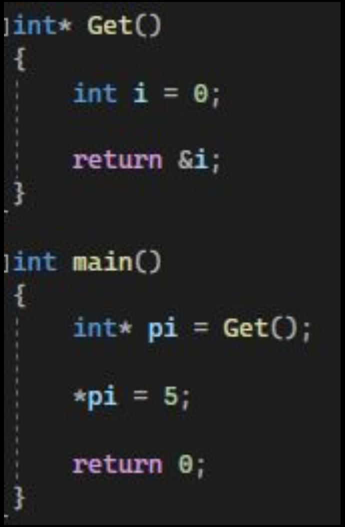
Le tas :
Allocation dynamique


```
int* pi = ( int* ) malloc( sizeof( int ) );

if( pi == NULL )
    exit( 1 );

*pi = 5;

printf( "%p\n", pi ); // => 5

free( pi );
```



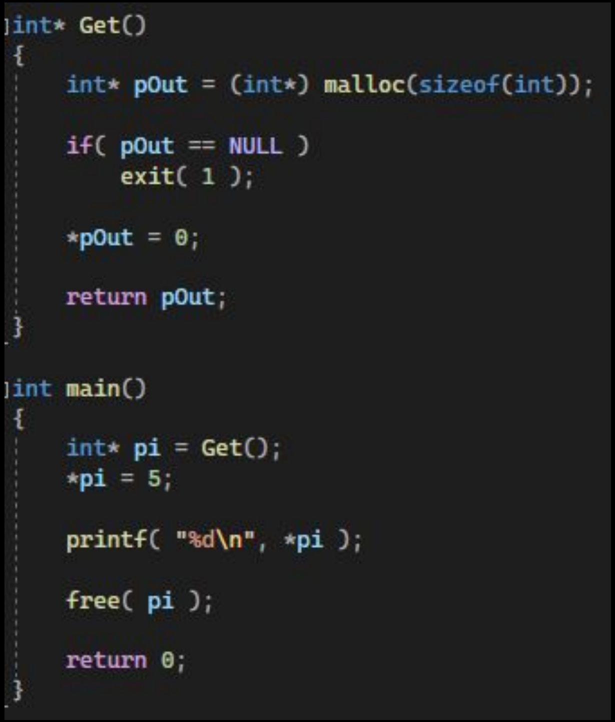
Nom de variable	Adresse	Valeur
pi	0x7ffd2cf03744	0x7ffd2cf0374c
	0x7ffd2cf03745	
	0x7ffd2cf03746	
	0x7ffd2cf03747	
	0x7ffd2cf03748	
	0x7ffd2cf03749	
	0x7ffd2cf0374a	
	0x7ffd2cf0374b	
	0x7ffd2cf0374c	5
	0x7ffd2cf0374d	
	0x7ffd2cf0374e	
	0x7ffd2cf0374f	

Le tas :
Allocation dynamique



```c
int* Get()
{
    int i = 0;

    return &i;
}

int main()
{
    int* pi = Get();

    *pi = 5;

    return 0;
}
```

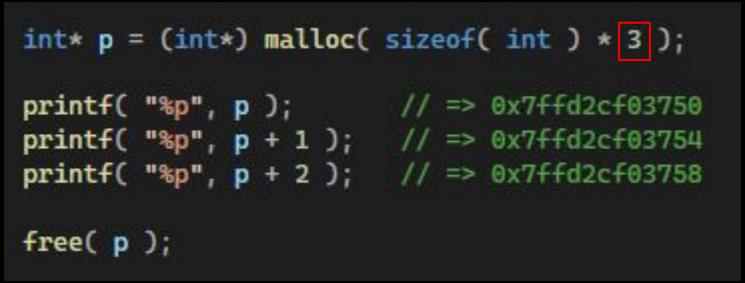
✗



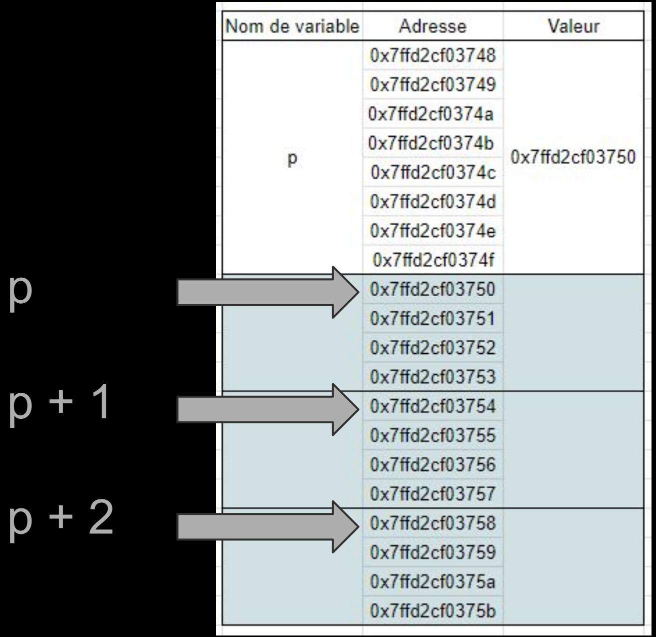
```c
int* Get()
{
    int* pOut = (int*) malloc(sizeof(int));

    if( pOut == NULL )
        exit( 1 );

    *pOut = 0;

    return pOut;
}

int main()
{
    int* pi = Get();
    *pi = 5;

    printf( "%d\n", *pi );

    free( pi );

    return 0;
}
```

✓

Le tas :
Arithmétique des pointeurs

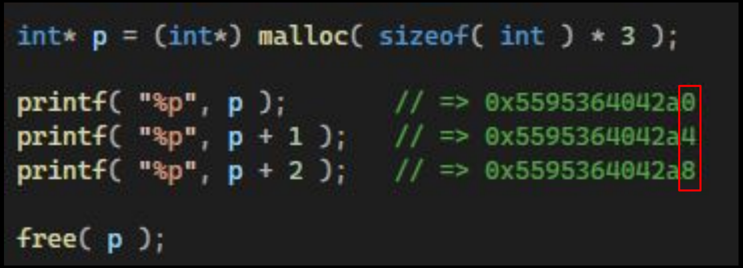


```c
int* p = (int*) malloc( sizeof( int ) * 3 );

printf( "%p", p );        // => 0x7ffd2cf03750
printf( "%p", p + 1 );    // => 0x7ffd2cf03754
printf( "%p", p + 2 );    // => 0x7ffd2cf03758

free( p );
```

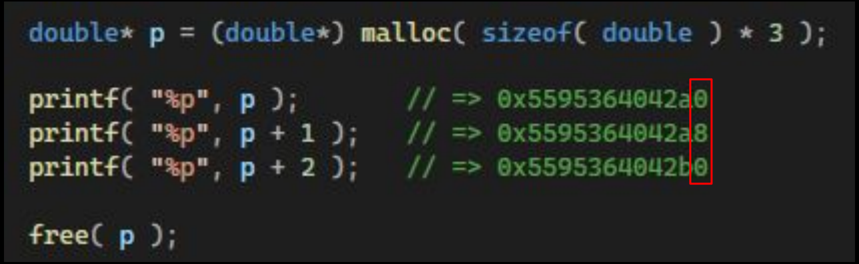


Nom de variable	Adresse	Valeur
p	0x7ffd2cf03748	0x7ffd2cf03750
	0x7ffd2cf03749	
	0x7ffd2cf0374a	
	0x7ffd2cf0374b	
	0x7ffd2cf0374c	
	0x7ffd2cf0374d	
	0x7ffd2cf0374e	
	0x7ffd2cf0374f	
	0x7ffd2cf03750	
	0x7ffd2cf03751	
	0x7ffd2cf03752	
	0x7ffd2cf03753	
	0x7ffd2cf03754	
	0x7ffd2cf03755	
	0x7ffd2cf03756	
	0x7ffd2cf03757	
	0x7ffd2cf03758	
	0x7ffd2cf03759	
	0x7ffd2cf0375a	
	0x7ffd2cf0375b	

Le tas :
Arithmétique des pointeurs

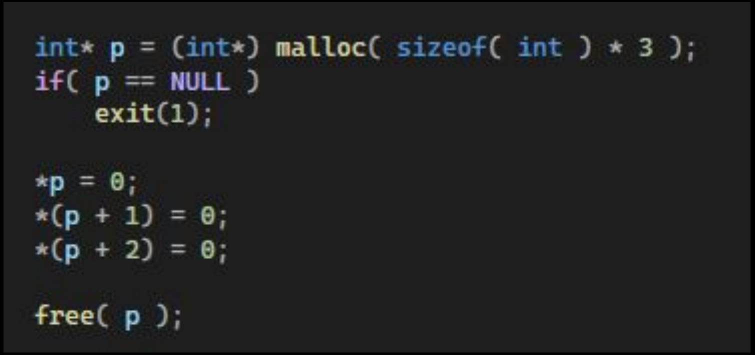


```c
int* p = (int*) malloc( sizeof( int ) * 3 );

printf( "%p", p );         // => 0x5595364042a0
printf( "%p", p + 1 );     // => 0x5595364042a4
printf( "%p", p + 2 );     // => 0x5595364042a8

free( p );
```

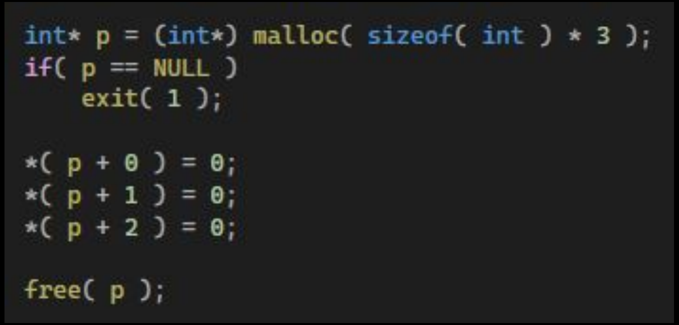


```c
double* p = (double*) malloc( sizeof( double ) * 3 );

printf( "%p", p );         // => 0x5595364042a0
printf( "%p", p + 1 );     // => 0x5595364042a8
printf( "%p", p + 2 );     // => 0x5595364042b0

free( p );
```

Le tas :
Arithmétique des pointeurs

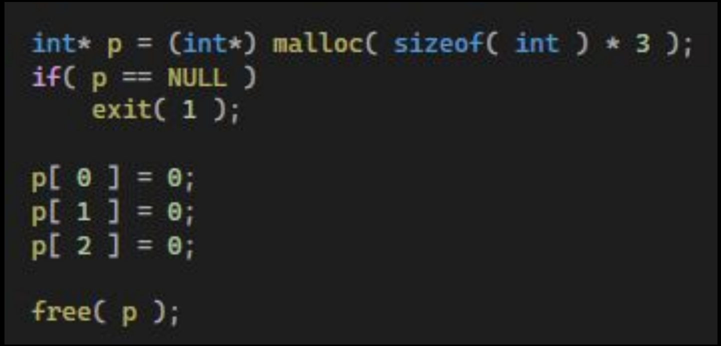


```c
int* p = (int*) malloc( sizeof( int ) * 3 );
if( p == NULL )
    exit(1);

*p = 0;
*(p + 1) = 0;
*(p + 2) = 0;

free( p );
```



