Charactère en C++  
- #  
- include  
- <>  
- iostream // pour le cout  
- conio.h // pour le \_getch()  
- iomanip // pour les manipulateurs tels setw(), setprecision(), etc…  
- string // pour les type string  
- cmath  
- Windows.h // pour Sleep()  
- using  
- namespace  
- std  
- ;  
- int ou integer (ou long) en anglais voulant dire entier utilise 32 bit de ram, que des nombres entiers  
- double utilise des nombres réels  
- main  
- ()  
- {}  
- “” put text within to let the system display it  
- ‘’ pour identifier un charactère à l’aide de leur valeur dans la table ASCII  
- /\* \*/  
- //  
- \ annonce que le prochain chiffre sera en octal  
- \x annonce que le prochain chiffre sera en hexa  
- cout let’s you display text (console out)  
- <<  
- cin ou console in demande à l’utilisateur de rentrer des données ou de faire une action, pour que l’action de l’utilisateur soit enregistrer il faut que ce dernier appuie sur la touche enter. À l’avenir, toujours placé if(cin.fail()) {cout << un message d’erreur; cin.clear();}suite à un cin pour s’assurer que l’utilisateur à mis le bon type d’information dans ce dernier, clear va nettoyer les erreurs présente dans cin. Pour nettoyer le tampon du cin il faut faire cin.ignore(cin.rdbuf()->in.avail()).  
- cin.fail() s’assure que la fonction placé avant le point n’a pas raté sa fonction.  
- cin.clear() permet de nettoyer les erreurs présentent dans la fonction placé avant le point.   
- cin.ignore(cin.rdbuf()->in.avail()) permet de nettoyer le tampon de caractère du cin.  
- >>  
- \_getch() waits for an input (get character)  
- \_getche() affiche le charactère enfoncé  
- \_kbhit() Permet de vérifier s’il y a des caractères dans la mémoire tampon de l’ordinateur. Permet d’obtenir à l’avance s’il y a une file de caractères dans le tampon.

Est souvent utiliser en conjonction avec \_getch(). Permet aussi de vider la mémoire tampon avant de poser une série de questions (while(\_kbhit()) \_getch()). Permet aussi de vérifier qu’il n’y a pas de nouveau caractère dans le tampon.  
- toupper() montre la lettre majuscule, un char doit être entre les parenthèses  
- tolower() montre la lettre miniscule, un char doit être entre les parenthèses  
- endl skips a line  
- "\n" permet de passer une ligne  
- "\t" permet de faire une tabulation (voir ce type de raccourci, charactère spéciaux, à la page 34 du manuel)  
- (+-/\*=) sont des opérateurs, le reste ce trouve etre des opérants  
-< <= > >= == != sont tous des opérateurs de taille p.185 du manuel pour tous les opérateurs  
- . utiliser le point pour les decimals et nom la virgule (,)  
- E ou e signifie exposant dans une formule mathématique  
- long long ou son alias time\_t nombre entier contenu sur 64 bits ou 8 octets  
- unsigned long long ou son alias size\_t nombre entier contenu sur 64 bits ou 8 octets mais ne peut contenir que des nombres positif  
- size\_t = unsigned long long  
- char nombre entier 1 octet (historiquement utiliser pour produire le numéro d’un charactère tel que « a ») Voir table ASCII original ou étendus à la fin du manuel. Transforme les chiffres en leur charactères correspondant dans la table. On peut y utiliser du décimal, de l’octal ou même de l’hexa   
- unsigned rend notre type strictement positif  
- short nombre entier 2 octet  
- float nombre réel 4 octets  
- double nombre réel 8 octets  
- long double nombre réel 10 octets  
- bool valeur booléeenne vrai ou fausse 1 octet  
- string chaine de charactère contenu entre « » (nombre d’octets variable), s’ajuste à la taille de ta variable pour utiliser doit être inclus grâces a #include <string>. Un string est enfaite un tableau de char finissant par un caractère de buté étant 0. Voir tableau de char pour plus détails. On peut donc appelé un caractère précis dans notre chaine par exemple :   
string s = ‘’Bonjour’’; cout << s[s.size()-1]; Cela affichera r.   
- const permet de créer une constante   
- setw() permet d’enligner des chiffres ou du texte, aide à les mettre en colonne  
- setprecision() permet de déterminer le nombre de chiffres après la virgule qui seront affichés (est un manipulateur permanent)  
-left ou right détermine le côté à partir du quel l texte sera affiché (est un manipulateur permanent)  
- .size() permet de déterminer le nombre de charactère dans un contenant   
- setfill ‘’ permet de déterminer un nouveau charactère de remplissage au lieu de blanc (est un manipulateur permanent)  
- getline(cin,variable) est un remplacement à cin qui lui va prendre **tous les charactères** jusqu’au enter, pas jusqu’au première espace. Elle ne peut par contre que ramasser du texte, pas de chiffre  
- clrscr() clear screen efface ce qu’il y a dans la console et remet le curseur en haut à gauche  
- clreol() efface la ligne courante à partir de où est positionner le curseur sans pour autant le déplacer   
-clreoscr() efface ce qu’il y a dans la console à partir de où est positionner le curseur sans pour autant le déplacer

