Mots clés du langage

— Les types de données : char, double, float, int, long, short, void.
— Spécifieurs supplémentaires de type : auto/register, const, extern/static, inline, signed/unsigned.
— Boucles : do-while, for, goto, while.
— Branchements (conditionnels ou non) : break/continue, if/else, return, switch/case.
— Création de nouveaux types : enum, struct, typedef, union.
— Autres : main (fonction principale), sizeof (opérateur unaire).
— Spécifiques au C++ : bool, class, delete/new, false/true, private/protected/public, this, virtual.

Nommer ses variables et/ou fonctions et/ou structures ...

— Les caractères permis sont : les caractètes alphanumériques non accentués + le _ (prononcez underscore et non tiret du 8).
— Le nom commence par : une lettre ou un underscore.
— Les minuscules et majuscules sont différenciés.
— Voir les conventions de nommage selon l'objet nommé et le langage.
— Exemples : En une lettre, généralement a, b, c, d, ... pour des constantes ; i, j, k, l, ... pour des variables de boucle ; x, y, z, w pour des coordonnées. En plusieurs lettres/chiffres, pi (OK), 2pi (non-OK), _2pi (OK), pi_2 (OK, pi sur 2), factorielle, somme, somme_des_carres, sommeDesCarres, ...

Instruction / Portées / Déclarations / Définitions

— Une instruction désigne une ou plusieurs actions à faire exécuter par la machine. Elle se termine par un point-virgule. On peut par exemple demander à créer une variable tout en l'initialisant. Une instruction peut comporter plusieurs sous-instructions séparés par des opérateurs tels que la virgule (séquence inconditionnelle), le ET ou le OU logiques (séq. conditionnelle).
— La portée d'un élément est l'espace dans lequel ce dernier est défini. La portée commence à la position à laquelle l'élément a été déclaré (ou prototypé) et n'est valable qu'au sein du bloque (accolade ouvrante – accolade fermante) où la déclaration a eu lieu. Si un élément est déclaré en dehors de tout bloque, la portée est au moins valable sur le reste du fichier source (voir `extern` pour des portées allant au-delà du fichier). Voir un exemple de *variable shadowing* :

```
1  #include <stdio.h>
2  int main(void) {
3    int a = 42;
4    {
5      int a = 0;
6      printf("i = %d\n", a);
7    }
8    printf("i = %d\n", a);
9    return 0;
10 }
```

— Déclarer/Définir une variable :

```
[specifiers] <type> nom_de_la_variable [ = ... ];
```

Lors d'une simple déclaration, aucune valeur ne doit être affectée à la variable. Par ailleurs, une déclaration vaut définition sans valeur initiale (voir cas particulier des variables `extern`).
— Déclarer (prototyper) une fonction :

```
[specifiers] <type> nom_de_la_fonction(<type1[, type2[, ...]] | void>);
```

— Définir une fonction :

```
[specifiers] <type> nom_de_la_fonction(<type1 nom1[, type2 nom2[, ...]] | void>) {
      [ instructions; ... ]
}
```

Type	Occupation mémoire	Plage de valeurs
char	1 octet	-128 à 127
unsigned char	1 octet	0 à 255
int	2 ou 4 octets	Selon l'architecture
unsigned int	2 ou 4 octets	Selon l'architecture
short	2 octets	-32768 à 32767
unsigned short	2 octets	0 à 65535
long	4 octets	-2147483648 à 2147483647
unsigned long	4 octets	0 à 4294967295
long long	8 octets	-2^{63} à $2^{63}-1$
unsigned long long	8 octets	0 à $2^{64}-1$
Type à virgule flottante		
float	4 octets	3.4×10^{-38} à 3.4×10^{38} (IEEE 754)
double	8 octets	1.7×10^{-308} à 1.7×10^{308} (IEEE 754)
long double	10 octets	3.4×10^{-4932} à 3.4×10^{4932} (IEEE 754)