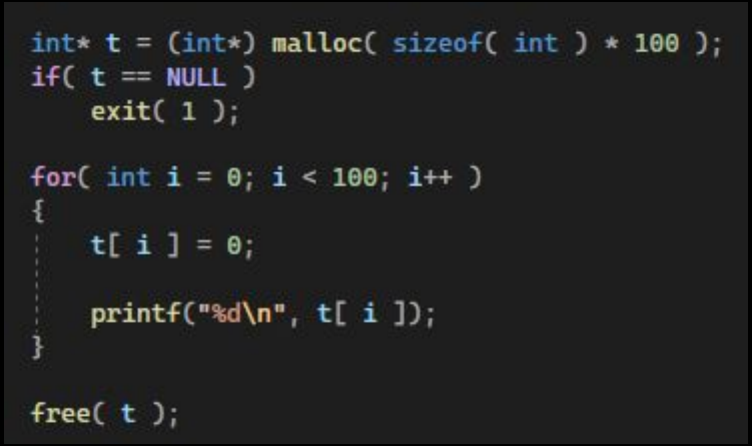
```c
int* p = (int*) malloc( sizeof( int ) * 3 );
if( p == NULL )
    exit( 1 );

*( p + 0 ) = 0;
*( p + 1 ) = 0;
*( p + 2 ) = 0;

free( p );
```

Le tas :
Arithmétique des pointeurs

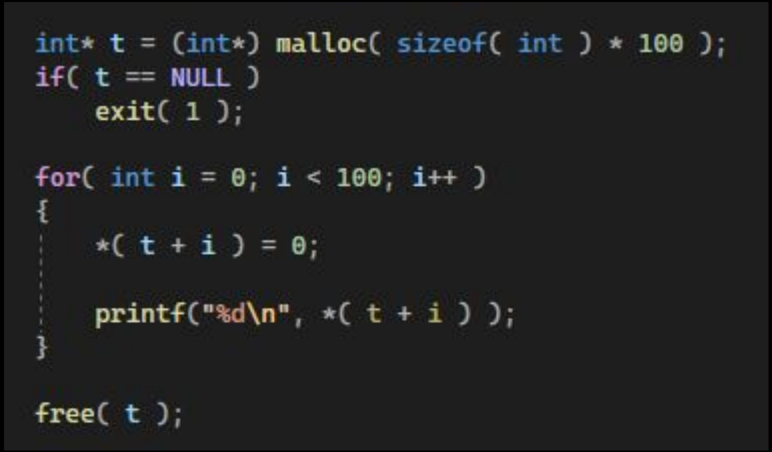


```c
int* p = (int*) malloc( sizeof( int ) * 3 );
if( p == NULL )
    exit( 1 );

p[ 0 ] = 0;
p[ 1 ] = 0;
p[ 2 ] = 0;

free( p );
```

Les tableaux

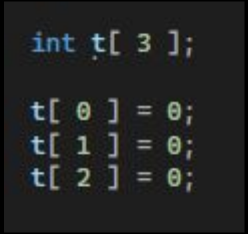


```c
int* t = (int*) malloc( sizeof( int ) * 100 );
if( t == NULL )
    exit( 1 );

for( int i = 0; i < 100; i++ )
{
    t[ i ] = 0;

    printf("%d\n", t[ i ]);
}

free( t );
```

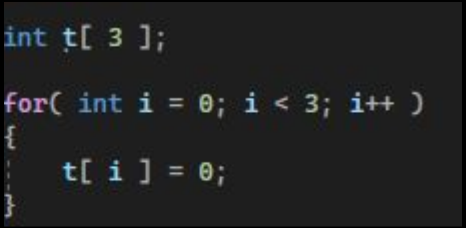


```c
int* t = (int*) malloc( sizeof( int ) * 100 );
if( t == NULL )
    exit( 1 );

for( int i = 0; i < 100; i++ )
{
    *( t + i ) = 0;

    printf("%d\n", *( t + i ) );
}

free( t );
```

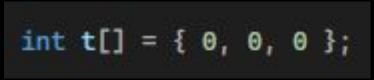
Les tableaux



```c
int t[ 3 ];

t[ 0 ] = 0;
t[ 1 ] = 0;
t[ 2 ] = 0;
```

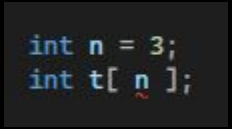


```c
int t[ 3 ];

for( int i = 0; i < 3; i++ )
{
    t[ i ] = 0;
}
```



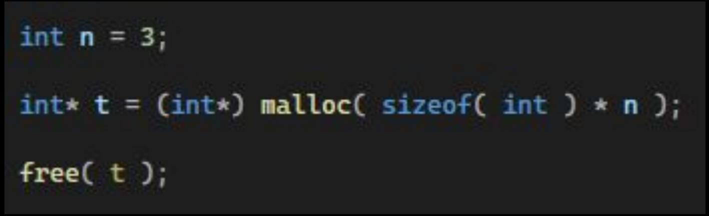
```c
int t[] = { 0, 0, 0 };
```

Les tableaux



```c
int n = 3;
int t[ n ];
```

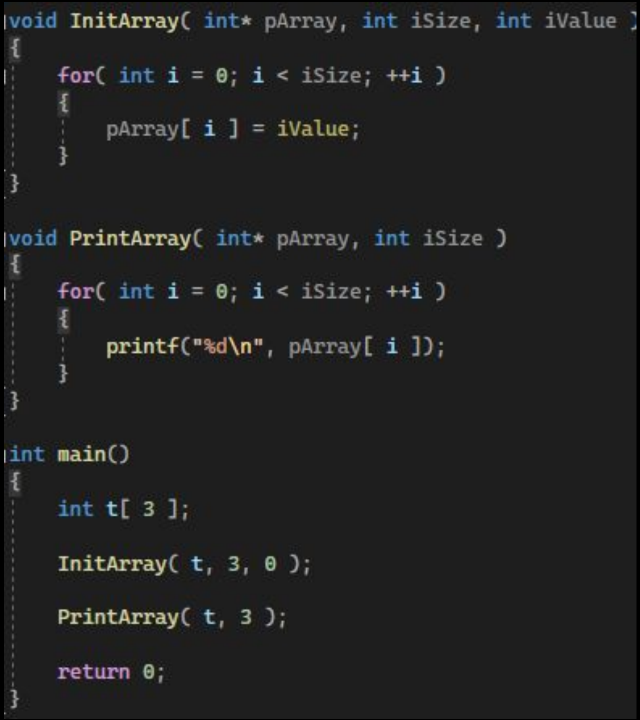
✗



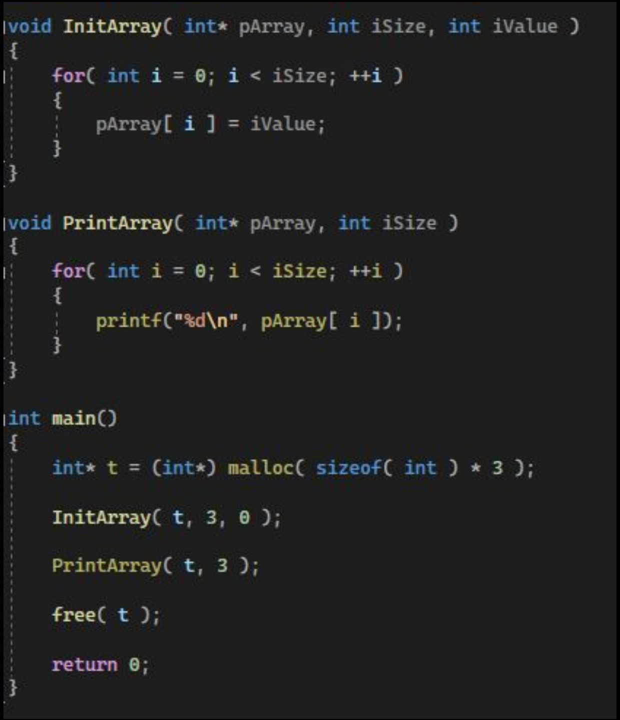
```c
int n = 3;

int* t = (int*) malloc( sizeof( int ) * n );

free( t );
```

✓

Les tableaux :
Passage en paramètre

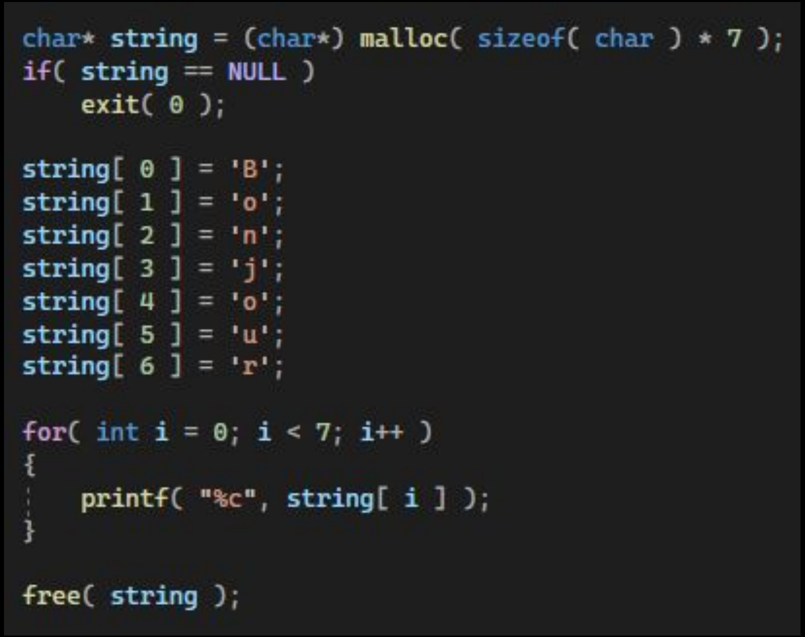


```c
void InitArray( int* pArray, int iSize, int iValue )
{
    for( int i = 0; i < iSize; ++i )
    {
        pArray[ i ] = iValue;
    }
}

void PrintArray( int* pArray, int iSize )
{
    for( int i = 0; i < iSize; ++i )
    {
        printf("%d\n", pArray[ i ]);
    }
}

int main()
{
    int t[ 3 ];

    InitArray( t, 3, 0 );

    PrintArray( t, 3 );

    return 0;
}
```



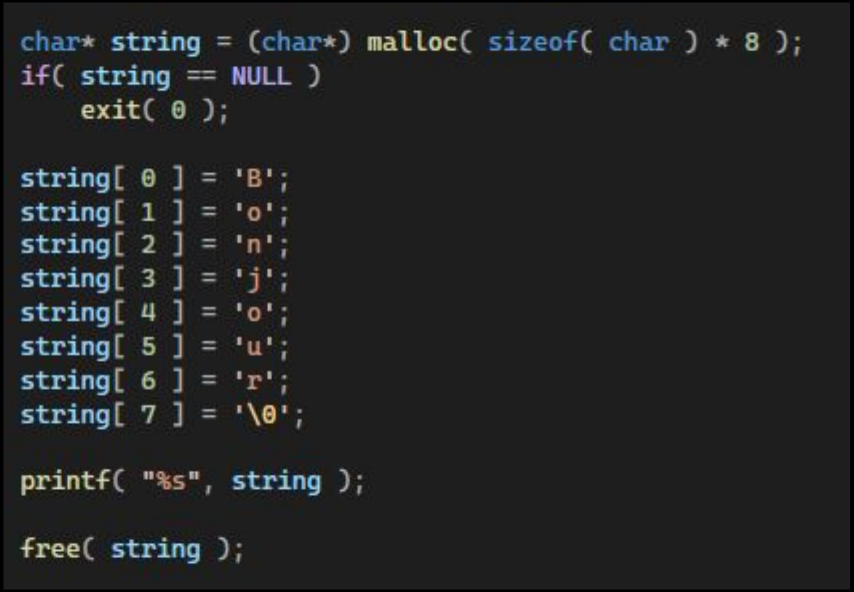
```c
void InitArray( int* pArray, int iSize, int iValue )
{
    for( int i = 0; i < iSize; ++i )
    {
        pArray[ i ] = iValue;
    }
}

void PrintArray( int* pArray, int iSize )
{
    for( int i = 0; i < iSize; ++i )
    {
        printf("%d\n", pArray[ i ]);
    }
}

int main()
{
    int* t = (int*) malloc( sizeof( int ) * 3 );

    InitArray( t, 3, 0 );

    PrintArray( t, 3 );

    free( t );

    return 0;
}
```

