**פתרון חלק א':**

**1.** מהי העברת פרמטר כרפרנס?

העברת פרמטר כרפרנס מקבילה לשימוש במצביע, אך נוחה יותר. כלומר כל שינוי שנבצע בתוך פונקציה באמצעות פרמטרים שמוגדרים כרפרנס, יישמר גם בתוך הפונקציה וגם מחוצה לה.

ההבדל בינה לבין העברת ערך הוא שכאשר נשתמש בהעברת פרמטר כרפנרס, השינויים שנבצע יישמרו גם מחוץ לפונקציה למעשה עד סיום התוכנית, לעומת העברת ערך שיישמר עד סיום הפונקציה.

ההבדלים בינה לבין העברת מצביע הם:

א. רפרנס – חייב להיות מאותחל, מצביע – לא חייב להיות מאותחל.

ב. רפרנס – לא יכול להפנות לערך NULL, מצביע – יכול להצביע לערך NULL.

ג. רפרנס – לאחר האתחול לא נוכל לשנות את המשתנה שאיתו אנחנו מתואמים, מצביע – לאחר ההשמה הראשונית ניתן לשנות את המשתנה אליו אנחנו מצביעים.

ד. רפרנס – לעיתים, אין צורך במקום נוסף בזיכרון, מצביע – המצביע תופס מקום בזיכרון.

ה. רפרנס – סינטקס ידידותי יותר.

**2.** בשפת C++ על מנת להשיג את האפקט של realloc (שפת C), כאשר מדובר במערך דינאמי יחיד, תחילה "נעתיק" למערך זמני את הנתונים, נמחק את הזיכרון הישן שהוקצה באמצעות DELETE, נקצה זיכרון חדש בגודל הרצוי ולאחר מכן נעדכן את הערך החדש במערך, לדוגמא:

(נניח שקיים) int\* arr;

delete[] arr; (לאחר "העתקת" הנתונים למערך זמני)

arr = new int[newSize]; - הקצאת זיכרון בגודל המבוקש

arr = temp; - העברת הנתונים מהמערך הזמני למערך החדש

כאשר מדובר במערך דינאמי כפול, תחילה "נעתיק" למערך זמני את הנתונים, נמחק את הזיכרון הישן באמצעות לולאה, ולאחר מכן נקצה זיכרון בגודל הרצוי באמצעות לולאה ולאחר מכן נעדכן באמצעות לולאה את הערך החדש במערך, לדוגמא:

int\*\* arr; (נניח שקיים)

for(i=0;i<oldSize;i++){ (לאחר "העתקת" הנתונים למערך זמני)

delete[] arr[i];}

delete[] arr;

arr = new int\*[newSize]; - הקצאת זיכרון בגודל המבוקש

for(i=0;i<newSize;i++){

arr[i]=new int[size]; - הקצאת זיכרון בגודל המבוקש

arr[i] = temp[i];} – העתקת הנתונים מהמערך הזמני למערך החדש באמצעות הלולאה

**3.**  מהו ADT?

ADT (Abstract Data Type) הוא מפרט של סט מידע וסט הפעולות שניתן לבצע על אותו מידע (לדוגמא: מספרים ממשיים, מכונית, בנאדם, סטודנט וכו').

ADT מהווה ומכיל:

א. טיפוס נתונים חדש.

ב. סט של פעולות, סט הפעולות הזה מהווה ממשק (interface) לטיפוס החדש.

ג. פעולות הממשק הם המנגנון האחד והיחיד המאפשר גישה למבנה טיפוס הנתונים.

**4.** מהו כימוס?

כימוס (Encapsulation and Data hiding) הוא היכולת לאגד ביחד מידע עם פעולות – מאפשר ליצור טיפוס חדש של נתונים (להסתיר חלק מהם מהמשתמש במידה ורצוי), לדוגמא בשימוש במחלקה פעולות הממשק הם המנגנון האחד והיחיד המאפשר גישה למבנה טיפוס הנתונים (כמו שנאמר בהגדרת ADT), בנוסף לכך כאשר יש שימוש ב private אף אחד לא יכול לגשת או להפעיל את חברי המחלקה המוגדרים כך מלבד חברי המחלקה האחרים. ז"א שהם נגישים רק לחברי המחלקה עצמה. מהווה מחסום בין האובייקט והמשתמש.

**5.** מהי הכלה?

מצב שבו משתני המחלקה הם אובייקטים או מערך של אובייקטים או מצביעים לאובייקטים נקרא הכלה.

כלומר הכלה היא דרך להגדיר טיפוס חדש (מחלקה חדשה) ע"י שימוש בטיפוס קיים (מחלקה קיימת), כאשר היחס ביניהם מזוהה ע"י “has a” , לדוגמא **Circle** has a **Point**.

נשתמש בהכלה כאשר נרצה ליצור מחלקה מורכבת, ע"י שימוש בעזרת מחלקות אחרות פשוטות יותר המתאימות למחלקה, נניח בין דרכון לבין תאריך לידה, יש את הקשר “has a” – **Passport** has a **Birthdate**.

בנוסף לכך בדרכון יכולים להיות קיימים עוד אובייקטים, ולכן באמצעות הכלה ניקח את המחלקה המורכבת (Paspport) ונבנה אותה בעזרת מחלקות פשוטות יותר – וזה השימוש של הכלה.

**6.** מדוע ובאיזה מקרים אנו משתמשים בשורת האיתחול?

קריאה לפונקציה זה דבר יקר (overhead של מחסנית הקריאות), בפונקציות קצרות ה- overhead גדול יותר מעלות הפונקציה עצמה – בזבזני. באמצעות שורת האיתחול (inline) נבקש מהקומפיילר להעתיק את קוד הפונקציה במקום לקרוא לה ישירות, דבר זה חוסר הרבה בזמן ריצה. נשתמש בשורת האיתחול כאשר יהיה שימוש אך ורק בפונקציות פשוטות וקצרות, בלי תנאים, לולאות, מבני בקרה, קוד ארוך וכו'.

**7.** הבדלים בין שלושת הגדרות המשתנים שמשתמשות ב – const:

int x[] = { 5, 4, 3, 2, 1 };

const int \* ptr1 = x; 🡨 במקרה זה יווצר מערך integers **קבוע** שהתוכן שבו לא יהיה ניתן לשינוי .

int \* const ptr2 = x; 🡨 במקרה זה יווצר מערך של integers שהכתובת שלו **קבועה** ולא תיהיה ניתנת לשינוי.

const int\* const ptr3 = x; 🡨 במקרה זה יווצר מערך של integers **קבוע** וכתובתו **קבועה** ז"א התוכן שבו לא יהיה ניתן לשינוי וגם הכתובת לא תיהיה ניתנת לשינוי.

**מאור עטר - 318301231**