Résumé : Algorithmique → Langage C - Exemples

Algorithme principal / fonction principal d'un programme C

Algorithme Principal *Déclarations* *Début* *Fin*	`// directives` `int main ()` `{` `  // déclarations` `  // début` `  // fin` `  return 0 ;` `}`

Identifiants (id)

- Commencent par une lettre, comporte ensuite éventuellement des lettres, des chiffres ou « _ »

Fonction / fonction d'un programme C

Fonction id (décl. param.) : type *Déclarations* *Début* *retourner expr* *Fin*	`type id (décl.param.)` `{` `  // déclarations` `  // traitement` `  return expr ;` `}`

Procédure / fonction sans retour d'un programme C

Procédure id (décl. param.) *Déclarations* *Début* *Fin*	`void id (décl. param.)` `{` `  // déclarations` `  // traitement` `  [return;] // quitte` `}`

Types de données (type)

entier	`[signed / unsigned] char` `[signed / unsigned] short` `[signed / unsigned] int` `[signed / unsigned] long`
réel	`float, double`
caractère	`char`

Types basés sur des types de base

chaine(20)	`char[20]` *(cf. tableaux)*
booléen	`bool`

Valeurs littérales

entier	`-100, 2013` `0x`hexa, `0`octal
réel	`3.14, 3f, 1.5E-12`
caractère	`'c', '\n'`
chaine	`"bonjour à tous !"`
booléen (vrai, faux)	`true, false`

Déclarations de constantes et variables

const[ante] ID : type ← expr	`#define ID (expr)` `const type ID = expr;`
var[iable] id : type	`type id ;`

Résumé : Algorithmique → Langage C - Exemples

Affectation

id_var ← expr	`id_var = expr ;`

Expressions et opérateurs (expr)

numérique	*+, -, *, /, mod*	`+, -, *, /, %`
logique	*=, !=, <, <=, >, >=*	`=, !=, <, <=, >, >=`
logique	*et, ou, non*	`&&, ||, !`

Structures de contrôle conditionnelles

si expr_logique *Alors…* *finSi*	`if (exp_logique)` `{ … ; }`
si expr_logique *Alors…* *Sinon…* *finSi*	`if (exp_logique)` `{ … ; }` `else` `{ … ; }`
Selon expr *cas valeur1 : …* *cas valeurN : …* *cas sinon : …* *finSelon*	`switch (expr_numérique)` `{` `case valeur1 : … ; [break ;]` `case valeurN : … ; break ;` `default : …; break;` `}`

Structures de contrôle itératives

tantque expr_logique *…* *finTantque*	`while (expr_logique)` `{` `… ;` `}`
reperter *…* *jsuqu'à expr_logique*	`do` `{` `… ;` `} while (!expr_logique);`
pour v de x a y pas de n *finPour*	`for(v=x ;v<=y ;v=v+n)` `{` `…;` `}`

Autres fonctions des bibliothèques : entrées/sorties, nombres aléatoires

ecrire expr	`printf("…%format…",expr) ; // stdio`
ecrire « chaine »	`puts("chaine") ; // stdio`
lire v	`scanf("%format", &v) ; // stdio`
lire c	`c = getchar() ;` `while(getchar() != '\n'); // vider le buffer`
entier *reel* *caractère* *chaine*	`%i, %d , %ld, %u, %lu (autre exemple : %6d)` `%f, %lf, %e, %E (autre exemple : %3.2f)` `%c` `%s`
alea(1,10)	`srand((unsigned) time(NULL)); // stdlib et time` `nb = rand()%10 + 1; // stdlib`

Résumé : Algorithmique → Langage C - Exemples

Structures de données complexes (1) – tableau

Tableau de N éléments

var[iable] id : tableau de N type	`type id[N] ;`

Tableau de NxM éléments

var[iable] id : tableau de NXM type	`type id[N][M] ;`

Structures de données complexes (2) – structures

type *structure id* *déclaration variables* *finStructure*	**typedef** **struct** id **{** *déclaration des variables ;* **}** id ;