Les string



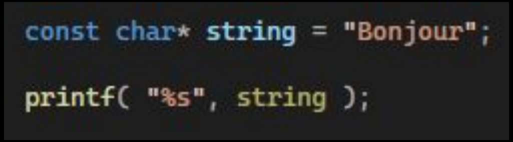
```c
char* string = (char*) malloc( sizeof( char ) * 7 );
if( string == NULL )
    exit( 0 );

string[ 0 ] = 'B';
string[ 1 ] = 'o';
string[ 2 ] = 'n';
string[ 3 ] = 'j';
string[ 4 ] = 'o';
string[ 5 ] = 'u';
string[ 6 ] = 'r';

for( int i = 0; i < 7; i++ )
{
    printf( "%c", string[ i ] );
}

free( string );
```

Les string

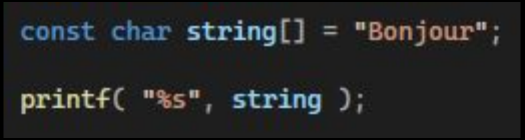


```c
char* string = (char*) malloc( sizeof( char ) * 8 );
if( string == NULL )
    exit( 0 );

string[ 0 ] = 'B';
string[ 1 ] = 'o';
string[ 2 ] = 'n';
string[ 3 ] = 'j';
string[ 4 ] = 'o';
string[ 5 ] = 'u';
string[ 6 ] = 'r';
string[ 7 ] = '\0';

printf( "%s", string );

free( string );
```

Les string

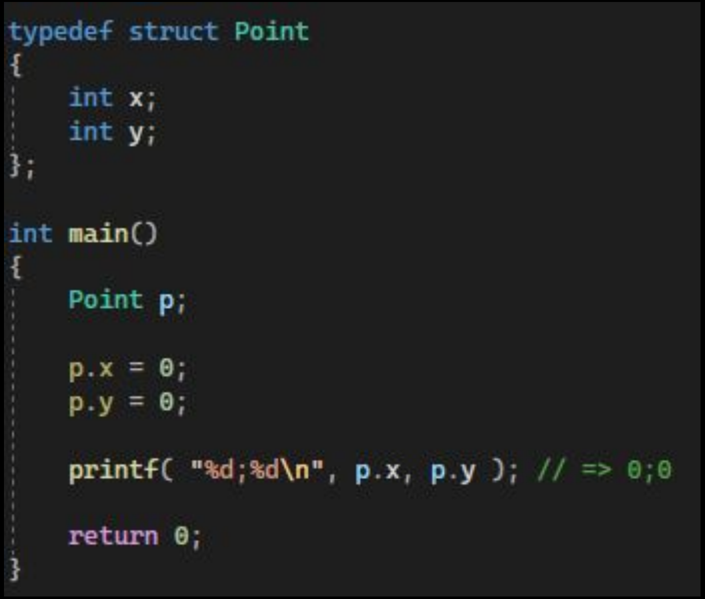


```c
const char* string = "Bonjour";

printf( "%s", string );
```



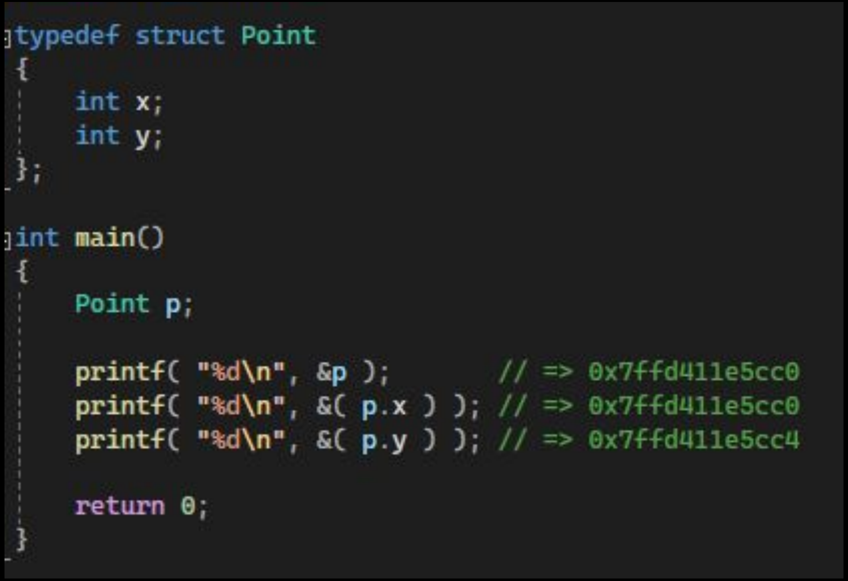
```c
const char string[] = "Bonjour";

printf( "%s", string );
```

Les struct



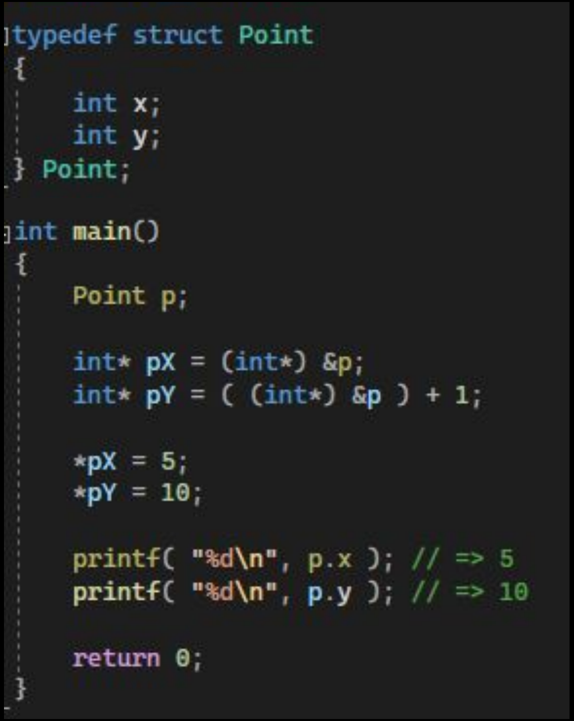
```c
typedef struct Point
{
    int x;
    int y;
};

int main()
{
    Point p;

    p.x = 0;
    p.y = 0;

    printf( "%d;%d\n", p.x, p.y ); // => 0;0

    return 0;
}
```

Les struct



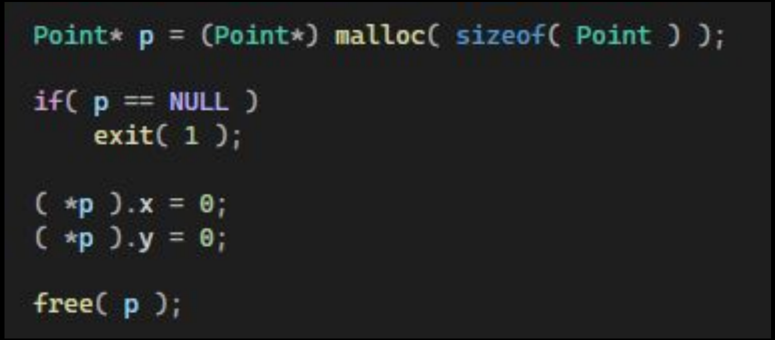
```c
typedef struct Point
{
    int x;
    int y;
};

int main()
{
    Point p;

    printf( "%d\n", &p );          // => 0x7ffd411e5cc0
    printf( "%d\n", &( p.x ) ); // => 0x7ffd411e5cc0
    printf( "%d\n", &( p.y ) ); // => 0x7ffd411e5cc4

    return 0;
}
```

Les struct

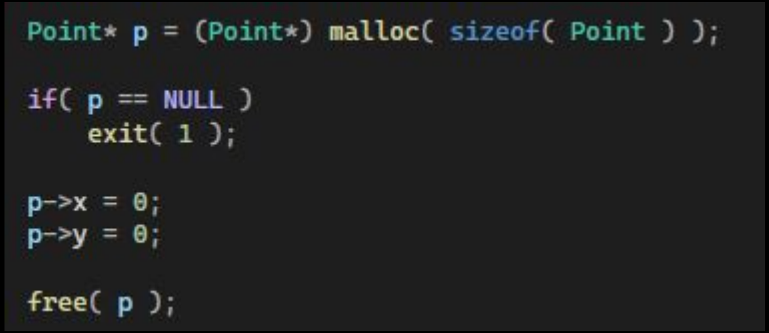


```c
typedef struct Point
{
    int x;
    int y;
} Point;

int main()
{
    Point p;

    int* pX = (int*) &p;
    int* pY = ( (int*) &p ) + 1;

    *pX = 5;
    *pY = 10;

    printf( "%d\n", p.x ); // => 5
    printf( "%d\n", p.y ); // => 10

    return 0;
}
```

Les struct

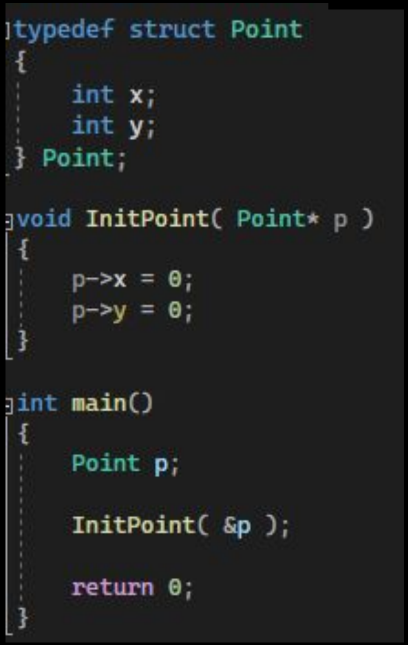


```c
Point* p = (Point*) malloc( sizeof( Point ) );

if( p == NULL )
    exit( 1 );

( *p ).x = 0;
( *p ).y = 0;

free( p );
```



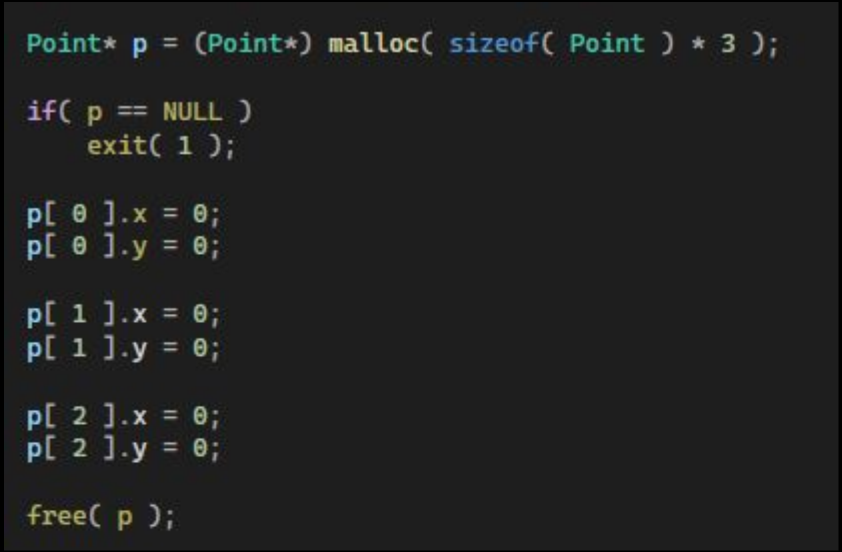
```c
Point* p = (Point*) malloc( sizeof( Point ) );

if( p == NULL )
    exit( 1 );

p->x = 0;
p->y = 0;

free( p );
```

Les struct



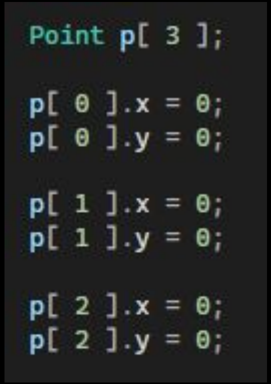
```
typedef struct Point
{
    int x;
    int y;
} Point;

void InitPoint( Point* p )
{
    p->x = 0;
    p->y = 0;
}

int main()
{
    Point p;

    InitPoint( &p );

    return 0;
}
```

