Pointer

&b	Rufe die Informationen auf die b zeigt ab.
*a = b	Speicher die Speichernummer für die Daten in b in der Variablen a ab.

Pointer werden benötigt, um z.B. auf Daten schneller zuzugreifen, oder um Methodenaufrufe mit Arrays zu realisieren.

Heap vs Stack

int *a = new int[10];	(Heap) Erstelle einen Pointer, der auf das erste Element eines Int-Arrays gesetzt ist und reserviere für das Array 10 Int-Speicherplätze (32Bit/4Byte) ab.
int *a = new int[10] ();	(Heap) Erstelle einen Pointer, der auf das erste Element eines Int-Arrays gesetzt ist und reserviere für das Array 10 Int-Speicherplätze (32Bit/4Byte) ab und überschreibe die bisher eingetragen Daten mit einer Null... Der Pointer wird entfernt, aber das Array bleibt erhalten (Memory Leak)
del [] a;	Lösche das Array a.
int s[10] = {1,2,3, 4,5,6,7, 8,9,0};	(Stack) Erstelle ein Array, das die ELemente ... enthält. Dieses Array wird gelöscht sobald die Funktion ausgeführt wurde.

Im Stack werden die Informationen nur im Arbeitsspeicher gespeichert. Dies führt zwar zu einer schnelleren übermittlung der Daten, hat aber den Nachteil das das das Datenvolumen stark begrenzt ist. Dem ggü. kann im Heap eine größere Datenmenge gespeichert werden.

C-String/C++-String

std::string a = "MAX"	(C++ String) Speicher den String MAX zurück
char name[4] = "MAX"	(C String) Speicher das Char-Array + die Abbruchbedingung (\0)
void doSomething (char *name){ ...}	Übergebe von dem Array(oder sonst einen char Pointer) die Adresse und führe die Fkt aus.

Strings gab es in C noch nicht. Stattdessen können Strings auch als Arrays interpretiert werden. Als Array von Buchstaben. Das Problem ist allerdings wir können das Array nicht einer Funktion übergeben. -> Pointer und Länge des Array wird benötigt.
Lösung -> Am Ende des Strings wird ein **\0** ausgegeben. Man benötigt dann nur noch den Pointer, da \0 die Abbruchbedingung darstellt.
In C++ wurde dann der "richtige" String eingeführt: -> std::string a = "ABC"

Pointer vs Referenz

Pointer: void doSomething(int *a) {...}

Referenz: void doSomething(const vector<int> &a) {...}

Pointer und Referenz können beide genutzt werden um eine Funktion zu übergeben, ohne das eine Kopie erstellt wird. Beide werden intern als Speicheradresse repräsentiert (bei Ponter vorgegeben, bei Referenzen sinnvoll).

Pointer werden für C-Array benötigt. Können mit einen +1 (a = a + 1) auf dem nächsten Eintrag zugreifen. (Großes Problem)

Referenzen stellen weniger Fkten als ein Pointer zur Verfügung. Repräsentiert aber direkt ein ganzes Objekt. (Sehr komfortabel)