-boolalpha permet d’afficher true ou false dans un bool  
-noboolalpha permet d’afficher 1 ou 0 dans un bool

-En c++, 0 c’est faux, mais tout ce qui n’est pas 0, est vrai (1 est compris mais n’importe quelle autre valeur aussi, de toute manière elle sera remplacée par 1 à l’affichage)  
  
-if()  
-else (n’est pas obligatoire suite à un if)  
- switch() est une instruction de branchement tel que le if ou le else. Les parenthèses ne peuvent contenir que des expressions entières, pas de doubles. Un bool peut aussi être utilisé car en soit c’est des 1 et des 0. Un char peut être utilisé puisque ce sont des entiers dans la mémoire de l’ordinateur. Cherche les case ou l’expression entière est la même que dans le switch.   
- case constante\_entière : définit les instructions qui seront utilisés si l’expression entière du case est la même que celle du switch. On ne peut pas donner plus d’une valeur à un case. Il faut réécrire plusieurs case même si les instructions pour les deux cas sont les même. On peut par contre écrire de la manière suivante si les instructions sont pareilles :  
case 3 :  
case 4 :  
 cout << « choix 3 ou 4 »;  
 break;  
- default : permet d’utiliser une suite d’instruction si aucun case n’était = au switch. C’est le cas par défaut.   
- break; permet de sortir de son propre switch. Le break n’est pas utile si l’on sort de toute manière du switch.   
  
Les loops (voir manuel cvm p.118à139):  
- for ( init ; condition ; post - instruction), la première partie permet de créer et initialiser des variables utiles à la boucle FOR, la partie condition est une expression booléenne pour décider s’il faut exécuter le bloc d’instruction {}, la partie post – instruction permet de modifier (typiquement incrémenter ou décrémenter) les variables de la boucle FOR. Les variables créer dans un FOR ne sont utilisable que dans le bloc du FOR, elle est locale.   
- break; On peut briser une boucle for grâce à break; à utiliser avec parcimonie, que pour des cas spéciaux.   
- continue; Permet de passer à la répétition suivante même s’il reste des instructions à exécuter. À utiliser avec parcimonie, que pour des cas spéciaux.   
- do {} while (condition); Permet de répéter (1 à N fois) les instructions de son bloc {}. La condition est une instruction booléenne. L’instruction est avant la condition. Attention au point-virgule obligatoire après la condition.  
- while (condition) Permet de répéter (0 à N fois) les instructions de son bloc {}. La condition est une expression booléenne. La condition est avant l’instruction.  
Comment choisir entre les 3?  
1. for : Si le nombre N de répétitions est connues d’avance, c’est probablement l’instruction à utiliser.  
2. while : Si le nombre de répétition est inconnues d’avance, c’est probablement while si aucune répétition est un cas possible.  
3. do while : Si le nombre de répétition est inconnues d’avance, c’est probablement do while si au moins une répétition est nécessaire.  
Remarque : c’est aussi une question de préférence, car ces 3 instructions de répétitions sont facilement interchangeables à l’écriture.  
  