Les struct



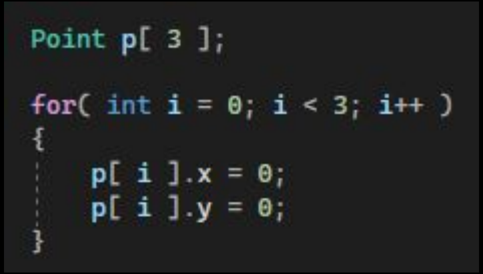
```c
Point* p = (Point*) malloc( sizeof( Point ) * 3 );

if( p == NULL )
    exit( 1 );

p[ 0 ].x = 0;
p[ 0 ].y = 0;

p[ 1 ].x = 0;
p[ 1 ].y = 0;

p[ 2 ].x = 0;
p[ 2 ].y = 0;

free( p );
```

Les struct

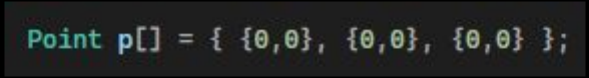


```
Point p[ 3 ];

p[ 0 ].x = 0;
p[ 0 ].y = 0;

p[ 1 ].x = 0;
p[ 1 ].y = 0;

p[ 2 ].x = 0;
p[ 2 ].y = 0;
```

Les struct

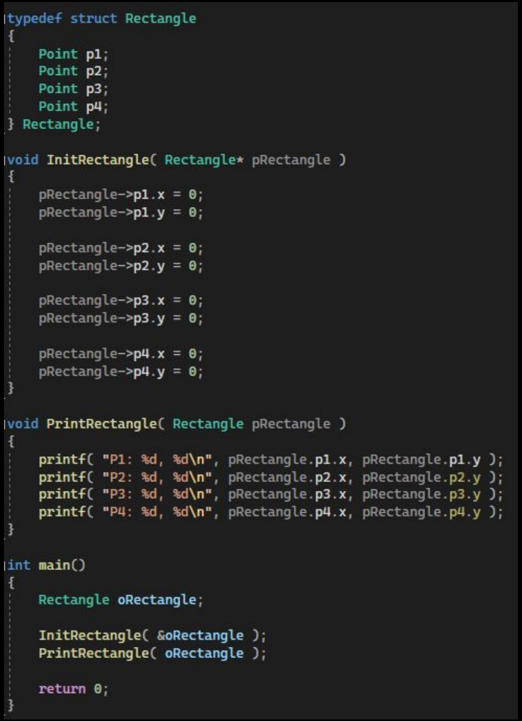


```c
Point p[ 3 ];

for( int i = 0; i < 3; i++ )
{
    p[ i ].x = 0;
    p[ i ].y = 0;
}
```

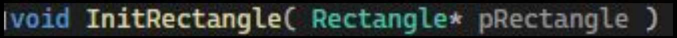


```c
Point p[] = { {0,0}, {0,0}, {0,0} };
```

Envoi par valeur/adresse

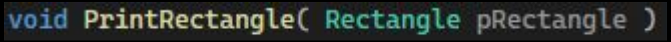


```c
typedef struct Rectangle
{
    Point p1;
    Point p2;
    Point p3;
    Point p4;
} Rectangle;

void InitRectangle( Rectangle* pRectangle )
{
    pRectangle->p1.x = 0;
    pRectangle->p1.y = 0;

    pRectangle->p2.x = 0;
    pRectangle->p2.y = 0;

    pRectangle->p3.x = 0;
    pRectangle->p3.y = 0;

    pRectangle->p4.x = 0;
    pRectangle->p4.y = 0;
}

void PrintRectangle( Rectangle pRectangle )
{
    printf( "P1: %d, %d\n", pRectangle.p1.x, pRectangle.p1.y );
    printf( "P2: %d, %d\n", pRectangle.p2.x, pRectangle.p2.y );
    printf( "P3: %d, %d\n", pRectangle.p3.x, pRectangle.p3.y );
    printf( "P4: %d, %d\n", pRectangle.p4.x, pRectangle.p4.y );
}

int main()
{
    Rectangle oRectangle;

    InitRectangle( &oRectangle );
    PrintRectangle( oRectangle );

    return 0;
}
```



```c
void InitRectangle( Rectangle* pRectangle )
```

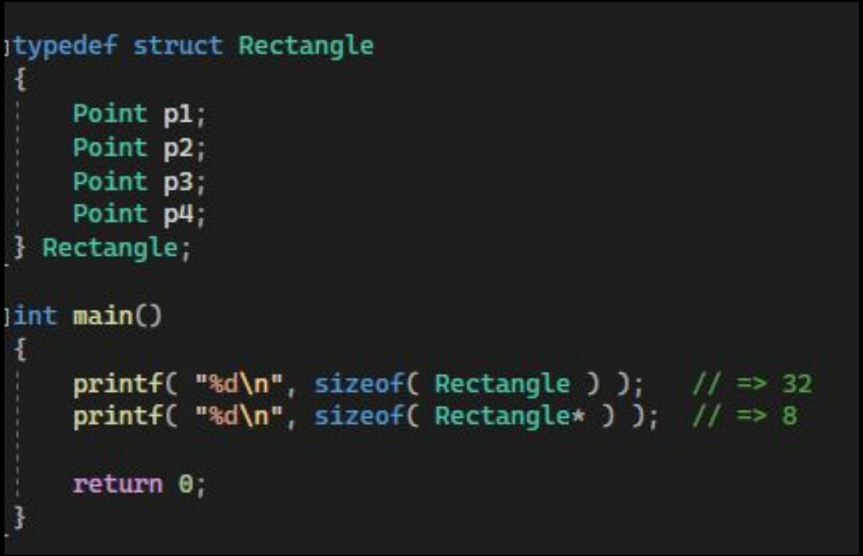
✓



```c
void PrintRectangle( Rectangle pRectangle )
```

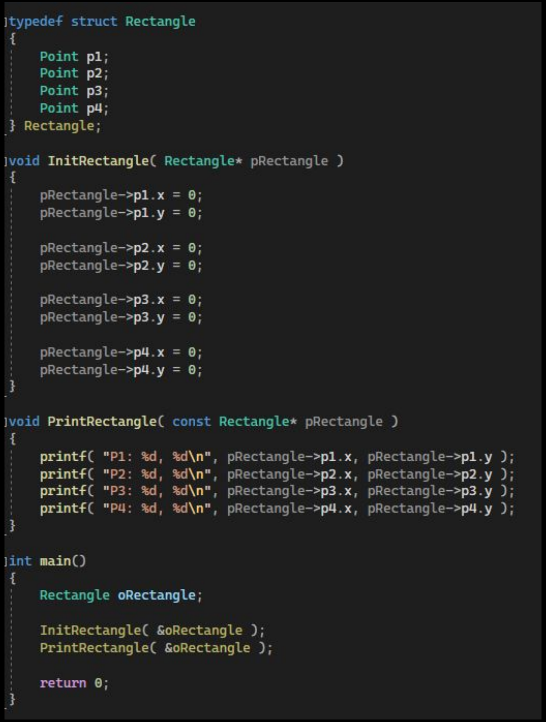
?

Envoi par valeur/adresse

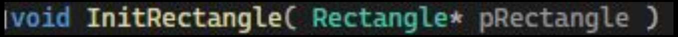


```c
typedef struct Rectangle
{
    Point p1;
    Point p2;
    Point p3;
    Point p4;
} Rectangle;

int main()
{
    printf( "%d\n", sizeof( Rectangle ) );   // => 32
    printf( "%d\n", sizeof( Rectangle* ) );  // => 8

    return 0;
}
```

Envoi par valeur/adresse

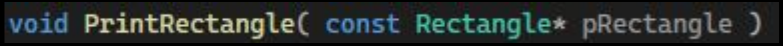


```c
typedef struct Rectangle
{
    Point p1;
    Point p2;
    Point p3;
    Point p4;
} Rectangle;

void InitRectangle( Rectangle* pRectangle )
{
    pRectangle->p1.x = 0;
    pRectangle->p1.y = 0;

    pRectangle->p2.x = 0;
    pRectangle->p2.y = 0;

    pRectangle->p3.x = 0;
    pRectangle->p3.y = 0;

    pRectangle->p4.x = 0;
    pRectangle->p4.y = 0;
}

void PrintRectangle( const Rectangle* pRectangle )
{
    printf( "P1: %d, %d\n", pRectangle->p1.x, pRectangle->p1.y );
    printf( "P2: %d, %d\n", pRectangle->p2.x, pRectangle->p2.y );
    printf( "P3: %d, %d\n", pRectangle->p3.x, pRectangle->p3.y );
    printf( "P4: %d, %d\n", pRectangle->p4.x, pRectangle->p4.y );
}

int main()
{
    Rectangle oRectangle;

    InitRectangle( &oRectangle );
    PrintRectangle( &oRectangle );

    return 0;
}
```



```c
void InitRectangle( Rectangle* pRectangle )
```

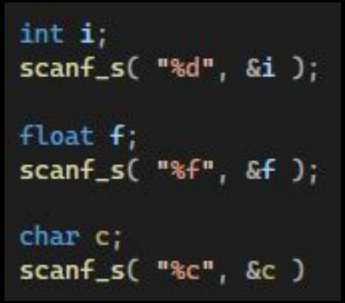
✓



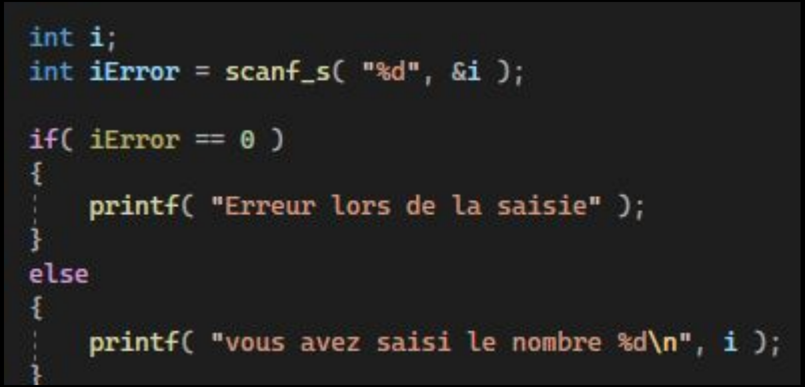
```c
void PrintRectangle( const Rectangle* pRectangle )
```

✓

Entrées

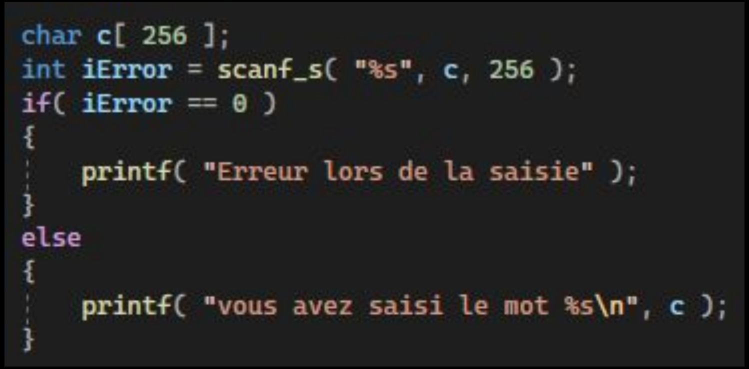

```
int i;
scanf_s( "%d", &i );

float f;
scanf_s( "%f", &f );

char c;
scanf_s( "%c", &c )
```

