Algo	C/C++

Les bases

Un algorithme est une méthode
- suffisamment générale pour permettre de traiter toute une classe de problèmes ;
- combinant des opérations suffisamment simples pour être effectuées par une machine.

Trois étapes caractérisent la résolution d'un problème
- comprendre la nature du problème posé et préciser les données fournies ;
- préciser les résultats que l'on désire obtenir ;
- déterminer le processus de transformation des données en résultats.

Ces trois étapes ne sont pas indépendantes.

Un algorithme est une suite d'instructions élémentaires décrites dans un langage universel exécutées de manière séquentielle. Il est indépendant du langage de programmation.
Un langage de programmation est un langage commun entre machine et programmeur. Il implante ou réalise un algorithme.

Architecture d'un programme	
définition de l'ensembles des sous-programmes programme principal	directives de pré-compilation définition de l'ensembles des sous-programmes programme principal
Architecture de la fonction principale	

Début
…
Fin
	int main()
	{
	…;
	return 0;
	}

Une variable est le nom utilisé dans un programme pour faire référence à une donnée d'un certain type. Sa valeur peut varier au cours du temps contrairement aux constantes.

Déclaration des variables simples	
nom : entier *nom* : réel *nom* : booléen *nom* : caractère	int *nom*; ou short (plus court) float *nom*; ou double (plus long) bool *nom*; char *nom*;
Les constantes	
nom = *valeur*	const *type nom* = *valeur*;
L'affectation	
variable ← …	*variable* = …;
Les entrées et les sorties	
Saisir (… , …) Afficher (… , …)	cin >> … >> …; cout << … << …;

Opérateurs relationnels et logiques			
=	ou	==	||
≠	et	!=	&&
<	non	<	!
≤		<=	
>		>	
≥		>=	

Les commentaires peuvent s'insérer de deux manières différentes

- // *commentaire*
- /* *commentaires* */

Les structures de contrôle

Si

Si (*condition*) alors
    …
Sinon
    …
FinSi

```
if (condition)
{
    …;
}
else
{
    …;
}
```

Selon

Selon (*variable*) faire
    valeur : …
    …
    Autrement : …
FinSelon

```
switch (variable)
{
    case valeur :
        …;
        break;
    …
    default :
        …;
}
```

Pour

Pour *i* allant de … à … par pas de … faire
    …
FinPour

```
for (i = …; i …; i = …)
{
    …;
}
```

Tant que … faire

TantQue *condition* faire
    …
FinTantQue

```
while (condition)
{
    …;
}
```

Faire … tant que

Faire
    …
TantQue *condition*

```
do
{
    …;
}
while (condition);
```

Les sous-programmes

Les sous programmes sont des sous ensemble du programme.
Il existe deux types de sous-programmes :

- La fonction : retourne une unique variable d'un certain type ;
- La procédure : ne renvoie rien.

Les paramètres formels sont les variables utilisées lors de la déclaration du sous-programme. Ils reçoivent une valeur de l'extérieur.
Les paramètres effectifs sont les variables ou valeurs fournies lors de l'appel du sous-programme. C'est aussi la valeur renvoyée par une fonction.

Architecture d'une fonction	

Fonction *nom* (*nom* : *type*, …) : *type*
Préconditions : …
Données : *nom*, …
Résultat : *nom*
Description : …
Variables locales : *nom* : *type*, …
Début
…
retourner …
Fin *nom*
	type nom (*type nom*, …)
	{
	…;
	return …;
	}

Appel d'une fonction	
variable ← *nom* (*paramètres effectifs*) Afficher (*nom* (*paramètres effectifs*))	*variable* = *nom* (*paramètres effectifs*); cout << *nom* (*paramètres effectifs*);

Architecture d'une procédure	

Procédure *nom* (*nom* : *type*, …)
Préconditions : …
Données : *nom*, …
Description : …
Variables locales : *nom* : *type*, …
Début
…
Fin *nom*
	void *nom* (*type nom*, …)
	{
	…;
	}

Appel d'une procédure	
nom (*paramètres effectifs*)	*nom* (*paramètres effectifs*);

Passage par adresse : données/résultats

Le sous-programme peut accéder en mémoire à la valeur que le code appelant cherche à lui transmettre. Donc, contrairement au passage par valeur, le code appelant aura accès aux modifications faites sur la valeur dans le sous-programme.

