סטנדרטים וקריאות

מבוא

כללי

אם אתם קוראים את המבוא הזה, סביר להניח שאתם הולכים לכתוב קוד. לאחר שתכתבו קוד, כל מיני אנשים יצטרכו לקרוא אותו (למשל, תוכניתנים שיבצעו שיפורים ושינויים בתוכנה שאתם עבדתם עליה).

החוברת הזאת באה לעזור להם, או ליתר דיוק לעזור לכם לעזור להם.

מקריאת החוברת, תלמדו **סגנון** כתיבה, תפתחו **חוש אסתטי**, ותתרגלו לחשוב על **קורא הקוד** שלכם.

בנוסף, צריך לזכור שקוד הוא התוצר העיקרי של כותב תוכנה. הקוד הוא הדבר שיישאר אחריך ויעיד על העבודה שלכם. גם כשאתם עדיין בפרוייקט, ואחרים רוצים לקבל מושג על המקצועיות שלכם, המקור הכי ברור והכי נכון של מידע בנושא, הוא הקוד שלכם. כדאי שייצג אתכם בצורה רהוטה וחיובית.

מטרות החוברת

חוברת זו מיועדת לחניכי קורס תכנות, ונועדה להעביר לקורא שיקולים, הנחיות וכללים לכתיבת תוכניות איכותיות, קריאות, אשר תואמות את הסטנדרטים הנהוגים בבסמ״ח.

החוברת כולה בנויה על הסטנדרט הנקרא ״סטנדרט הונגרי״, עם תוספות ושינויים קלים. ה״סטנדרט ההונגרי״ הינו סטנדרט מקובל בעולם כולו לכתיבת קוד קריא ואיכותי. כך שישנו סיכוי סביר שבעתיד, כאשר תיתקל בקוד כלשהו, יהיה לך קל יותר להבין אותו.

עבודה סטנדרטית

שימו לב שיכול בהחלט להיות שתתקלו בחוברת בהרבה דברים שלא מובנים כי לא למדתם אותם עדיין. בכל מקרה, בעת הקריאה התרכזו בסגנון הכתיבה, ובחלקים ובנושאים שאתם צריכים לתוכניות שלכם. אם אתם צריכים לכתוב תנאי, תחפשו דוגמה לתנאי, ואת הנקודות הכתובות על תנאים.

שגיאה נפוצה היא לכתוב לא תקנית ואז ״לדגם״ לפי הסטנדרטים. לחלקכם נראה שזה יחסוך זמן. אז זהו, שלא. כמה שיותר מהר תלמדו לכתוב מדוגם (סטנדרטי, מסודר, בהיר, קריא וכדומה), תחסכו זמן רב יותר, זמן אותו תוכלו להשקיע בדברים חשובים יותר.

תוכן עניינים

מבוא 2
תוכן עניינים 3
פרק 1 - תקנים חזותיים **4**
פרק 2 - תיעוד **9**
הערות הקדמה 9
הערות לוגיות 12
מאגר כללי תיעוד 14
פרק 3 - משתנים, קבועים וטיפוסים **15**
מאגר כללי משתנים, קבועים וטיפוסים 18
פרק 4 - מבני בקרה **19**
קטעים ביצועיים - בלוקים 19
תנאים וביטויים לוגיים 20
פקודות ברירה 20
מאגר כללי מבני בקרה 22
פרק 5 - מערכים **23**
מערכים קבועים 23
מאגר כללי מערכים 25
פרק 6 - שגרות **26**
מאגר כללי שגרות 28
נספחים **29**
נספח 1 - סדר הגדרת מודול 29
נספח 2 - קידומות סטנדרטיות 30
נספח 3 - מודול לדוגמה 31

פרק 1 - תקנים חזותיים

כללי

המראה הכללי של התוכנית הוא גורם שמשפיע על קריאותה בצורה הבסיסית ביותר. מאוד נפוץ בתחום החזותי שהחלטה על סטנדרט תהיה שרירותית וללא סיבה מיוחדת (״למה תמיד לשים רווח מצידי אופרטור? אפשר גם בלי״), כי לפעמים האחידות בין קטעי הקוד היא החשובה, ולא ההחלטה שמאחוריה.

בפרק זה נגדיר סטנדרטים חזותיים כמו עימוד וריווח. בניגוד לפרקים האחרים, לפרק זה לא יהיה מאגר כללים מסכם בסופו, מפני שבלאו הכי מדובר באוסף מסודר של כללים.