Entrées



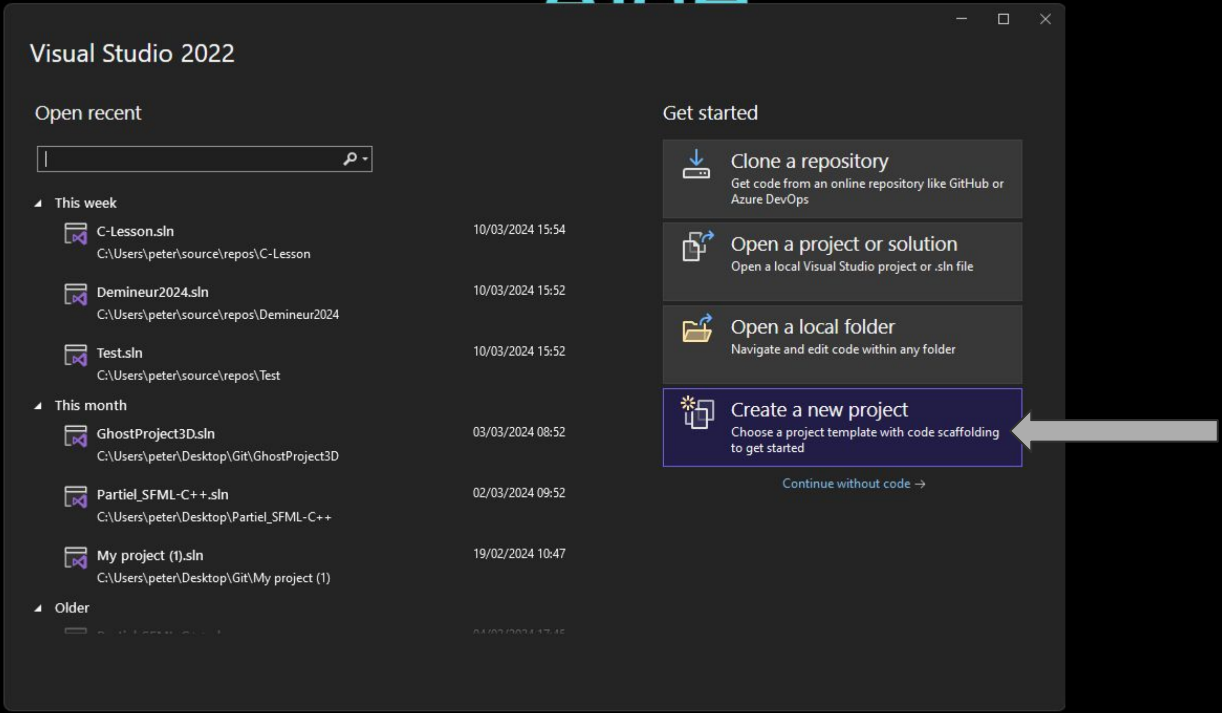
```c
int i;
int iError = scanf_s( "%d", &i );

if( iError == 0 )
{
    printf( "Erreur lors de la saisie" );
}
else
{
    printf( "vous avez saisi le nombre %d\n", i );
}
```

Entrées



```
char c[ 256 ];
int iError = scanf_s( "%s", c, 256 );
if( iError == 0 )
{
    printf( "Erreur lors de la saisie" );
}
else
{
    printf( "vous avez saisi le mot %s\n", c );
}
```

Aide :
Créer un projet



Visual Studio 2022

Open recent

This week

C-Lesson.sln — C:\Users\peter\source\repos\C-Lesson — 10/03/2024 15:54

Demineur2024.sln — C:\Users\peter\source\repos\Demineur2024 — 10/03/2024 15:52

Test.sln — C:\Users\peter\source\repos\Test — 10/03/2024 15:52

This month

GhostProject3D.sln — C:\Users\peter\Desktop\Git\GhostProject3D — 03/03/2024 08:52

Partiel_SFML-C++.sln — C:\Users\peter\Desktop\Partiel_SFML-C++ — 02/03/2024 09:52

My project (1).sln — C:\Users\peter\Desktop\Git\My project (1) — 19/02/2024 10:47

Older

Get started

Clone a repository
Get code from an online repository like GitHub or Azure DevOps

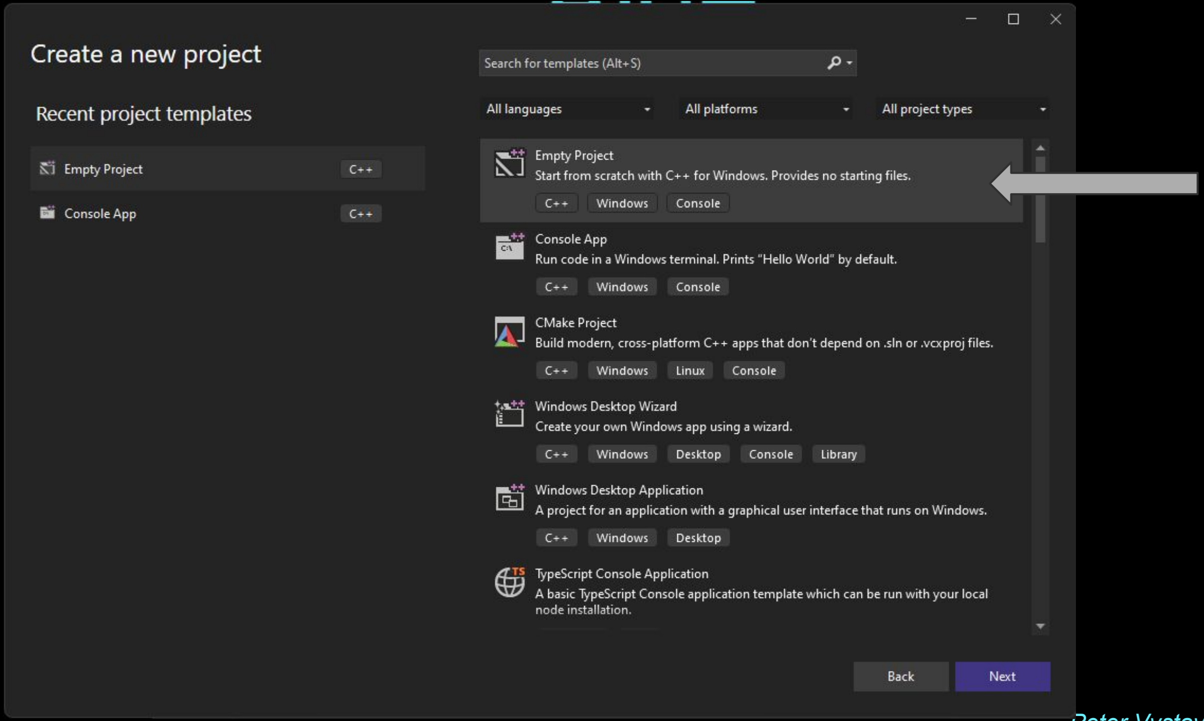
Open a project or solution
Open a local Visual Studio project or .sln file

Open a local folder
Navigate and edit code within any folder

Create a new project
Choose a project template with code scaffolding to get started

Continue without code →

Aide :
Créer un projet



Create a new project

Recent project templates

Empty Project	C++
Console App	C++

Search for templates (Alt+S)

All languages · All platforms · All project types

Empty Project
Start from scratch with C++ for Windows. Provides no starting files.
C++ · Windows · Console

Console App
Run code in a Windows terminal. Prints "Hello World" by default.
C++ · Windows · Console

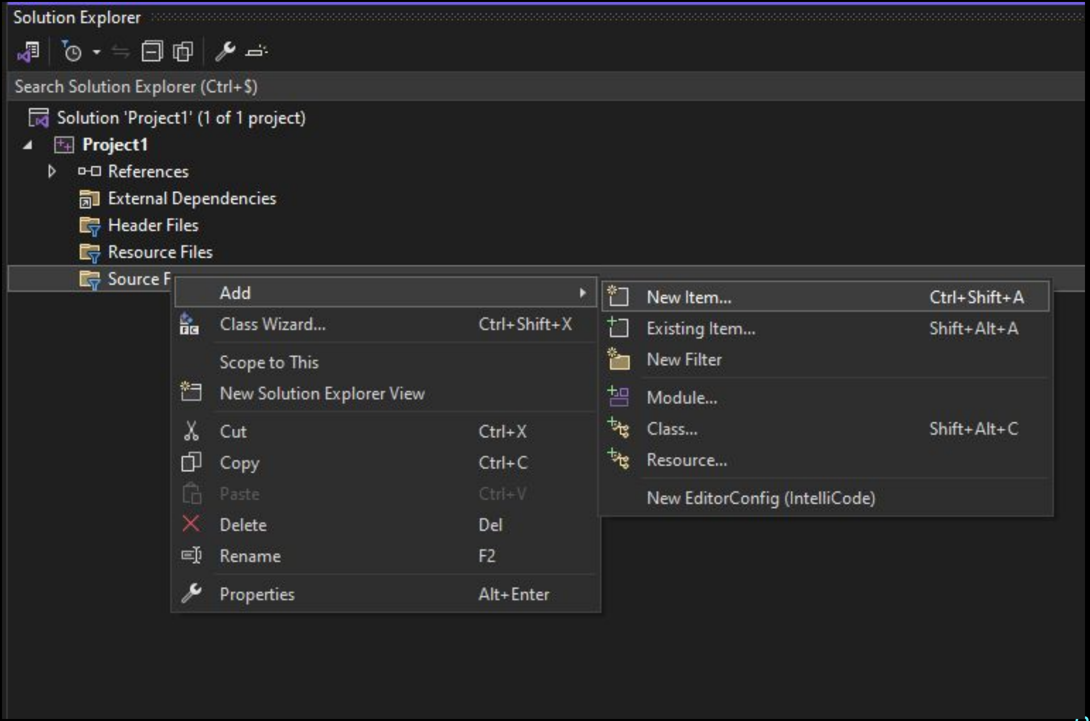
CMake Project
Build modern, cross-platform C++ apps that don't depend on .sln or .vcxproj files.
C++ · Windows · Linux · Console

Windows Desktop Wizard
Create your own Windows app using a wizard.
C++ · Windows · Desktop · Console · Library

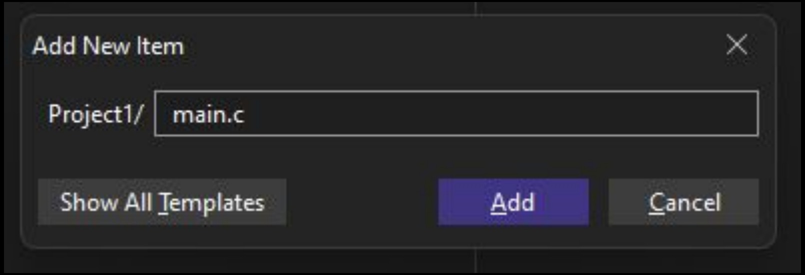
Windows Desktop Application
A project for an application with a graphical user interface that runs on Windows.
C++ · Windows · Desktop

TypeScript Console Application
A basic TypeScript Console application template which can be run with your local node installation.

