Fiche n°1: Structure Générale d'un Programme
Types et opérateurs élémentaires

Structure d'un programme et déclarations

Algorithmique	Langage C#

// Commentaire sur une ligne
/* Commentaire sur plusieurs lignes*/
 	
PROGRAMME Exemple
 	
VAR
Déclaration des variables
 	
DEBUT
 	
Corps du programme
 	
FIN
	// commentaire sur une seule ligne
	/* commentaire sur plusieurs lignes */
	
	…
	static void Main(string[ ] args)
	{
	Déclaration des variables
	dans le corps du programme
	}
	
	Chaque instruction se termine par un ;

✍ ***Le langage C# différencie les majuscules des minuscules.*** Les mots-clé des instructions sont définis en minuscules. Exemple le mot-clé INT ne serait pas reconnu car seul int est connu . Une variable devra être utilisée dans un programme en respectant scrupuleusement la casse définie lors de sa déclaration. Exemple: la variable moy est différente de la variable Moy.	

Définition des types de données

Variable et Type Simple	
VAR a,b :**entier** c: **réel** x: **caractère** nom: **chaine** trouve : **booléen**	 **int** a,b ; sur 32 bits **double** d ; sur 64 bits **char** x ; **string** nom; **bool** trouve ; la valeur 0 équivaut à False
Constante	
CONST pi ✍ 3,1416 le type est facultatif (trouvé grâce à la valeur)	 **const** double pi = 3.1416 ; Le type est obligatoire
Les opérateurs élémentaires	
+ - * commun aux entiers et réels / division réelle DIV division entière (sans virgule) MOD reste de la division entière **Concaténation** (juxtaposition) pour les chaînes seulement : &	+ - * pour les entiers et réels / division - réelle si l'un des deux opérande est réel - entière si les deux opérandes sont entiers % reste de la division entière **+** avec des chaînes => concaténation

Fiche n°2: instructions élémentaires

Instructions élémentaires

Algorithmique	Langage C++
Ecirture à l'écran :AFFICHAGE	
Afficher expression ou **Aff** expression On peut afficher n'importe quel type d'expression. On peut même afficher plusieurs expressions à la suite, séparées par une virgule. **Afficher** "Voici A: ", a, " \net B ", b \n permet de passer à la ligne dans une chaîne.	**Console.WriteLine(**expression**);** avec retour à la ligne après l'affichage **Console.Write(**expression**);** sans retour à la ligne On peut afficher tout type d'expression Mais on ne peut pas afficher plusieurs expressions à la suite avec la même instruction. **Console.Write (**"Voici A: ") Console.WriteLine (a); Console.Write ("et B"); Console.WriteLine(b);
Lecture à partir du clavier : SAISIE	

Saisir ch
 	
On peut saisir plusieurs variables dans la même instruction
Saisir nb1, nb2
 	
Tout type de variable peut être saisi (sauf les booléens)
 	
En algorithmique, on ne se préoccupe généralement pas des problèmes de conversions, pour se concentrer sur la logique du traitement.
	string ch ;
	ch = Console.ReadLine();
	
	La méthode ReadLine ne permet de lire qu'une seule variable.
	
	On ne peut saisir directement qu'une variable de type string.
	Pour récupérer une valeur numérique saisie au clavier dans une variable numérique, il faut faire une conversion de la variable chaîne saisie.
	
	ex1 : int nbe;
	nbe = **Convert.ToInt32**(ch);
	
	ex2 : double nbr;
	nbr = **Convert.ToDouble**(ch);

AFFECTATION	
variable ✍ expression Le type de la variable doit être compatible avec le type de l'expression. ex : b ? a +c - (c * d) / 2	**variable = expression** Le type de la variable doit être compatible avec le type de l'expression. ex : b = a +c - (c * d) / 2 ;

Rappel :
En programmation, on appelle expression tout ce qui possède une valeur à l'exécution. Une expression est d'un certain type. Une expression peut être :

- une valeur littérale (qui apparaît telle quelle dans le code) ex : 10, "bonjour"
- une variable ou une constante ex : x, pi, tarif, …
- une opération, simple ou complexe ex : (x+y)/4, "Bonjour"+nom, a < b
- le résultat d'une méthode ex : Console.ReadLine( )

Fiche n°3: structures conditionnelles

opérateurs utilisables dans les expressions booléennes	
opérateurs de comparaison = ? < > ? ?	**==** (égalité) **!=** (différence) **> < >= <=** (supériorité infériorité)
opérateurs logiques NON ET OU (opérateurs logiques)	**!** (non) **&&**(et) **||** (ou)

Rappel :
Expression booléenne = expression qui a pour valeur VRAI ou FAUX
synonymes : condition, test, expression conditionnelle
Une expression booléenne est formée le plus souvent de comparaison(s)

✍ ***Attention au piège du signe = en C#***
en C#, le signe = ne sert qu'à effectuer des affectations et ne peut pas servir pour comparer deux expressions. Pour savoir si deux expressions sont égales, utiliser le double symbole **==**

La conditionnelle simple SI ALORS / if	

Si *expression booléenne*
  Alors *Instruction(s)*
Finsi
	if (*expression booléenne***)**
	{
	  Instruction(s) ;
	}

L'alternative SI ALORS SINON / if else	

Si *expression booléenne*
  Alors *Instruction(s)*
  Sinon *Instruction(s)*
Finsi
	if (*expression booléenne*)
	{
	  Instruction(s) ;
	}
	else
	{
	  Instruction(s) ;
	}

La structure de choix SELON / switch	

Selon expression
  cas valeur1 **:** instruction(s)
  cas valeur2 **:** instruction(s)
  …
  autres cas : instruction(s)
FinSelon
 	
- L'expression qui suit le Selon doit être de type **entier ou caractère**, jamais d'un autre type.
-Les valeurs des différents cas doivent évidemment correspondre au type de l'expression.
 	
