שיעורי בית

שבוע 2, שאלה 1

מייקל סטרגולב

שאלות

1. מהו reference וכיצד מגדירים reference למשתנה?

refrenceים בC++ הם בעצם סוגי משתנים אשר דרכם ניתן לגשת אל משתנה אחר – ולשנות אותו בדרך עקיפה.

refrence פועל ככינוי עבור המשתנה, מה שמאפשר אליו גישה באמצעות שם נוסף מבלי ליצור עותק נוסף.

refrence מוגדר באמצעות הסימן & לאחר סוג המשתנה, בעת הכרזת המשתנה.

Type& refrenceName = variableName;

- **Type –** סוג המשתנה
- **refrenceName** – שם הרפרנס
- **variableName** – שם המשתנה שאליו הרפרנס מקושר.

2. מנו שני יתרונות בהעברת משתנים לפונקציה by reference על פני שיטות אחרות.

אחד משני היתרונות בהעברת משתנים לפונקציה by refrence על פני שיטות אחרות היא **יעילות הזיכרון והביצועים.**

כאשר מעבירים משתנה על פי refrence, **לא מתבצעת העתקה של הערך** – אלא בפשוטה מועבר "כינוי" למשתנה המקורי.

דבר זה מאוד משמעותי במיוחד כאשר אנו נעבוד עם אובייקטים (מופעים של קלאסים) או מבני נתונים גדולים, מאחר והימנעות מהעתקה חוסכת זמן עיבוד וזיכרון.

יתרון נוסף הוא **היכולת לשנות את הערך המקורי** של המשתנה אליו מקושר הrefrence.

בניגוד להעברה לפי ערך (pass by value), שבה הפונקציה עובדת **עם עותק של המשתנה ולא משפיעה עליו,** בשיטת refrence הפונקציה יכולה **לשנות את הערך המקורי** של המשתנה.

זהו דבר שימושי כאשר רוצים לשמור שינויים שמתבצעים בתוך הפונקציה ושיתעדכנו מחוץ לה, מבלי להשתמש בפוינטרים מסורבלים.

3. מה הם ההבדלים בין pointer לreference ?מדוע reference נחשב ל"בטוח" יותר?

ההבדלים בין pointer לrefrence

מאפיין	Pointer	Refrence
הגדרה	משתנה המאחסן כתובת זיכרון של משתנה אחר.	כינוי למשתנה קיים.
סימון	משתמש בסימן * לגישה לערך המשתנה.	משתמש בסימן & בעת ההכרזה.
שינוי האובייקט המצביע	ניתן לשנות את הכתובת שעליה הPointer מצביע.	לא ניתן לשנות את המשתנה שהReference מצביע עליו לאחר ההכרזה.
ערך ריק	יכול להיות NULL או nullptr.	לא יכול להיות ריק – חייב להיות מקושר למשתנה בעת ההכרזה.
דרישות לאתחול	ניתן להכריז Pointer ללא אתחול.	Reference חייב להיות מאותחל בזמן ההכרזה.
פעולות מתמטיות	ניתן לבצע פעולות אריתמטיות על הPointer.	לא ניתן לבצע פעולות אריתמטיות על Refrence.
שימוש בזיכרון	דורש גישה ישירה לכתובת זיכרון.	פועל ככינוי למשתנה בזיכרון, מבלי לגשת ישירות לכתובת.

הסיבות להן refrence נחשב ל"בטוח" יותר מאשר פוינטרים הם כי refrence **דורש אתחול,** ולכן אינו יכול להיות מקושר למשתנה שאינו מוגדר או כתובת ריקה.

לעומת פוינטרים, שאינם דורשים אתחול כלשהו ועלולים להצביע לכתובת זיכרון אקראית, מה שיכול להוביל לשגיאות גישה מאוד נפוצות (כמו גישה לnullptr).

בנוסף – לאחר שrefrence נקשר למשתנה, לא ניתן לשנות אותו כך שיקושר למשתנה אחר!

ואילו פוינטרים גמישים יותר, וניתן לשנות את המשתנה אשר עליהן מצביעים בכך שמשנים את הכתובת אשר מחזיקה, אך גמישות זו יכולה לגרום לבעיות כמו שגיאות גישה לזיכרון אם פוינטר מוחלף או מתעדכן בצורה שגויה.

4. נתונה הפונקציה ונתון משתנה y מסוג int:

```
void square(int x, int& result)
{
        result = x * x;
}
```

האם הקריאות הבאות תקינות? אם לא, הסבירו מה הבעיה.

- square(3, y);

<u>קריאה זו אכן תקינה</u>, הקלטים מותאמים לפרמטרים אשר מצפה להן הפונקציה.

הפונקציה מצפה לערך מספר שלם כפרמטר ראשון – x, ואכן בקריאה מועבר המספר השלם 3.

והפונקציה מצפה לrefrence כפרמטר שני – result, ואכן בקריאה מועבר המשתנה y (משתנה מסוג int) אשר הפונקציה תוכל לקרוא כrefrence.

- square(3, &y);

<u>קריאה זו אינה תקינה</u>, הקלטים אינם מותאמים לפרמטרים אשר מצפה להן הפונקציה.

הפונקציה מצפה לrefrence כפרמטר שני – ואילו ניתן לראות כי בקריאה מגישים לפונקציה את הכתובת של משתנה y (ניתן לראות ע"פ הסימן & לפני השם של המשתנה), מה שאומר שנשלח מצביע (כתובת) ולא refrence, ולכן **הקריאה אינה תקינה.**

- square(3, 6);

<u>קריאה זו אינה תקינה</u>, הקלטים אינם מותאמים לפרמטרים אשר מצפה להן הפונקציה.

הפונקציה מצפה לrefrence כפרמטר שני – ואילן ניתן לראות כי בקריאה מגישים לפונקציה מספר שלם במקום refrence.

דבר זה גורם לקריאה שאינה תקינה מכיוון שאיננו יכולים להמיר מספר שלם קבוע אל refrence – מכיוון שאין משתנה אשר מקושר למספר קבוע! ומספר קבוע אינו יכול להציג ככינוי למשתנה אחר, מה שגורם לכך שאינו אפשר בכלל גם להמיר אותו לrefrence בדרך ישירה בפונקציה.

5. מה הבעיה בכל אחת מהפונקציות הבאות:

```
int&  getLocalVar()
{
        int x = 10;
        return x;
}
```

משתנה x הוא משתנה מקומי המוגדר בתחום של הפונקציה – מה שאומר, **שברגע שהפונקציה מסתיימת, המשתנה x יוצא המתחום שלו והזיכרון שהוקצה לו משתחרר**.

הפונקציה מחזירה refrence לכתובת הזיכרון של x, אך כתובת זו אינה חוקית לאחר סיום הפונקציה, מכיוון שהמשתנה כבר אינו קיים בזיכרון.

```
int&  getDynamicVar()
{
        int *x = new int(10);
        return *x;
}
```

הפונקציה יוצרת משתנה דינמי באמצעות new ומחזירה refrence אליו.

אומנם הrefrence המוחזר יפנה לזיכרון חוקי מכיוון שזיכרון דינמי אינו משתחרר בסוף הפונקציה – אלא רק עד שאנו משחררים אותו (או מערכת ההפעלה בסיום התוכנית כולה).

אך דבר זה עלול להביא לבעיות במקרה והאדם המשתמש בפונקציה אינו זוכר **לשחרר את הזיכרון אשר הקצנו למשתנה הדינמי בסוף שימושו.**

דבר זה עלול להוביל ל**דליפת זיכרון**, וניהול זיכרון לקוי.