ריווח

דוגמה לשימוש בריווח (רווחים חשובים מודגשים באפור)

```
// Check if it's valid
if (nNumber >= 2107)
{
    // Switch the example
    switch (nExample)
    {
        // In case some const
        case (SOME_CONST):
        {
            // A comment with no space before it
            if (bIsValid)
            {
                ProcNoSpaceAfterMe(nOne, (nTwo + nThree));
            }

            cout << "The user name is: " << GetUsername() << endl;

            // Close a file
            fsFile.close();

            break;
        }
        // The default case
        default:
        {
            break;
        }
    }
}
```

תווי רווח

- בין פקודת ה-**return** לסוגריים המקיפים את ערך ההחזר, יבוא תו רווח.
- לפני ואחרי **אופרטורים** יהיו תווי רווח (גם "<<" ו-">>" הם אופרטורים).
  - לפני אופרטור **פסיק** לא יהיה תו רווח, אלא רק אחריו.
  - מצידי אופרטור "." ואופרטור "->" (**Member Access Operators**) לא יהיו תווי רווח.
  - בין אופרטור הוספה או הפחתה ("++" או "--") למשתנה עליו הוא פועל, לא יבוא תו רווח (כלל זה תקף עבור **כלל האופרטורים האונארים**).
- בין מילה שמורה המתחילה מבנה בקרה (if, case, while ...) לסוגריים שלאחריה, יהיה תו רווח.
- בין שם **שגרה** לסוגריים המתחילים את ההגדרה או השליחה של הפרמטרים, לא יהיו תווי רווח.
  - לדוגמה, נקרא לשגרה strcat כך: strcat("Standards", "Readability").
- אחרי פתיחת **סוגריים** ולפני סגירת סוגריים לא יבואו תווי רווח.
- לעולם **לא** יהיו שני תווי רווח (או יותר) **רצופים**.
- אין להציב רווחים במקומות **שלא מוגדרים** בפרק (כמו לפני נקודה פסיק, או לפני **נקודתיים** של case).

שורות רווח

- לפני **הערה** תבוא שורת רווח.
- לפני פקודת **עצירה** (break) ולפני פקודת **החזר** (return) תבוא שורת רווח.
- אחרי הערת שם **המודול** (שבאה בראש כל עמוד) תבוא שורת רווח.
- בין **אזורי ההגדרה** השונים (שמוגדרים בנספח 1) תהיינה שורות רווח.
  - בין פקודות בתוך **אותו אזור** לא יהיו שורות רווח (אלא במקרים שהוגדרו ספציפית).
    - לדוגמה, בין פקודות include לבין עצמן לא יהיו שורות רווח.
  - בין **ההערה הראשית** לשגרת ה-main, ובין **הערת שגרה** לשגרה שהיא מתארת, **לא נציב** שורות רווח.
    - בין **סגירת בלוק** (עם סוגריים מסולסלים) לפקודה הבאה אחריה תבוא שורת רווח.
  - בין שתי סגירות בלוק **רצופות** לא תהיה שורת רווח.
    - ישנם מקרים בהם **לעולם לא נציב** שורות רווח (אפילו אם יש צורך בהן לפי הכללים שלעיל):
  - **בשורה הראשונה והאחרונה** של בלוק.
  - לעולם לא תהיינה **שתי שורות** רווח (או יותר) **ברציפות**.
  - בין חלקים של **מבנה בקרה**, כמו בין case ל-case ובין if ל-else.
  - אין להציב שורות רווח במקומות **שלא מוגדרים** בפרק זה (למשל בסוף הקובץ, או בין סוגריים מסולסלים רצופים).
    - בין רצף פקודות cout cin לרצף פקודות cout cin תבוא שורת רווח. כמו שניתן לראות כאן:

```
cout << "Enter the number of departments" << endl;
cin >> nNumOfDept;

cout << "Enter the amount of fuel" << endl;
cin >> nTotalFuel;
```

- בין רצף פקודות cout cin לפקודה אחרת שבאה אחריהם, תוכל לבוא שורת רווח (לא חובה).