Résumé : Algorithmique → Langage C - Exemples

Fichiers texte

Ecriture d'un fichier

var pf : FICHIER *var i : ENTIER* *pf ← OUVRIR* *« monFichier1.txt » en ECRITURE* *POUR i DE 1 à 10* *ECRIRE pf, i, «», « essai », i* *FINPOUR* *FERMER pf*	`#include <stdio.h>` `int main()` `{` `FILE * pf = NULL;` `int i ;` `pf = fopen("monfichier1.txt","w");` `for(i=1;i<=10;i++)` `{` `fprintf(pf,"%d %s%d\n",i,"essai",i);` `}` `fclose(pf);` `pf = NULL;` `return 0` `}`

Lire un fichier texte par caractères

```
#include <stdio.h>
int main()
{
 FILE * pf = NULL;
 int n = 0;
 char c;

 pf = fopen("monfichier1.txt","r");
 if (pf==NULL)
  perror ("Erreur d'ouverture !");
  else
  {
   while ((c=fgetc(pf)) != EOF)
   {
     putchar(c);
     n++;
   }
   puts ("fin de fichier !.");
   printf ("%d octets lus", n);
   fclose (pf);
   pf = NULL ;
  }

 return 0;
}
```

Résumé : Algorithmique → Langage C - Exemples

Lire un fichier texte ligne par ligne

```
var pf : FICHIER
var n : ENTIER ← 0
var ch : CHAINE

pf ← OUVRIR
« monFichier1.txt » en
LECTURE

SI ERREUR_FICHIER pf
 ALORS
  ECRIRE « erreur »
 SINON
  LIRE_FICHIER pf, ch
  TANTQUE NON
FIN_DE_FICHIER pf
   ECRIRE ch
   LIRE_FICHIER pf, ch
   n ← n + 1
  FINTANTQUE
  ECRIRE n, « lignes lues »
  FERMER pf
FINSI
```

```c
#include <stdio.h>
int main()
{
 FILE * pf = NULL;
 int n = 0;
 char ch[2000];


 pf = fopen("monfichier1.txt","r");


 if (pf==NULL)
  perror ("Erreur d'ouverture !");
  else
  {
   fgets(ch, sizeof ch, pf);
   while (!feof(pf))
   {
     puts(ch);
     fgets(ch, sizeof ch, pf);
     n++;
   }

   printf ("%d lignes lues", n);
   fclose (pf);
   pf = NULL;
  }

 return 0;
}
```

Résumé : Algorithmique → Langage C - Exemples

Lire un fichier texte formaté

var pf : FICHIER *var n : ENTIER ← 0* *var nom : CHAINE* *var num : ENTIER*	`#include <stdio.h>` `int main()` `{` `  FILE * pf = NULL;` `  int n = 0;` `  int num;` `  char nom[2000];`
pf ← OUVRIR « monFichier1.txt » en LECTURE	`  pf = fopen("monfichier1.txt","r");`
SI ERREUR_FICHIER pf *ALORS* *ECRIRE « erreur »* *SINON* *LIRE_FICHIER pf, num, nom* *TANTQUE NON FIN_DE_FICHIER pf* *ECRIRE num, ch* *LIRE_FICHIER id, num, nom* *n ← n + 1* *FINTANTQUE* *ECRIRE n, « lignes lues »* *FERMER pf* *FINSI*	`  if (pf==NULL)` `   perror ("Erreur d'ouverture !");` `   else` `   {` `    fscanf(pf, "%d %s",&num,nom) ;` `    while (!feof(pf))` `    {` `      printf("\nEtudiant num %d : %s",num,nom);` `      n++;` `      fscanf(pf, "%d %s",&num,nom);` `    }` `    printf ("%d lignes lues", n);` `    fclose (pf);` `    pf = NULL;` `   }` `  return 0;` `}`