- % modulo opérateur complémentaire à la division et n’est utilisable que sur les entiers, donne le reste, ce qui n’a pas pu être diviser 5%2 =1, 10%2 =0, 3%6 =3   
  
Les transformations :  
- floor(), donne l’entier immédiatement inférieur   
- ceil(), donne l’entier immédiatement supérieur   
- round(), arrondie à l’entier le plus proche   
- abs(), retourne la valeur absolue  
  
Fonctions mathématiques, retourne tous des chiffres réels/doubles :   
- pow(), power l’exponentielle, pow(2 , 6), la base est le premier chiffre, l’exposant le second  
- log, logarithme, donne l’exposant nécessaire pour obtenir le chiffre voulu à partir de la base, donc log10(100) donnera 2. Le premier chiffre est la base. (Plusieurs variation voir notes profs)  
- sqrt(), racine carré d’un chiffre.  
  
Simulation de hasard :  
- rand() Générateur de nombres entier (Elle ne montre que des chiffres entre 0 et 32767 inclusivement) . On peut faire rand()%un nombre pour obtenir un chiffre entre 0 et le nombre choisi -1.  
- srand(i) initialise ou réinitialise l’état de départ du générateur (seed random). Il faut qu’il y ait un entier entre les parenthèses.   
- time(0) donne le nombre de secondes depuis le 1 janvier 1970  
  
  
Les opérateurs suivant sont des opérateurs logiques, affecte des opérateurs bool :  
- and && A besoin d’avoir un true de chaque côté sinon elle est fausse  
- or || A besoin d’avoir un false de chaque côté sinon elle est vrai  
- not ! opérateur unaire ne prenant qu’un seul opérant à droite, donne l’opposé de la valeur bool placé à sa droite, donc si true, alors le résultat sera false et inversement   
  
- ++ et --, opérateur unaire donc ayant une seule opérante, celle-ci peut par contre se trouver à droite ou à gauche dépendant de la situation, par contre cela ne fera pas tout le temps exactement la même chose. À pour cause d’ajouter 1 ou de retirer 1 à un réel ou un entier. (Si l’on fait cout << V++; où V=10, la valeur 10 sera affiché, on aura la valeur avant qu’elle ne soit augmentée, bien que V est tout même = à 11 alors que si l’on fait cout << ++V; , la valeur affiché sera celle après qu’elle est été augmenté, donc 11 sera affiché. La même chose est valide pour les --.

- ? (exp\_bool ? exp1 : exp2) opérateur conditionnel ternaire, qui a trois opérants. Permet de vérifier si la première variable est vraie ou fausse. Si elle est vrai, elle fera apparaitre la première valeur. Si elle est fausse, elle fera apparaitre la deuxième valeur. Le type des deux expressions doivent être le même (exp1 et exp2 doivent donc être du même type) Les parenthèses sont nécessaires dans la majorité des cas pour cause de la priorité des opérations.  
  
- Opérateurs d’affectations : =, +=, -=, \*=, /=, %= . Permet de rendre l’écriture plus concise. Donc, au lieu d’écrire c = c + (expression), on peut écrire c += (expression). Cela est valide pour chaque symbole écris précédemment.  
  
- gotoxy(x,y) permet de positionner le curseur au poin x,y souhaité. La fenêtre de la console fait 120 caractères de largeur(x) et 30 lignes de hauteur(y). Les coordonnées vont de (0,0) à (119, 29)  
- wherex() Donne la position en x actuel  
- wherey() Donne la position en y actuel  
  
- + Cet opérateur permet de faire la concaténation/juxtaposition de string. Attention : ‘’Une chaine’’ n’est pas un string. Il faut absolument qu’il y est un string à gauche de l’opérateur.  
Ex : string s;  
s = ‘’La’’;  
s = s + ‘’ ‘’ + ‘’concat\x82nation’’;  
s = s + ‘ ‘ + ‘’!!!’’;   
Cela affichera « La concaténation !!! »