TABLE 1 – Types de données élémentaires

Format	Type de donnée	
%d	Entier	`short, int`
%i	Entier	`short, int`
%u	Entier non signé	`unsigned short, unsigned int`
%o	Entier octal	`short, int`
%x, %X	Entier hexadécimal	`short, int`
%l, %ld, %li, %lu, %lo, %lx, %lX	Entier long	`long`
%lld, %lli, %llu, %llo, %llx, %llX	Entier long 64 bits	`long long`
%c	Caractère ASCII	`char, unsigned char`
%f, %F, %g, %G	Flottant	`float, double`
%e, %E	Flottant (exponentiel)	`float, double`
%Lf, %LF, %Lg, %LG, %Le, %LE	long double	`long double`
%s	Chaîne de caractères	`char *`
%p	Pointeur	`<type> *`

TABLE 2 – (Quelques) Formatages de la fonction `printf`

Séquence d'échappement	Action
\n	Nouvelle ligne (`new line`)
\t	Tabulation horizontale
\v	Tabulation verticale
\b	Retour d'un caractère arrière (`backspace`)
\r	Retour chariot (`carriage return`)
\f	Saut de page (`form feed`)
\a	Signal sonore (`alarm`)
\'	Affiche une apostrophe
\"	Affiche un guillemet
\\	Affiche un `Backslash`
\ddd	Affiche les codes ASCII en octale
\xddd	Affiche les codes ASCII en hexadécimale

TABLE 3 – Séquences d'échappement

Les branchements conditionnels et les boucles

Faire en cours (live) :

— un exemple avec if, if/else et test en ligne :

```
/* ceci est un commentaire en C */
// ceci est un commentaire en C++
/* if simple */
if(<condition : expression à évaluer>)
   <instruction ou bloque d'instructions à exécuter si condition vraie>
/* if/else */
if(<condition>)
   <instruction ou bloque d'instructions à exécuter si condition vraie>
else
   <instruction ou bloque d'instructions à exécuter sinon>
/* test en ligne (généralement le résultat est affecté quelque part) */
<condition> ? <valeur si vraie> : <valeur sinon>;
```

— un exemple avec switch/case :

```
/* le switch a pour utilité de remplacer plusieurs if/else if à la suite */
/* pour un élément testé A */
if(A == valeur0) {
   [faire ... 0]
} else if (A == valeur1) {
   [faire ... 1]
} else if (A == valeur2) {
   [faire ... 2]
} else /* sinon simple, cas restants */ {
   [faire ... autres]
}
/* à l'aide d'un switch devient */
switch(A) {
  case valeur0:
    [faire ... 0]
    break;
  case valeur1:
    [faire ... 1]
    break;
  case valeur2:
    [faire ... 2]
    break;
  default:
    [faire ... autres]
    break;
}
```

— un exemple (boucle) avec for :

```
for([faire avant déb. for] ; [continuer tant que vraie] ; [faire en fin de chaque itération])
   <instruction ou bloque à exécuter dans la boucle>
```

— un exemple (boucle) avec while :

```
while(<faire tant que vraie>)
   <instruction ou bloque à exécuter dans la boucle>
```

— un exemple (boucle) avec do-while :

```
do { // faire au moins une fois
   <instruction ou bloque à exécuter dans la boucle>
} while(<refaire tant que vraie>);
```

```
1  #include <stdio.h>
2  int main(void) {
3    printf("Hello World\n");
4    return 0;
5  }
```

FIGURE 1 – Premier programme

```
1  #include <stdio.h>
2  /* Ceci est un commentaire */
3  void p01(void) {
4    printf("Hello World\n");
5  }
6  void p02(void) {
7    char chaine[] = "Hello World\n";
8    int i;
9    for(i = 0; i < 12; i++)
10     putchar(chaine[i]);
11 }
12 void p03(char ch[]) {
13   printf("%s", ch);
14 }
15 void p04(char ch[]) {
16   while(*ch != '\0')
17     putchar(*ch++);
18 }
19 int main(void) { /* Main est la fonction principale */
20   char chaine[] = "Hello World\n";
21   p01();
22   p02();
23   p03(chaine);
24   p04(chaine);
25   return 0;
26 }
```