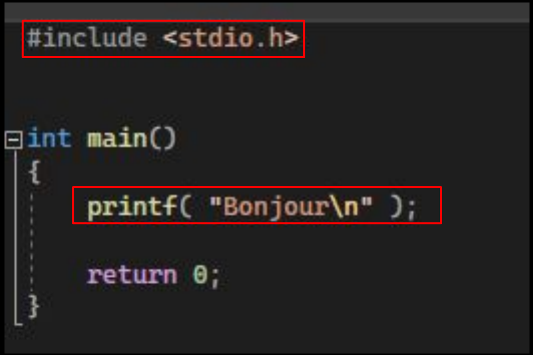
Aide :
Créer un projet



Aide :
Créer un projet

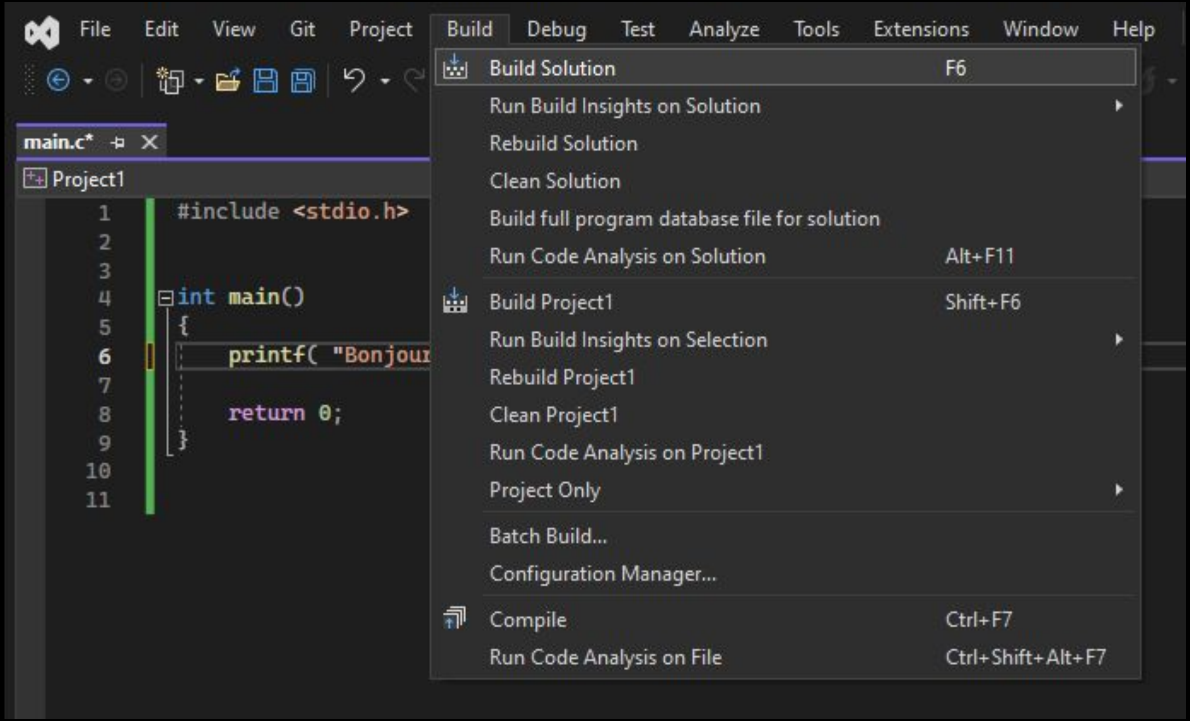


Aide :
Créer un projet



```
#include <stdio.h>


int main()
{
    printf( "Bonjour\n" );

    return 0;
}
```

Aide :
Compiler le programme



File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help

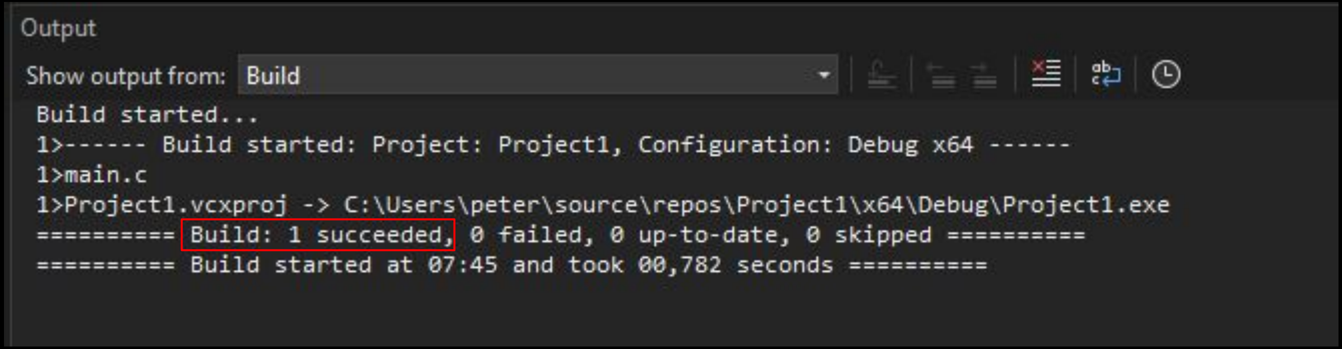
main.c*

Project1

```
#include <stdio.h>


int main()
{
    printf( "Bonjour

    return 0;
}
```

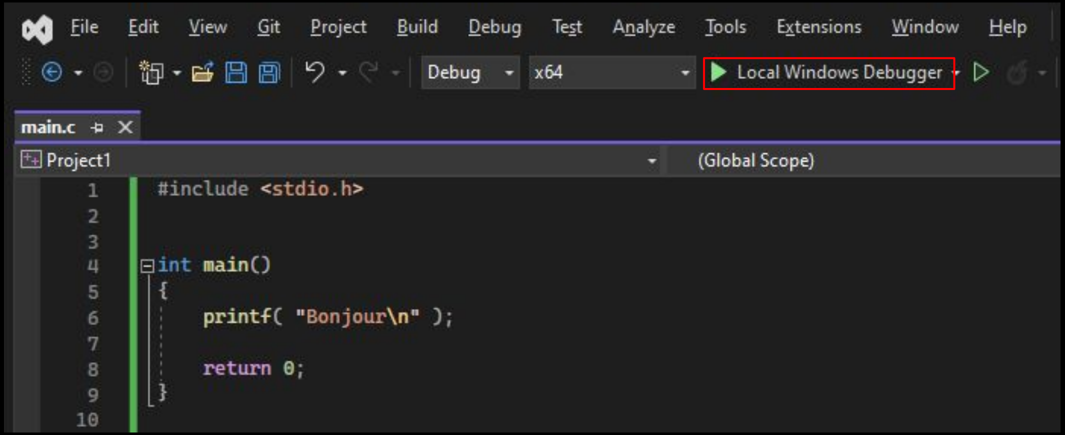
Build Solution	F6
Run Build Insights on Solution	
Rebuild Solution	
Clean Solution	
Build full program database file for solution	
Run Code Analysis on Solution	Alt+F11
Build Project1	Shift+F6
Run Build Insights on Selection	
Rebuild Project1	
Clean Project1	
Run Code Analysis on Project1	
Project Only	
Batch Build...	
Configuration Manager...	
Compile	Ctrl+F7
Run Code Analysis on File	Ctrl+Shift+Alt+F7

Aide :
Compiler le programme



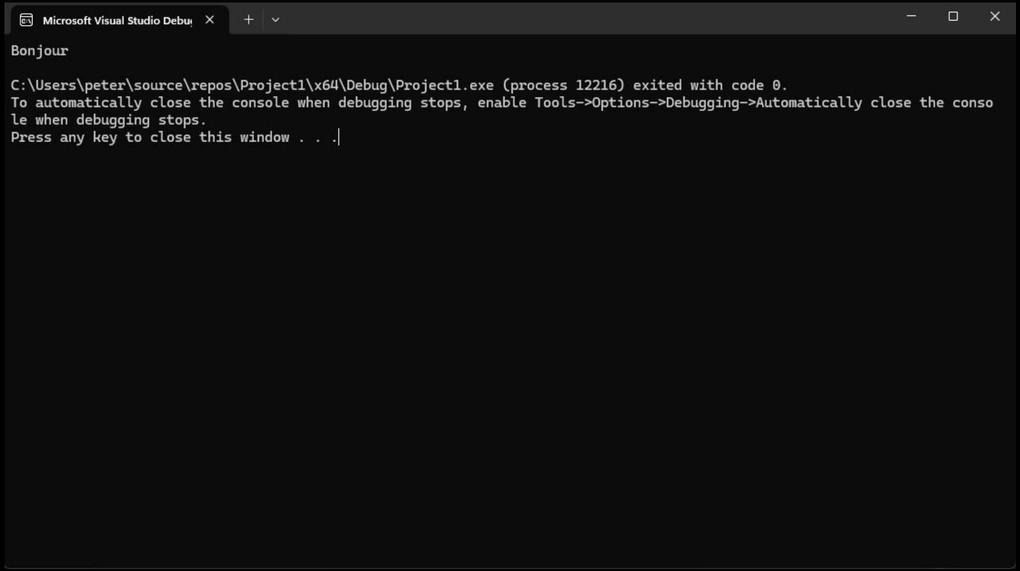
```
Build started...
1>------ Build started: Project: Project1, Configuration: Debug x64 ------
1>main.c
1>Project1.vcxproj -> C:\Users\peter\source\repos\Project1\x64\Debug\Project1.exe
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
========== Build started at 07:45 and took 00,782 seconds ==========
```

Aide :
Lancer le programme



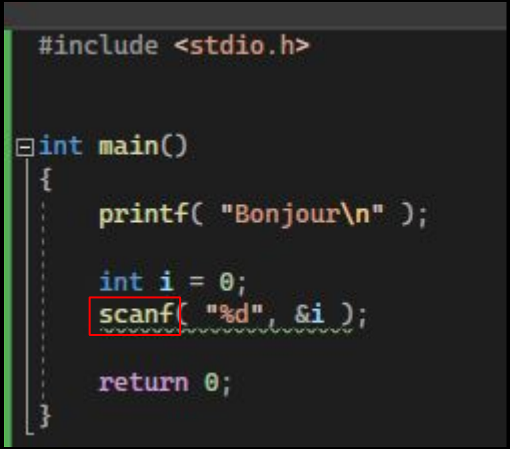
```c
#include <stdio.h>


int main()
{
    printf( "Bonjour\n" );

    return 0;
}
```

Aide :
Lancer le programme



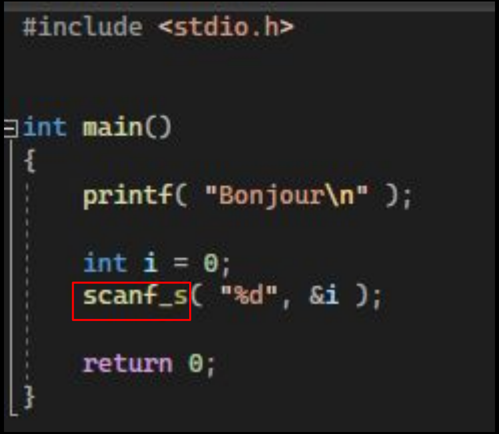
```
Bonjour

C:\Users\peter\source\repos\Project1\x64\Debug\Project1.exe (process 12216) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the conso
le when debugging stops.
Press any key to close this window . . .
```

Aide : scanf



```c
#include <stdio.h>


int main()
{
    printf( "Bonjour\n" );

    int i = 0;
    scanf( "%d", &i );

    return 0;
}
```