Résumé : Algorithmique → Langage C - Exemples

Ecriture, lecture (gestion d'erreur supprimée pour simplifier la lecture)

```
structure Agenda
   num : ENTIER;
   nom : CHAINE (20)
finStructure

DECLARATION
var pf : FICHIER
var nomf : CHAINE ←
« monFichier1.dat »
var num : ENTIER
var nom : CHAINE(20)
var ag : Agenda

DEBUT
pf← OUVRIR nomf en
ECRITURE BINAIRE

ECRIRE « numero »
LIRE ag.num
TANTQUE ag.num <> 0
   ECRIRE « nom »
   LIRE ag.nom
   ECRIRE_FICHIER pf, ag
   ECRIRE « autre num ? »
   LIRE ag.num
 finTANTQUE
FERMER pf
FINSI

pf← OUVRIR nomf en
LECTURE BINAIRE
LIRE_FICHIER pf, ag
TANTQUE NON
FIN_DE_FICHIER(
   ECRIRE ag.num, ag.nom
   LIRE_FICHIER pf, ag
finTANTQUE
FERMER pf
FINSI

FIN
```

```c
#include <stdio.h>

typedef struct Agenda {
    int num;
    char nom[20];
} Agenda;

int main()
{
 FILE * pf = NULL;
 char nomf [] = "monfichier1.dat";
 int num;
 char nom[20];
 Agenda ag ;


                                  // ECRITURE
 pf = fopen(nomf,"wb");


 printf("numéro ?");
 scanf("%d",ag.num);
 while (ag.num != 0)
 {
   printf("nom (sans espaces) ?");
   scanf("%s",&ag.nom);
   fwrite(&ag,sizeof(Agenda),1,pf);

   printf("autre numéro ? (0 = fin)");
   scanf("%d",ag.num);
 }
 fclose(pf);




                                 // LECTURE
 fichier = fopen(nomf,"rb");
 fread(&ag,sizeof(Agenda),1,pf);
 while(!feof(pf))
 {
   printf("\n %d %s %ld",ag.num, ag.nom);
   fread(&ag,sizeof(Agenda),1,pf);
 }
 fclose(pf);

return 0;
}
```

Résumé : Algorithmique → Langage C - Exemples

Mise à jour en fonction d'un numéro

```
structure Agenda
   num : ENTIER;
   nom : CHAINE (20)
finStructure

DECLARATION
var pf : FICHIER
var nomf : CHAINE ←
« monFichier1.dat »
var num : ENTIER
var ag : Agenda

DEBUT
pf← OUVRIR nomf en
ECRITURE BINAIRE

ECRIRE « numero »
LIRE num
TANTQUE num <> 0
  POSITIONNER_FICHIER pf,
num
   LIRE_FICHIER pf, ag
    ECRIRE « nouv.nom »
    LIRE ag.nom
    ECRIRE_FICHIER pf, ag
    ECRIRE « autre num ? »
    LIRE num
 finTANTQUE
FERMER pf
FINSI

FIN
```

```c
#include <stdio.h>

typedef struct Agenda {
    int num;
    char nom[20];
} Agenda;

int main()
{
 FILE * pf = NULL;
 char nomf [] = "monfichier1.dat";
 int num;
 char nom[20];
 Agenda ag ;


                                  // ECRITURE
 pf = fopen(nomf,"wb");

printf("no ?");
 scanf("%d",&num);
 while (num!=0)
 {
  res = fseek(fichier,((n-
1)*sizeof(Agenda)),SEEK_SET);
  if (res == 0)
     {
      fread(&ag,sizeof(Agenda),1,fichier);
      printf("%d %s",ag.num,ag.nom);
      printf("\nNouveau nom ?");
      scanf("%s",&ag.nom);
      fseek(fichier,((n-
1)*sizeof(Agenda)),SEEK_SET);

if(fwrite(&ag,sizeof(Agenda),1,fichier) !=
1) // 1 enregistrement écrit
        perror("erreur maj");
     printf("no ?");
     scanf("%d",&num);
     }
 }

 fclose(pf);

return 0;
}
```