FIGURE 2 – Différentes méthodes pour imprimer une chaîne de caractères

```
1  main() {
2    int i;
3    char num[] = "123456"; /* Mettez votre numero d'etudiant a la place */
4    for(i = 0; num[i] != '\0'; i++)
5      putchar('a' + num[i] - '0');
6    putchar('\n');
7  }
```

FIGURE 3 – Que fait ce programme ? (exo01.c)

Exercices :

1. Écrire les fonctions :
   - `int prod(int n);` **retournant** le produit des n premiers entiers strictement positifs ;
   - `int somme(int n);` **retournant** la somme des n premiers entiers ;
   - `int prodp(int n)` **retournant** le produit des n premiers entiers strictement positifs et pairs ;
   - `int sommep(int n)` **retournant** la somme des n premiers entiers pairs ;
   - `int prodi(int n)` **retournant** le produit des n premiers entiers impairs ;
   - `int sommei(int n)` **retournant** la somme des n premiers entiers impairs ;
   - `int puissance(int x, int n)` **retournant** x à la puissance n.

   Pour faire simple, créer et utiliser ces fonctions dans un même programme **affichant par la suite** le retour de chacune d'entre-elles en utilisant la fonction `printf`.

Plus loin

Récursivité ? Quand une fonction s'appelle elle-même :

```
// n! = n x (n - 1)
// 2 cas particuliers :
//   0! = 1
//   1! = 1
int factorielle(int n) {
   if(n < 2)
      return 1;
   return n * factorielle(n - 1);
}
```

```
1  #include <stdio.h>
2  unsigned long fact_term(unsigned long n, unsigned long s) {
3    if(n < 2) return s;
4    return fact_term(n - 1, n * s);
5  }
6  unsigned long fib(unsigned long n) {
7    if(n < 1) return 0;
8    if(n == 1 || n == 2) return 1;
9    return fib(n - 1) + fib(n - 2);
10 }
11 int main(void) {
12   unsigned long i;
13   for(i = 0; i <= 12; i++)
14         printf("%lu! = %lu\n", i, fact_term(i, 1));
15   for(i = 0; i <= 47; i++)
16         printf("fib(%lu) = %lu\n", i, fib(i, 1, 1));
17   return 0;
18 }
```

FIGURE 4 – Factorielle en récursivité terminale et Fibonacci en récursivité simple (`factnfib.c`).

```
1  #include <stdio.h>
2  int main(void) {
3    unsigned char c = 'A', i;
4    for (i = 128; i != 0; i = i / 2)
5      if (c & i) putchar('1');
6      else putchar('0');
7    putchar('\n');
8    return 0;
9  }
```

```
1  #include <stdio.h>        /* Pour les fonctions printf, fprintf et putchar */
2  #include <stdlib.h>       /* Pour la fonction atol, strtoul ... */
3  int main(int argc, char ** argv) {
4    unsigned long v = 0, i, j;
5    if(argc != 2) {
6      fprintf(stderr, "usage : %s <nombre a convertir>\n", argv[0]);
7      exit(1);
8    }
9    j = 1UL << ( (sizeof(v) << 3) - 1);
10   v = strtoul(argv[1], NULL, 10);
11   printf("Le nombre %lu s'ecrit : ", v);
12   for (i = 0; i < (sizeof(v) << 3); i++, j >>= 1) {
13     if (v & j) putchar('1');
14     else putchar('0');
15   }
16   printf(" en binaire\n");
17   return 0;
18 }
```

FIGURE 5 – Conversion d'un décimal en binaire (`dec2bin.c`).

Exercices :

- Écrire une fonction itérative pour calculer Factorielle ;
- Écrire une fonction itérative pour calculer Fibonacci ;

— Écrire les deux fonctions `somme_i` (itérative) et `somme_rt` (récursivité terminale) qui calculent la somme des n premiers entiers ;
— Écrire la fonction `fib_rt` qui calcule Fibonacci en récursivité terminale ;
— Modifier `dec2bin.c` (cf. FIG.5) afin d'obtenir une impression en base octale (8) ;
— Modifier `dec2bin.c` (cf. FIG.5) afin d'obtenir une impression en base hexadécimale (16) ;