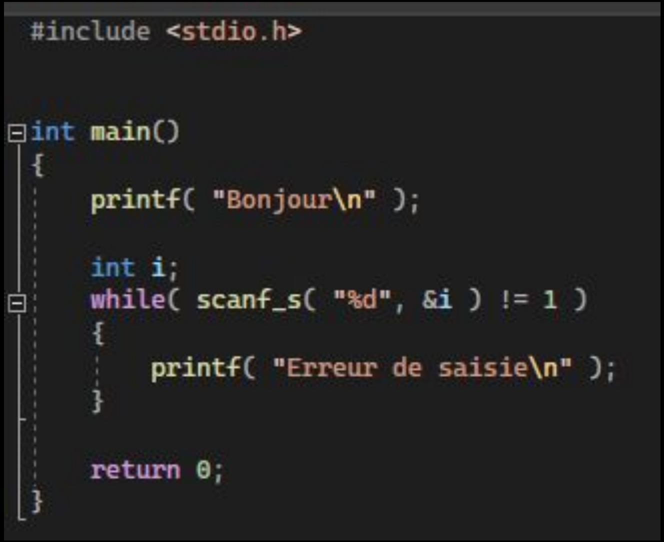
✗



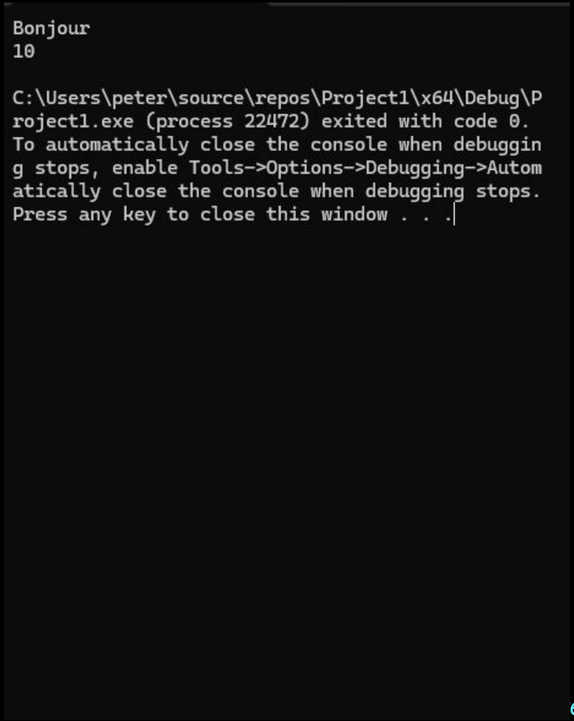
```c
#include <stdio.h>


int main()
{
    printf( "Bonjour\n" );

    int i = 0;
    scanf_s( "%d", &i );

    return 0;
}
```

✓

Aide : scanf

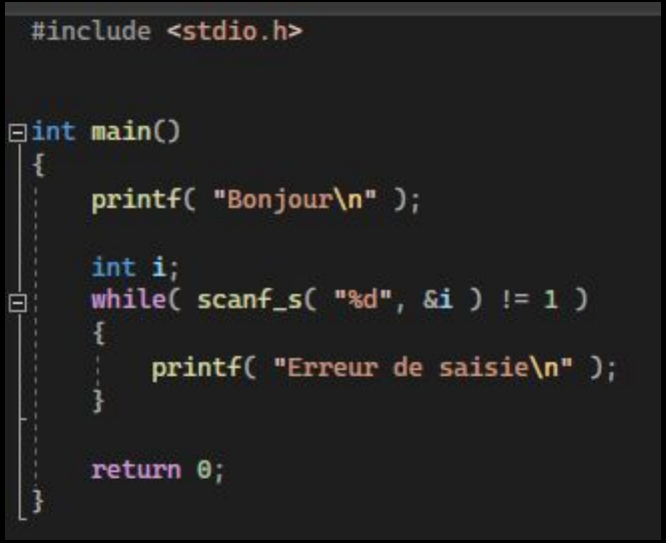


```c
#include <stdio.h>


int main()
{
    printf( "Bonjour\n" );

    int i;
    while( scanf_s( "%d", &i ) != 1 )
    {
        printf( "Erreur de saisie\n" );
    }

    return 0;
}
```

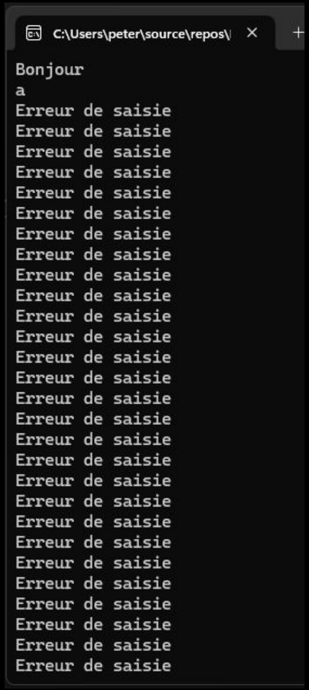


```
Bonjour
10

C:\Users\peter\source\repos\Project1\x64\Debug\P
roject1.exe (process 22472) exited with code 0.
To automatically close the console when debuggin
g stops, enable Tools->Options->Debugging->Autom
atically close the console when debugging stops.
Press any key to close this window . . .
```

Aide : scanf



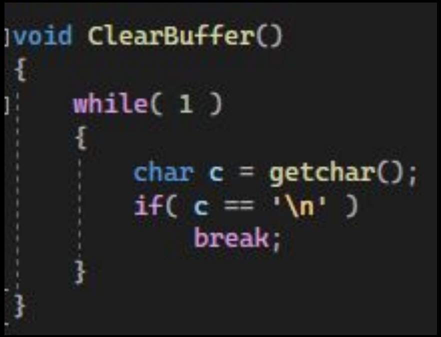
```c
#include <stdio.h>


int main()
{
    printf( "Bonjour\n" );

    int i;
    while( scanf_s( "%d", &i ) != 1 )
    {
        printf( "Erreur de saisie\n" );
    }

    return 0;
}
```



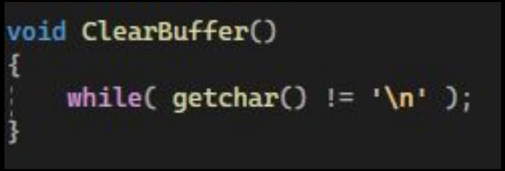
```
Bonjour
a
Erreur de saisie
Erreur de saisie
Erreur de saisie
Erreur de saisie
Erreur de saisie
Erreur de saisie
Erreur de saisie
Erreur de saisie
Erreur de saisie
Erreur de saisie
Erreur de saisie
Erreur de saisie
Erreur de saisie
Erreur de saisie
Erreur de saisie
Erreur de saisie
Erreur de saisie
Erreur de saisie
Erreur de saisie
Erreur de saisie
Erreur de saisie
Erreur de saisie
Erreur de saisie
Erreur de saisie
Erreur de saisie
Erreur de saisie
Erreur de saisie
Erreur de saisie
```

?

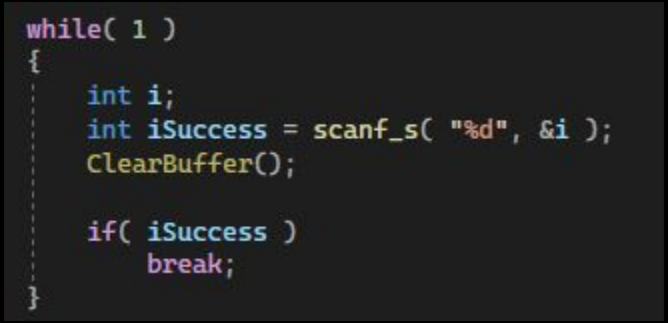
Aide : scanf



```c
void ClearBuffer()
{
	while( 1 )
	{
		char c = getchar();
		if( c == '\n' )
			break;
	}
}
```

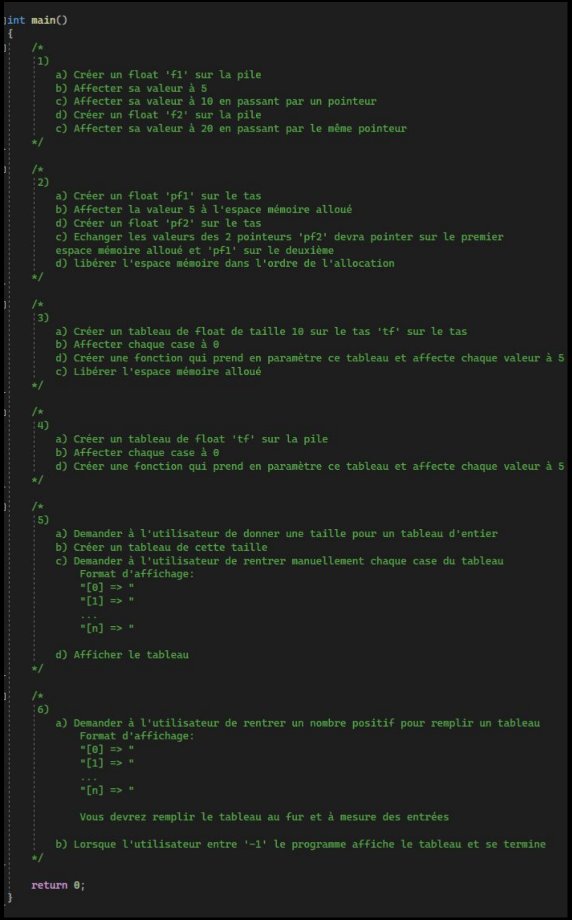


```c
void ClearBuffer()
{
	while( getchar() != '\n' );
}
```



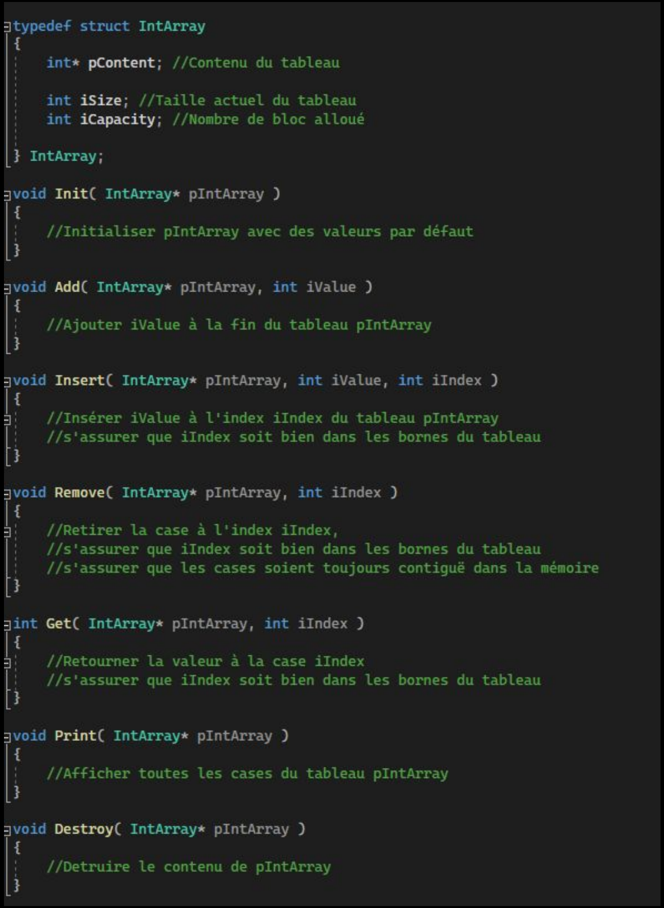
```c
while( 1 )
{
	int i;
	int iSuccess = scanf_s( "%d", &i );
	ClearBuffer();

	if( iSuccess )
		break;
}
```

Exercices: Pointeurs & Tableaux

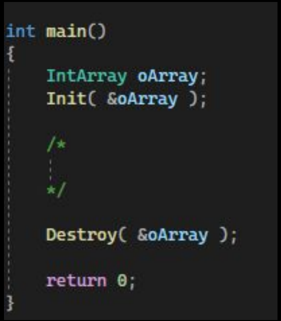


```c
int main()
{
    /*
    1)
        a) Créer un float 'f1' sur la pile
        b) Affecter sa valeur à 5
        c) Affecter sa valeur à 10 en passant par un pointeur
        d) Créer un float 'f2' sur la pile
        c) Affecter sa valeur à 20 en passant par le même pointeur
    */

    /*
    2)
        a) Créer un float 'pf1' sur le tas
        b) Affecter la valeur 5 à l'espace mémoire alloué
        d) Créer un float 'pf2' sur le tas
        c) Echanger les valeurs des 2 pointeurs 'pf2' devra pointer sur le premier
        espace mémoire alloué et 'pf1' sur le deuxième
        d) libérer l'espace mémoire dans l'ordre de l'allocation
    */

    /*
    3)
        a) Créer un tableau de float de taille 10 sur le tas 'tf' sur le tas
        b) Affecter chaque case à 0
        d) Créer une fonction qui prend en paramètre ce tableau et affecte chaque valeur à 5
        c) Libérer l'espace mémoire alloué
    */

    /*
    4)
        a) Créer un tableau de float 'tf' sur la pile
        b) Affecter chaque case à 0
        d) Créer une fonction qui prend en paramètre ce tableau et affecte chaque valeur à 5
    */

    /*
    5)
        a) Demander à l'utilisateur de donner une taille pour un tableau d'entier
        b) Créer un tableau de cette taille
        c) Demander à l'utilisateur de rentrer manuellement chaque case du tableau
            Format d'affichage:
            "[0] => "
            "[1] => "
            ...
            "[n] => "

        d) Afficher le tableau
    */

    /*
    6)
        a) Demander à l'utilisateur de rentrer un nombre positif pour remplir un tableau
            Format d'affichage:
            "[0] => "
            "[1] => "
            ...
            "[n] => "

            Vous devrez remplir le tableau au fur et à mesure des entrées

        b) Lorsque l'utilisateur entre '-1' le programme affiche le tableau et se termine
    */

    return 0;
}
```

