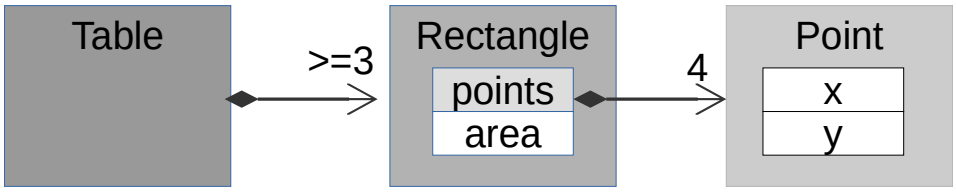
Chapitre 1 – Application pour l'utilisation de pointeurs **Exercice Algo 1.11**
Applikation zur Verwendung von Zeigern

Dans cet exercice, nous voulons réaliser une application pour utiliser des structures, des tableaux et des pointeurs dans différentes situations. De plus, l'algorithme de l'exercice 1.5 est réutilisé à la fin.

In dieser Übung wollen wir eine Applikation zur Verwendung von Strukturen, Tabellen und Zeigern in verschiedenen Situationen realisieren. Am Schluss wird zudem noch der Algorithmus bubblesort der Übung 1.5 wiederverwendet.

```
b)
Rectangle 1 point values are    : [ 0, 0][ 0,10][10,10][10, 0]
Rectangle 2 point values are    : [10,10][..,..]..
Rectangle 3 point values are    : [ 0, 5][..,..]..
c)
Rectangle 1 point addresses are : [0x61FECC,...][...,..]...
Rectangle 2 point addresses are : [........,...][...,..]...
Rectangle 3 point addresses are : [........,...][...,..]...
d)
Rectangle 1 computed area is    : 100
Rectangle 2 computed area is    : ...
Rectangle 3 computed area is    : ...
e)
Rectangle 1 point values are    : [ 0, 0][ 0,10][10,10][10, 0]
Rectangle 2 point values are    : [ 0, 5][..,..]..
Rectangle 3 point values are    : [10,10][..,..]..
```

a) Définissez une **structure** pour spécifier les coordonnées d'un point (x/y) à 16 bits. Ensuite définissez une structure pour un type de **rectangle**, qui contient 4 points de ce type pour les coordonnées du rectangle ainsi qu'un autre élément structurel pour le calcul de la surface du rectangle. Définissez et initialisez maintenant un **tableau** d'au moins 3 rectangles et initialisez ce tableau lors de définition avec des valeurs significatives.
Definieren Sie eine **Struktur** um die Koordinaten eines Punktes (x/y) mit 16bit angeben zu können. Danach definieren Sie eine Struktur für einen **Rechteck**-Typ, welche 4 solche Punkte für die Koordinaten des Rechtecks sowie noch eine weiteres Strukturelement für die Flächenberechnung des Rechtecks beinhaltet. Nun definieren und initialisieren Sie eine **Tabelle** von mindestens 3 Rechtecken und initialisieren sie diese Tabelle gleich bei der Deklaration mit sinnvollen Werten.




```c
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>

typedef struct {
  uint16_t x;
  uint16_t y;
} Point_t;

enum {
  ePointsPerRectangle = 4
};

typedef struct {
  Point_t points[ePointsPerRectangle];
  unsigned int area;
} Rect_t;


int main(void)
{
  // a) Create a variable as an array of structures of rectangles.
  //    Each rectangle has 4 points (Structure x/y as 16bit unsigned values) and
  //    an area information
  Rect_t myArrOfRectangles[] = {   {{{ 0, 0}, { 0,10}, {10,10}, {10, 0}}, 0 }
                                 , {{{10,10}, {10,30}, {30,30}, {30,10}}, 0 }
                                 , {{{ 0,5}, { 0,40}, { 5, 40}, { 5, 5}}, 0 }
                               };

  return 0;
}
```

b) Maintenant les **coordonnées des rectangles** du tableau doivent être imprimés sur la console en utilisant printf. Pour ce faire, implémentez une fonction qui peut afficher les informations pour un rectangle (transfert des paramètres par pointeur) et appelez cette fonction pour le nombre d'éléments du tableau.
Nun sollen die **Koordinaten der Rechtecke** von der Tabelle auf der Konsole mittels printf ausgegeben werden. Implementieren Sie hierzu eine Funktion, welche jeweils die Informationen für ein Rechteck ausgeben kann (Parameterübergabe per Zeiger) und rufen Sie diese Funktion für die Anzahl Elemente in der Tabelle auf.

```
/// @brief  Function to print the coordinates (x,y) of a rectangle
/// @param  pRect pointer to the rectangle to show info
/// @param  nbr   rectangle number to show in printf
/// @return -
static void showRectanglePointCoordinates(Rect_t const * const pRect, unsigned int nbr)
{
  printf("Rectangle %d point values are    : ", nbr);
  for (unsigned int i=0; i<ePointsPerRectangle; i++) {
    printf("[%2d,%2d]", pRect->points[i].x, pRect->points[i].y);
  }
  printf("\n");
}


int main(void)
{
  // a) ...

  // b) Print contents of rectangle array using function and pointers
  printf("\nb)\n");
  for (unsigned int i=0; i<cNumberOfRectangles; i++) {
      showRectanglePointCoordinates(&myArrOfRectangles[i],i+1);
  }

  return 0;
}
```