עימוד ותיחום

- **סוגריים מסולסלים** המגדירים בלוק יועמדו באותו העימוד של הקוד מסביבם (לדוגמה סוגריים מסולסלים של תנאי יועמדו באותו טור של התנאי עצמו, ראה דוגמה).
- **התוכן** בתוך כל בלוק יוזז הזחה אחת קדימה (ביחס לסוגריים המסולסלים).
- על מנת לעמד שורות או חלקים משורות, נשתמש אך ורק **בטאבים** ולא ברווחים.
- ערך ההחזר של פקודת ה-**return** וערך התנאי של **case** יתוחמו בסוגריים.
- **מספר התווים** בשורה לא יעלה על 90.
  - במידה ואורך הפקודה מעל 90 תווים, יש לפצל את הפקודה לשורות, על פי הכללים הבאים:
    - יש לעמד ביטויים חשבוניים **זה מתחת זה**, בהתאם לביטוי בתוכו הם נמצאים.
    - בפיצול פקודת לולאת for, כל אחת משלוש פקודות הלולאה תהיה בשורה נפרדת.
    - האופרטור שאחריו מגיע הפיצול, ייכתב **בסוף השורה העליונה**.
    - האופרטורים היחידים אותם כן נוריד לשורה הבאה הם "<<" ו-">>" בפקודות קלט ופלט.
    - בשורה בה יש פקודת השמה, כאשר אין אפשרות לפצל את השורה על פי כללים אלו, נרד שורה אחרי אופרטור ההשמה **ונזיח טאב אחד ימינה.**
    - אין לקלוט מספר קלטים שונים באותה פקודת cin.
- לפני קבלת קלט ממשתמש תוצג הודעה ידידותית המסבירה איזה קלט יש להכניס.

דוגמאות לפיצולי שורה אפשריים

```
arrnAverages[nNumber] =
    (nFirstNum + nSecondNum + nThirdNum) / NUM_OF_NUMBERS;
nAverage = (nFirstNum + nSecondNum + nThirdNum) /
           NUM_OF_NUMBERS;

if ((dFactor * ((nFirstNum + nSecondNum + nThirdNum +
                 nFourthNum) / NUM_OF_NUMBERS) > dLimit) ||
    (IsFunc(bIsThisReallyThisLong, bIsEnough,
            nSomeNumber)))
{
    cout << "This is a very very very very very "
         << "long message" << endl;
}
```

סוגריים

נתחום ביטויים בסוגריים כדי למנוע דו-משמעות בפענוח הקוד, הן מצד הקומפיילר והן מצד המשתמש. לקומפיילרים שונים ייתכנו סדרי קדימויות שונים, לכן לא נותיר מקום לספק ונעזר בסוגריים לקביעת קדימויות.

דוגמה לשימוש נכון בסוגריים

```
1    for (nIndex = AVG_BEGIN - 1; (nIndex < (VEC_SIZE - 1)) || (bIsFinish); nIndex++)
2    {
3         nAverage = ((nIndex * nAverage) + arrnNumbers[nIndex]) / (nIndex + 1);
4    }
5
6    cout << "Total: "
7         << nFactor * AddToTotal(nAverage * (VEC_SIZE - AVG_BEGIN + 1), nInterest) << endl;
```

- אין לתחום ביטויים בסוגריים **ללא סיבה**.
- אין לתחום ערכים מיידים **בודדים** בסוגריים נפרדים.
- כאשר ישנם מספר אופרטורים (השוואתיים ו/או חשבוניים) ברצף, נתחום בסוגריים לפי **הקדימות** הרצויה.
  - יש לבצע תיחום כאשר יש מספר אופרטורים ברצף, גם אם לפי סדר פעולות חשבון אין צורך בכך.
    - **לדוגמה**: (nPrice * nTax) / nDiners) + nSecurityCost)
    - כאשר יש מספר אופרטורים רצופים שחל עליהם חוק החילוף (חיבור ו/או חיסור ביחד, כפל ו/או חילוק ביחד), ניתן לתחום אותם בסוגריים משותפים ולא בנפרד.
      - **לדוגמה**: (nHa + nBu + nBit) / 3
      - **לדוגמה**: pow(nB, 2) - (4 * nA * nC)
      - **לדוגמה**: nSuccesses * 100 / nAttempts
- נתייחס לאופרטור ההשמה (״=״) ולאופרטור פסיק (״,״) כבעלי קדימות ראשונה תמיד, ולכן לצידם אין צורך לתחום בסוגריים ביטויים חשבוניים.