-Après un cas, on peut éventuellement avoir une liste de plusieurs valeurs.
 	
-autres cas est facultatif
	switch(expression**)**
	{
	  case valeur1 **:** instruction(s);
	  break;
	  case valeur2 **:** instruction(s);
	  break;
	  …
	  default : instruction(s); **break;**
	}
	- L'expression qui suit le switch doit être de type entier ou string
	- Après chaque case on ne peut trouver qu'une seule valeur (correspondant au type de l'expression).
	✍ Chaque ensemble d'instruction d'un case doit se terminer par **break;** sinon le programme enchaîne sur les instructions du case suivant.
	-default est facultatif et doit aussi se terminer par break.

Fiche n°5: fonctions et procédures

Structure d'un programme	
Les fonctions et les procédures sont écrites en dehors du programme principal (appelé programme), de préférence avant	Les fonctions et procédures sont écrites en dehors de toute procédure évémentielle, mais à l'intérieur des accolades de la "section" class. En mode texte, le programme principal est la fonction créée automatiquement : **static void Main()** En mode graphique, il n'y a pas vraiment de programme principal, car c'est l'interface graphique qui permet de gérer le programme. Les sous-programmes classiques sont appelés par les procédures événementielles.
Les variables **globales** sont déclarées tout au début, en dehors des sous-programmes et du programme dans une section appelée : **var globales** Les autres variables sont **locales** au sous-programme (ou au programme principal) où elles ont été déclarées	Les variables qui sont déclarées en dehors de toute fonction sont des variables **globales** (à la classe) Les variables déclarées à l'intérieur d'une fonction sont **locales** à cette fonction.
Les Fonctions	

Fonction nomfonction**(***type_param1 nomparam1, typeparam2 param2, …*) :
type_retour
Début
…
retourne expression_retour
FinFonc
 	
Par défaut les paramètres sont de nature entrée (donnée)
	type_retour nom_fonction (*type_param1 nomparam1, typeparam2 param2, …*)
	{
	…
	return expression_retour
	}
	
	ex : double exponentielle (double C, int n)
	{
	instructions de calcul;
	return *resultat*;
	}

Les Procédures	

Procédure nomprocédure(nature type nom, nature type nom,…)
Début
…
Fin
 	
Pas de retourne
Natures des paramètres
E = Entrée (donnée) ou
S = Sortie (résultat) ou
E/S = Entrée/Sortie (donnée/résultat)
 	
Appel : l'appel d'une procédure est une instruction à part entière
 	
Quelle que soit la nature des paramètres, il ne faut rien indiquer devant, on met uniquement le nom de la variable paramètre
	On remplace le type de retour par void
	
	void nomproc(typePara nomPara,…)
	{
	
	}
	
	passage par défaut = passage par valeur (paramètres en entrée)
	
	Pour le **passage par référence** (paramètres en sortie ou en entrée/sortie):
	il faut faire précéder le paramètre par le mot clé **out** (ou ref)
	Pour les paramètres "out" il faut indiquer ce mot clé à l'appel aussi
	
	Toute action sur un paramètre "out" se répercute automatiquement sur la variable correspondante passée en paramètre à l'appel

Fiche n°6: Tableaux et structures

Les structures

Déclarées dans la section des Type

Type

Structure nom
champ1 : type
champ2 : type
Finstruct

ex :

Structure personnne
nom : chaine
age : entier
FinStruct

Le type des champs peut être une autre structure ou un tableau

Déclarées dans la classe, en dehors de Main et de toute autre sous-programme.

struct nom
{
public type champ1 ;
public type champ2 ;
…
}

ex :
struct personne
{
public string nom ;
public int age ;
}

Le type des champs peut être une autre structure ou un tableau

Les Tableaux

Déclarés dans la section des variables

nom : tableau [1..MAX] de type

ex :
mois : tableau[1..12] de chaines

le type des éléments peut être une structure
famille : tableau[1..100] de personne

Déclarés en dehors de Main et de tout sous-programme pour une portée globale ou à l'intérieur d'un sous-programme pour une portée locale

type **[]** nom = **new** type[MAX] ;

ex :
string [] mois = new string[12] ;

idem si le type des éléments est une structure
personne[] famille = new personne[100] ;

Remarque importante :
la taille du tableau peut être une variable. Mais une fois le tableau créé, sa taille ne peut plus varier, même si la valeur de la variable utilisée pour déclarer sa taille change.

ex :
int taille ;
//on fait saisir la taille à l'utilisateur
taille = Convert.ToInt32(Console.ReadLine()) ;
// on créé un tableau de cette taille
int [] montab = new int[taille] ;

Mais une fois le tableau créé, si on modifie la valeur de taille, ça n'a pas d'influence sur le tableau. Il garde la taille qu'il avait à la déclaration