Résumé : Algorithmique → Langage C - Exemples

Pointeurs

Pointeur sur type de base

```
ALGO testPointeur
VAR
   i, j : ENTIER
   ptr : POINTEUR VERS ENTIER

DEBUT
   i ← 10
   j ← 20

  ECRIRE ADRESSE de i, ADRESSE DE j, ADRESSE DE ptr

   ptr ← ADRESSE DE i
   ECRIRE ptr, DEREFERENCER ptr

   ptr ← ADRESSE DE j
   ECRIRE ptr, DEREFERENCER ptr
FIN
```

```
#include <stdio.h>
int main()
{

  int i, j;
  int * ptr = NULL;


  i = 10;
  j = 20;


  printf("%p %p %p\n",&i, &j, &ptr);


  ptr = & i;
  printf("%p %d\n",ptr, *ptr);


  ptr = & j;
  printf("%p %d\n",ptr, *ptr);


 return 0;
}
```

Pointeur sur type structure

```
ALGO testPointeur
TYPE
   STRUCTURE Point
     x : ENTIER
     y : ENTIER
   finSTRUCTURE
VAR
   p1 : Point
   ptr : POINTEUR VERS Point
DEBUT
   p1.x ← 5, p1.y ← 10
  ECRIRE ADRESSE de p1, ADRESSE DE ptr

   ptr ← ADRESSE DE p1
   ECRIRE ptr→x, ptr→y
   (ptr→x)←(ptr→x)+10
   (ptr→y)←(ptr→y)+20
   ECRIRE ptr→x, ptr→y
FIN
```

```
#include <stdio.h>
typedef struct Point
{
  int x, y;
} Point;

int main()
{
  Point p1 ;
  Point * ptr = NULL;

  p1.x = 5, p1.y = 10;

  printf("%p %p \n",&p1, &ptr);


  ptr = & p1;
  printf("%d %d\n",ptr->x, ptr->y);
  printf("%d %d\n",(*ptr).x, (*ptr).y);

  ptr->x+=10, ptr->y+=20 ;
  printf("%d %d\n",ptr->x, ptr->y);

 return 0;
```

	}

Sous-programmes FONCTIONS et modes de passage des paramètres

Les modes de passage des paramètres aux fonctions devraient toujours être par valeur ou par référence constante.

Sous-programmes PROCEDURES et modes de passage des paramètres

Paramètres en entrée / passage par valeur (ou par référence constante)

PROCEDURE afficherAge (ENTREE age : ENTIER) DEBUT ECRIRE « vous avez », age, « ans » FIN ALGO principal DECLARATIONS VARIABLES unAge : ENTIER DEBUT LIRE unAge afficherAge(unAge) FIN	#include <stdio.h> void afficherAge(const int age) { printf("vous avez %d ans", age); } int main() { int unAge; scanf("%d", &unAge); afficherAge(unAge); return 0; }

Paramètres en sortie

PROCEDURE calculerAge (ENTREE anneeNais, annee : ENTIER, SORTIE age : ENTIER) DEBUT age ← annee - anneeNais FIN ALGO principal DECLARATIONS VARIABLES an1, an2, ageCalc : ENTIER DEBUT LIRE an1, an2 calculerAge(an1, an2, ageCalc) afficherAge(ageCalc) FIN	#include <stdio.h> void afficherAge(const int age) { printf("vous avez %d ans", age); } void calculerAge(const int anneeNais, const int annee, int * age) { (*age) = annee - anneeNais ; } int main() { int an1, an2, ageCalc; scanf("%d %d", &an1, &an2); calculerAge(an1, an2, &ageCalc); afficherAge(ageCalc); return 0; }

Résumé : Algorithmique → Langage C - Exemples

Paramètres en entrée/sortie

PROCEDURE prochainAge *(ENTREE/SORTIE age : ENTIER)* *DEBUT* *age ← age + 1* *FIN* *ALGO principal* *DECLARATIONS* *VARIABLES unAge : ENTIER* *DEBUT* *LIRE unAge* *prochainAge(unAge)* *afficherAge(unAge)* *FIN*	`#include <stdio.h>` `void afficherAge(const int age)` `{` `  printf("vous aurez %d ans", age);` `}` `void prochainAge(int * age)` `{` `  (*age)++ ;` `}` `int main()` `{` `  int unAge;` `  scanf("%d", &unAge);` `  prochainAge(&unAge);` `  afficherAge(unAge);` `  return 0;` `}`