Syntaxe du passage par adresse	
ajouter une ligne dans l'en-tête du sous-programme Données résultats : *nom*	ajouter une esperluette entre le type et le nom des paramètres formels : *type* & *nom*

Les tableaux

Un tableau est une structure de données qui contient une collection d'éléments de même type. Sa taille est fixe et définie lors de sa création.
Chaque élément a une position définie dans le tableau désignée par un indice : [i-1] désigne la $i^{ème}$ case du tableau.
Un tableau peut avoir plusieurs dimensions : [*1ère dimension*][*2ème dimension*]…
Comme le tableau est une variable complexe, il ne peut pas être retourné, il est automatiquement passé par adresse.

Déclaration de tableaux	
nom : tableau [*taille*]… de *type*	*type nom* [*taille*]…;

Les éléments d'un tableau sont manipulés individuellement. On ne peut pas faire d'opération sur un tableau entier. En revanche, chaque élément est une variable, toutes les opérations sont donc réalisables.
La seule opération faisable en C sur l'ensemble d'un tableau est l'initialisation en bloc :
type nom [*taille*]… = {*valeur*};

Les chaines de caractères

Les caractères sont codés par un numéro entre 0 et 255 (codage ASCII). On peut donc réaliser des opérations sur les caractères et les comparer.

Les chaines de caractères sont des tableaux de ce type se terminant par le caractère '\0'. Il faut donc une chaine de n+1 cases pour mettre n caractères. Un texte entre guillemets *"texte"* est aussi une chaine de caractère.
Les chaines de caractères sont de type complexe. Donc, comme les tableaux, elles ne se retournent pas et sont toujours en données/résultats.

Déclaration de chaines de caractères	
nom : chaine [*taille+1*] de caractères	char *nom*[taille+1];
Remplissage par l'utilisateur	
Saisir (*nom*)	cin >> *nom*;
Affichage	
Afficher (*nom*)	cout << *nom*;

La bibliothèque string.h contient de nombreux sous-programmes pour manipuler les chaines de caractères

- Concaténation de deux chaines :
  strcat (*chaine1*, *chaine2*);
  met la *chaine2* après la *chaine1* et rajoute un '\0' à la fin
- Comparaison de deux chaines :
  strcmp (*chaine1*, *chaine2*)
  renvoie 0 si les deux chaines sont identiques, 1 si *chaine1* > *chaine2* et -1 si *chaine1* < *chaine2*
- Copie :
  Strcpy (*chaine1*, *chaine2*);
  Copie *chaine2* dans *chaine1* et rajoute un '\0' à la fin

- Longueur d'une chaine :
  strlen (*chaine*);
  renvoie le nombre de caractères de la *chaine* (sans compter '\0')

Les structures

Les structures sont un type de variables complexes contenant plusieurs champs qui sont eux-mêmes de type simple ou complexe. La structure ainsi créée est un nouveau "type" de variable. Une variable de type structure est un enregistrement.
Comme les tableaux, les structures ne peuvent pas faire l'objet d'opérations ou de comparaisons cependant, les affectations fonctionnent.
Contrairement aux autres types complexes, une fonction peut retourner une structure.

Déclaration de structures	

Structure *nom*
nom : *type*
…
Fin structure
	struct *nom*
	{
	type nom;
	…
	};

Déclaration de variables de type structure	
nom_variable : *nom_structure*	*nom_structure nom_variable*;
Accès à un champ	
nom_variable.*nom_champ*	*nom_variable*.*nom_champ*;