- Sleep(millisecondes) Permet de mettre le programme en pause d’exécution

- type nom [N]; Array, Variable dimensionné. Il est possible de définir en une seule instruction, une série de N variables du même type. Syntaxe : type nom [ N ] ; Le nom est celui de la variable dimensionnée. N est la dimension (ou la taille) de la série et doit être un entier positif (>=1). La valeur de N doit être connu à la compilation (versus connu à l’exécution). On ne peut donc pas demander à l’usager de déterminer la valeur. Pour avoir accès à ces variables, il faut utiliser un indice. La première variable de la série est à l’indice 0. Toutes les autres variables se trouvent à l’indice N-1. N étant le nombre de variable de la série. Syntaxe : nom [ N-1 ]; On ne peut pas afficher l’entièreté du tableau sans marquer individuellement chaque variables. Le N/indice peut être placé dans une boucle. Donc si i fait partie d’un for, on peut mettre i au lieu de l’indice de nos variables. {} permet d’initialiser la valeur de tous les éléments à 0 dès la création de l’array. On peut mettre des valeurs entres ces parenthèses. En C++, il n’est pas obligatoire de mettre la taille d’un tableau si on définit toutes les variables que l’on souhaite avoir, cela est cependant peut recommander.

- Tableau/Array de char : char exemple[NB+1] = ‘’abcdefghijklmnopqrstuvwxyz’’ ou NB == 26 Un tableau de char peut être écrit ainsi, il a cependant besoin d’avoir une taille de 1 de plus que le nombre de caractère que l’on souhaite avoir dans note tableau. Car pour que le programme sache que l’on est à la fin de notre suite, il cherche un byte ayant la valeur 0, caractère de buté spécial 0. Par défaut un espace vide dans un tableau est de valeur 0, de où l’on a besoin d’avoir une taille de 1 de plus. Si on met la valeur 0, et non le caractère, dans notre chaine, le programme va s’arrêter a la partie du tableau ayant ce nouveau byte 0. C’est le seul tableau affichable directement en C++.  
  
-Tableau multidimensionnel : Voir les exemples c’est difficilement explicable. Mais finalement on utilise une formule pour se déplacer dans notre tableau. On a des variables qui définissent les différentes quantités que l’on va combiner pour déterminer la taille de notre tableau. On a d’autres variables, une pour chacune des précédentes, que l’on va utiliser pour notre équation, déterminant la variable précise que l’on souhaite faire afficher se trouvant à l’intérieur de notre tableau. Ça c’est le fonctionnement d’un tableau multidimensionnel.   
En C++, le langage peut faire exactement ça pour nous. Facilitant grandement la tâche.   
type nom [ D1 ] [ D2 ] [ D3 ]; Ceci serait un tableau à trois dimensions. On aurait N variables où N= D1 x D2 x D3. D1, D2 et D 3 sont les tailles de chacune des dimensions. Pour accéder à une variable précise, au lieu d’utiliser une formule, on écrit simplement nom [ i ] [ j ] [ k ] où la valeur de chaque dimension est représenté par une variable. La formule expliquer plutôt est faite par le langage. Pour initialiser des valeurs dans le tableau on sépare nos valeurs ainsi   
int T1 [ 2 ] [ 4 ] = { {1,2,3,4} , {1,2,3,4} }; À gauche de la virgule c’est la section 1 et à gauche la 2. Fonctionne peut importe le nombre de dimension. Exemple pour 3 dimensions :   
int T3 [ 2 ] [ 3 ] [ 2 ] = { { {1,2}, {1,2}, {1,2} } , { {1,2}, {1,2}, {1,2} } };  
  
Il est possible de forcer la conversion d’une variable en faisant ceci : type de variable (nom de la variable donc on veut changer le type). Il est important de penser à la taille des deux types de variable et si de l’information sera perdu.  
  
Il est possible d’initialiser plusieurs variables à la même valeur tel que c = b = a = 0;

Voir Manuel de cours p.7 pour avoir tous les types de contenants   
Voir Manuel de cours p.36 pour avoir tous les types de manipulateurs (ceux-ci peuvent tous se retrouver dans un cout) Permettent de changer de quelle manière le texte s’affiche   
Voir fin du Manuel de cours pour table aux ASCII

Pour former le nom d’une variable, on ne peut utiliser que des lettres majuscules, des lettres minuscules, des chiffres et les underscore (\_). On ne peut cependant pas commencer par un chiffre. Par conférence la première lettre d’un nom devrait toujours être une minuscule.