Exercices: Tableaux dynamiques

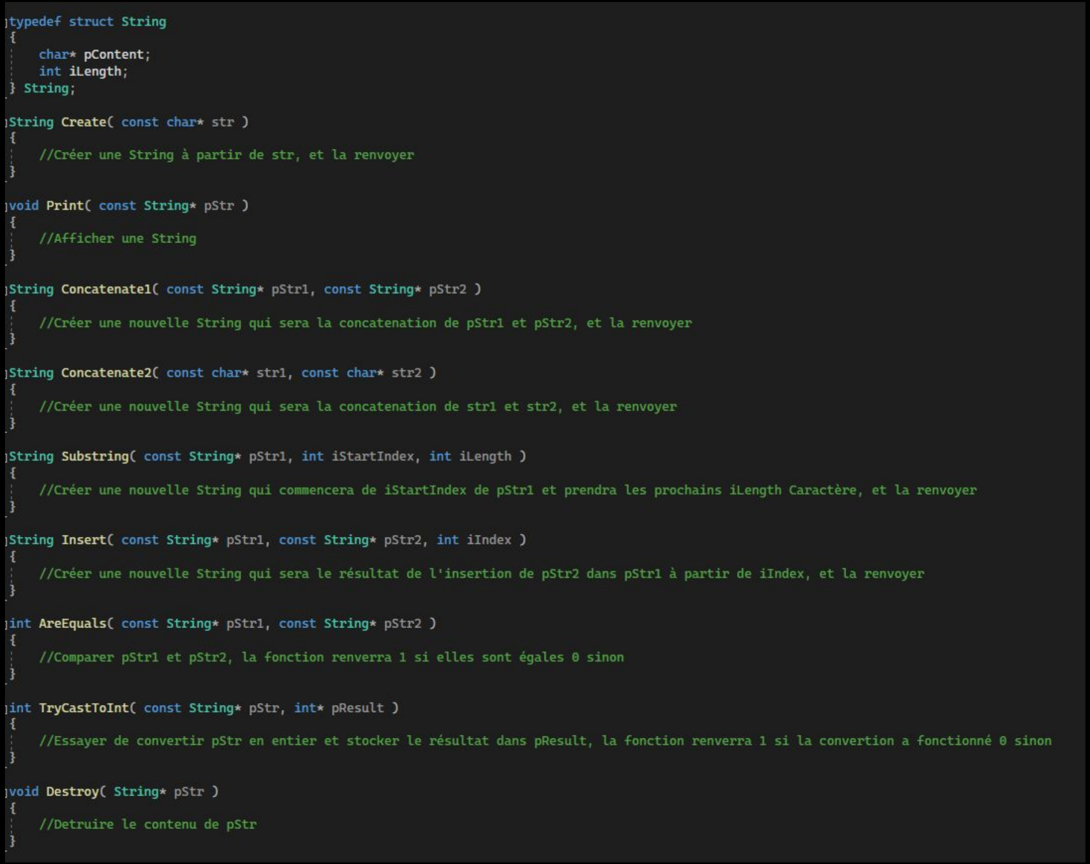


```c
typedef struct IntArray
{
    int* pContent; //Contenu du tableau

    int iSize; //Taille actuel du tableau
    int iCapacity; //Nombre de bloc alloué

} IntArray;

void Init( IntArray* pIntArray )
{
    //Initialiser pIntArray avec des valeurs par défaut
}

void Add( IntArray* pIntArray, int iValue )
{
    //Ajouter iValue à la fin du tableau pIntArray
}

void Insert( IntArray* pIntArray, int iValue, int iIndex )
{
    //Insérer iValue à l'index iIndex du tableau pIntArray
    //s'assurer que iIndex soit bien dans les bornes du tableau
}

void Remove( IntArray* pIntArray, int iIndex )
{
    //Retirer la case à l'index iIndex,
    //s'assurer que iIndex soit bien dans les bornes du tableau
    //s'assurer que les cases soient toujours contiguë dans la mémoire
}

int Get( IntArray* pIntArray, int iIndex )
{
    //Retourner la valeur à la case iIndex
    //s'assurer que iIndex soit bien dans les bornes du tableau
}

void Print( IntArray* pIntArray )
{
    //Afficher toutes les cases du tableau pIntArray
}

void Destroy( IntArray* pIntArray )
{
    //Detruire le contenu de pIntArray
}
```

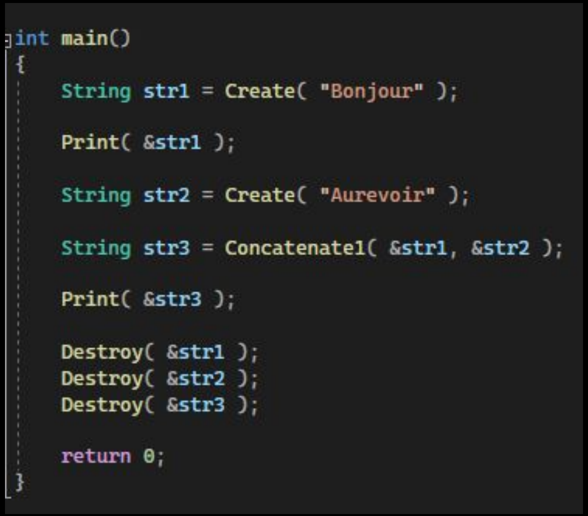


```c
int main()
{
    IntArray oArray;
    Init( &oArray );

    /*

    */

    Destroy( &oArray );

    return 0;
}
```

Exercices: Le type String

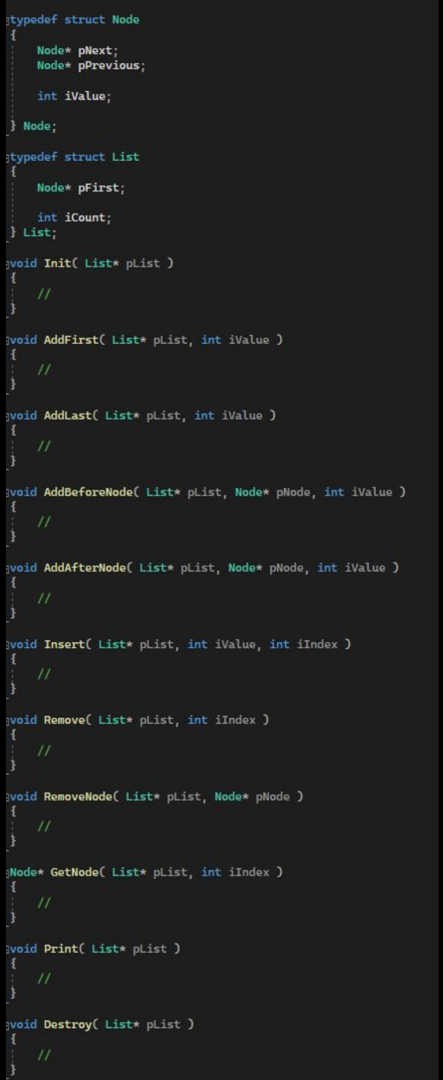


```c
typedef struct String
{
    char* pContent;
    int iLength;
} String;

String Create( const char* str )
{
    //Créer une String à partir de str, et la renvoyer
}

void Print( const String* pStr )
{
    //Afficher une String
}

String Concatenate1( const String* pStr1, const String* pStr2 )
{
    //Créer une nouvelle String qui sera la concatenation de pStr1 et pStr2, et la renvoyer
}

String Concatenate2( const char* str1, const char* str2 )
{
    //Créer une nouvelle String qui sera la concatenation de str1 et str2, et la renvoyer
}

String Substring( const String* pStr1, int iStartIndex, int iLength )
{
    //Créer une nouvelle String qui commencera de iStartIndex de pStr1 et prendra les prochains iLength Caractère, et la renvoyer
}

String Insert( const String* pStr1, const String* pStr2, int iIndex )
{
    //Créer une nouvelle String qui sera le résultat de l'insertion de pStr2 dans pStr1 à partir de iIndex, et la renvoyer
}

int AreEquals( const String* pStr1, const String* pStr2 )
{
    //Comparer pStr1 et pStr2, la fonction renverra 1 si elles sont égales 0 sinon
}

int TryCastToInt( const String* pStr, int* pResult )
{
    //Essayer de convertir pStr en entier et stocker le résultat dans pResult, la fonction renverra 1 si la convertion a fonctionné 0 sinon
}

void Destroy( String* pStr )
{
    //Detruire le contenu de pStr
}
```

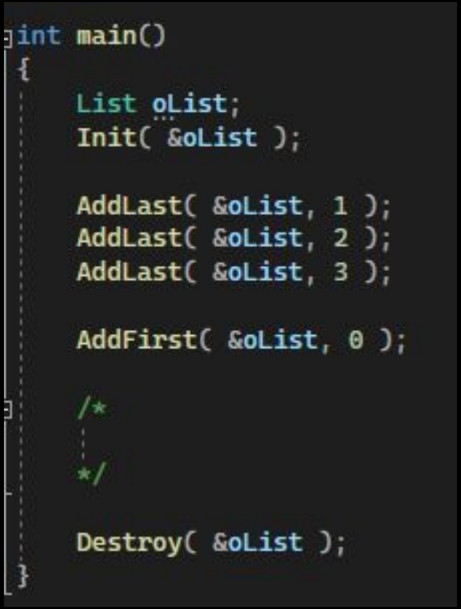


```c
int main()
{
    String str1 = Create( "Bonjour" );

    Print( &str1 );

    String str2 = Create( "Aurevoir" );

    String str3 = Concatenate1( &str1, &str2 );

    Print( &str3 );

    Destroy( &str1 );
    Destroy( &str2 );
    Destroy( &str3 );

    return 0;
}
```

Exercices: Les listes



```c
typedef struct Node
{
    Node* pNext;
    Node* pPrevious;

    int iValue;

} Node;

typedef struct List
{
    Node* pFirst;

    int iCount;
} List;

void Init( List* pList )
{
    //
}

void AddFirst( List* pList, int iValue )
{
    //
}

void AddLast( List* pList, int iValue )
{
    //
}

void AddBeforeNode( List* pList, Node* pNode, int iValue )
{
    //
}

void AddAfterNode( List* pList, Node* pNode, int iValue )
{
    //
}

void Insert( List* pList, int iValue, int iIndex )
{
    //
}

void Remove( List* pList, int iIndex )
{
    //
}

void RemoveNode( List* pList, Node* pNode )
{
    //
}

Node* GetNode( List* pList, int iIndex )
{
    //
}

void Print( List* pList )
{
    //
}

void Destroy( List* pList )
{
    //
}
```



```c
int main()
{
    List oList;
    Init( &oList );

    AddLast( &oList, 1 );
    AddLast( &oList, 2 );
    AddLast( &oList, 3 );

    AddFirst( &oList, 0 );

    /*

    */

    Destroy( &oList );
}
```