c) Outre les coordonnées, nous nous intéressons également aux **adresses mémoire des différentes coordonnées**. Implémenter une nouvelle fonction permettant d'afficher les informations d'un rectangle (transfert de paramètres par pointeur) et appeler cette fonction pour le nombre d'éléments du tableau. Que peut-on déterminer ? Comment les éléments (points → rectangles → tableau) sont-ils stockés dans la mémoire ?
Zuzüglich zu den Koordinaten interessieren uns noch die **Speicheradressen der einzelnen Koordinaten**. Implementieren Sie hierzu eine neue Funktion, welche jeweils die Informationen für ein Rechteck ausgeben kann (Parameterübergabe per Zeiger) und rufen Sie diese Funktion für die Anzahl Elemente in der Tabelle auf. Was kann festgestellt werden? Wie werden die Elemente (Punkte → Rechtecke → Tabelle) im Speicher abgelegt?

```
/// @brief  Function to print the memory locations of the coordinates (x,y)
/// @param  pRect pointer to the rectangle to show info
/// @param  nbr   rectangle number to show in printf
/// @return -
static void showRectanglePointAddresses(Rect_t const * const pRect, unsigned int nbr) {
  printf("Rectangle %d point addresses are : ", nbr);
  for (unsigned int i=0; i<ePointsPerRectangle; i++) {
    printf("[0x%p,0x%p]", &pRect->points[i].x, &pRect->points[i].y);
  }
  printf("\n");
}

int main(void)
{
  // b) ...

  // c) Print the addresses of the rectangle points (coordinates) and
  //    verify manually if they are reasonable
  printf("\nc)\n");
  for (unsigned int i=0; i<cNumberOfRectangles; i++) {
      showRectanglePointAddresses(&myArrOfRectangles[i],i+1);
  }

  return 0;
}
```

d) Dans cette partie, la variable membre de la structure rectangulaire pour la **surface** du rectangle définie en b) doit être calculée et sortie avec printf. Pour ce faire, implémentez une nouvelle fonction qui effectue le calcul de surface pour un rectangle (transfert de paramètres par pointeur) et appelez cette fonction pour le nombre d'éléments du tableau.
In diesem Teil soll nun die in b) definierte Member-Variable für die **Fläche** der Rechteck-Struktur berechnet und mittels printf ausgegeben werden. Implementieren Sie hierzu eine neue Funktion, welche jeweils die Flächen-Berechnung für ein Rechteck erledigt (Parameterübergabe per Zeiger) und rufen Sie diese Funktion für die Anzahl Elemente in der Tabelle auf.

```
/// @brief  Function to update the area information of a rectangle
/// @param  pRect pointer to the rectangle to compute the area
/// @return -
static void updateRectangleAreas(Rect_t* const pRect) {
  unsigned int const cArea =   abs(pRect->points[2].x - pRect->points[1].x)
              * abs(pRect->points[1].y - pRect->points[0].y);
  pRect->area = cArea;
}


int main(void)
{
  // c) ...

  // d) Compute the rectangle areas and update its area member variable
  printf("\nd)\n");
  for (unsigned int i=0; i<cNumberOfRectangles; i++) {
      updateRectangleAreas(&myArrOfRectangles[i]);
      printf("Rectangle %d computed area is    : %d\n",
              i+1, myArrOfRectangles[i].area);
  }

  return 0;
}
```

e) Dans cette partie, il s'agit maintenant de **trier le tableau** par surface à l'aide de l'algorithme bubblesort, puis de sortir les informations comme en b). Pour ce faire, appelez la fonction de tri à l'aide d'une variable du type **Pointer-to-Function**.
In diesem Teil geht es nun darum, die **Tabelle** mittels dem Algorithmus bubblesort nach Fläche zu **sortieren** und dann die Informationen wie in b) wieder auszugeben. Rufen Sie hierzu die Sortierfunktion mittels einer Variable vom Typ **Pointer-to-Function** auf.

```
/// @brief  Function to swap the rectangles A and B in an array
/// @param  pA pointer to the rectangle A
/// @param  pB pointer to the rectangle B
/// @return -
static void swap(Rect_t* const pA, Rect_t* const pB)
{
  Rect_t temp = *pA;
  *pA = *pB;
  *pB = temp;
}

/// @brief  Bubblesort sorting algorithm according areas of rectangles
/// @param  rects array of rectangles to sort according their areas
/// @param  nbrOfElements number of elements within the array of rectangles
/// @return -
static void bubblesort(Rect_t rects[], unsigned int nbrOfElements)
{
  unsigned int s;
  do {
    s=0;
    for (unsigned int i=1; i<nbrOfElements; i++) {
      if (rects[i].area<rects[i-1].area) {
        swap(&rects[i], &rects[i-1]);
        s++;
      }
    }

  } while (s!=0);
}

int main(void)
{
  // d) ...

  // e) Sort the rectangle array according the rectangle areas using function pointer
  //    to bubblesort algorithm (see Ex. 1.5) and print contents again b)
  printf("\ne)\n");
  typedef void (*SortFunction_t)(Rect_t rects[], unsigned int nbrOfElements);
  SortFunction_t pSortFunction = bubblesort;
  pSortFunction(myArrOfRectangles, cNumberOfRectangles);
  for (unsigned int i=0; i<cNumberOfRectangles; i++) {
      showRectanglePointCoordinates(&myArrOfRectangles[i],i+1);
  }

  return 0;
}
```