Les pointeurs servent à stocker l’adresse en mémoire d’une autre variable.

Les pointeurs peuvent être déclarés de trois façons :

Type\* var;

Type \*var;

Type \* var;

Lors d’une déclaration de plusieurs pointeurs sur une même ligne, chaque pointeur nécessite un \* le précédant : int \* ptr1, \* ptr2;

Si c’était int \* ptr1, ptr2;, ptr1 serait un pointeur et ptr2 un simple int;

Type étant le type de ce à quoi le pointeur pointe, si le pointeur doit pointer vers un String alors le type du pointeur doit être string.

L’Opérateur & permet d’accéder à l’adresse en mémoire d’une variable :

Foo = 25

Ptr = &Foo

Ptr == adresse mémoire, ex : 000000B810D7F318

L’Opérateur \* permet d’accéder à la valeur d’une valeur

Foo = 25

Ptr = \*Foo, valeur stockée à l’adresse de Foo

Ptr == 25

Si la variable à laquelle pointe un pointeur change de valeur, le pointeur change également de valeur et si le pointeur change de valeur la variable aussi.

Un pointeur peut pointer vers un tableau et devient ainsi équivalent au tableau.  
Un pointeur peut accéder aux indices du tableau de plusieurs façons :

Ptr = tableau; : Ptr retourne le premier indice du tableau [0]  
Ptr ++; point vers le prochain indice [1]  
Ptr = &tableau[2]; pointe vers l’adresse du troisième indice [2]  
Ptr = tableau + 3; pointe vers le quatrième indice [3]  
\*(ptr+1) ; pointe vers le prochain indice [4]

Le [indice] dest tableaux est un offset et fonctionne de façon similaire à \*(ptr+indice).

On peut Additionner et Soustraire des pointeurs mais la logique dépend de la grandeur du type en Byte ( char < short < int < long).  
Un ptrLong de type Long qui pointe vers à l’adresse mémoire 3000 et qui est incrémenté ++ptrLong ne vaut pas 3001, la nouvelle valeur étant 3000 + (la grandeur en Byte d’un Long dans le système, ex : 4 Bytes ) donc 3004 hypothétiquement. Même chose pour une soustraction;

L’emplacement de l’opérateur d’incrémentation (++) change la logique lors de l’opération lorsque utilisé conjointement avec l’opérateur de déréférencement :

\*p++ incrémente le pointeur et déréférence l’adresse non-incrémentée  
\*++p incrémente le pointeur et déréférence l’adresse pré-incrémentée  
++\*p déréférence le pointeur et incrémente la valeur à laquelle il pointe  
(\*p)++ déréférence le pointeur et post-incrémente la valeur

Aussi, dans une opération \*p++ = \*q++ , p est assigné q et ensuite les deux sont incrémentés, donnant essentiellement : \*p = \*q; ++p; ++q;

On peut ajouter le qualificateur const à un pointeur pour qu’il puisse accéder à une adresse/valeur sans la modifier : const int \* ptr, pointeur pointe vers un int const.

Un pointeur lui-même peut êtr const : const int\* const ptr, pointeur const pointe vers int const.

Le const peut venir avant ou après le type : const int \* ptr == int const \* ptr. Le \* doit être après par contre.

const char \* ptr = "hello"; déclare un tableau "hello" ainsi qu’un pointeur vers le premier élément du tableau donc l’adresse en mémoire de "h". On peut accéder aux autres éléments du ptr avec \*(ptr+indice) ou ptr[indice].