דוגמה לסוגריים מיותרים (מסומנים באפור)

```
1    for (nIndex = (AVG_BEGIN - 1); ((nIndex <= (VEC_SIZE - 1)) || (bIsFinish)); nIndex++)
2    {
3         nAverage = (((nIndex * nAverage) + arrnNumbers[nIndex]) / (nIndex + 1));
4    }
5
6    cout << "Total: "
7         << (nFactor * AddToTotal((nAverage * ((VEC_SIZE - AVG_BEGIN) + 1)), nInterest))
8         << endl;
```

אזורי הגדרה

- הקוד יחולק לאזורים שונים כדי להקל על קריאתו.
  - רשימת האזורים המלאה וסדרם נמצא ב**נספח 1**.
- אין לרדת שורה אחר **הגדרת** אזור (מלבד אחרי Code section, שבמקרה זה יש שורה רווח לפניו ואחריו).
- כאשר אין צורך באחד האזורים, נשמיט את **ההערה** שלו.
- בכל אזור הגדרה יתבצע **עימוד** של כל **שמות הפרמטרים** אחד מתחת לשני, וגם של כל **אופרטורי ההשמה** – לפני אופרטור ההשמה הרחוק ביותר.
  - באזורים **שבתוך השגרות** יתבצע עימוד באופן זהה.

עימוד שמות ואופרטורים לדוגמה (קטעי עימוד מסומנים באפור)

```
1    // Typedef definition
2    typedef int NewKindOfInt;
3    typedef char NewChar;
4
5    // Global variable definition
6    const NewKindOfInt g_nkonOne = 1;
7    const int          g_nTwo    = 2;
8    const NewKindOfInt g_nkonThirdAndLast;
9
10   // Function prototypes
11   void FirstProc();
12   int SecondFunc(int inFirstNum,
13                  int* iopnSecondNum);
```

פרק 2 - תיעוד

כללי

תיעוד הוא אחד החלקים החשובים ביותר שתוכניתן צריך לדעת לבצע כאשר הוא כותב תוכנית בשפה כלשהי.
תיעוד בא בצורות שונות, אבל מטרתו אחת - להסביר לתוכניתן שקורא את הקוד מה מבצעת התוכנית.
אם מבנה התיעוד קבוע, התוכניתן ידע לאיזה מידע לצפות ואיפה למצוא אותו, ולכן נגדיר סטנדרט בנושא.
חשוב מאוד לשים דגש על **איכות** התיעוד, ולא על הכמות.

הערות הקדמה

הערה ראשית

כשתוכניתן שאינו בקיא בתוכנית ינסה להבין כיצד היא פועלת, נרצה להסביר לו בכלליות את מטרתה, ללא פירוט מיותר. את ההסברים נרשום בהערה, שתיקרא הערה ראשית, שתבוא בראש **כל תוכנית** שנכתוב.
להערה הראשית מבנה קבוע שמקל על הבנת התהליכים המרכזיים בתוכנית. מבנה אחיד וקשיח מאפשר להגדיר בצורה אחידה את משמעות כל סעיף, ללא כפלי משמעות ואי-דיוקים (**לדוגמה** אם אין קלט, לא נשמיט את סעיף הקלט, כי כך הקורא עלול לתהות האם סתם שכחנו לכתוב את הסעיף, אלא נכתוב בסעיף הקלט None).

הערה ראשית לדוגמה

```
//---------------------------------------------------------------------------
//                                Pazam Meter
//                                -----------
//
// General : The program calculates remaining time in army.
//
// Input   : Recruit date and service end date.
//
// Process : Calculate the difference between both dates.
//
// Output  : Prints remaining time in the army.
//
//---------------------------------------------------------------------------
// Programmer : Fasa Noser
// Student No : 1222
// Date       : 20.05.1979
//---------------------------------------------------------------------------
void main()
...
```

מבנה

- בהערה הראשית יופיעו ארבעה סעיפים:
  - **סעיף כללי (General)** המכיל תיאור כללי של התוכנית והפעולה שהיא מבצעת. הקפידו לא לציין בסעיף זה מידע ספציפי לגבי פעולות קלט/פלט או עיבוד בתוכנית.
  - **סעיף קלט (Input)** המתאר את המידע שהתוכנית מקבלת, לרוב מהמשתמש או מקובץ.
  - **סעיף עיבוד (Process)** המתאר באופן תמציתי, אך ברור, את העיבוד שמבוצע בתוכנית. הקפידו לא להיסחף לפירוט מיותר אך גם לא לשכוח את הדברים החשובים שמתבצעים.
  - **סעיף פלט (Output)** המתאר את המידע שהתוכנית מוציאה/מייצרת. גם פה, פלט יכול לקבל צורות שונות (קובץ, הדפסות למשתמש).
- ארבעת הסעיפים יופיעו תמיד, ובמקרה שאין מה לכתוב באחד מהם יש לציין None.
- בהמשך ההערה נציין פרטי התוכניתן (שם, מספר חניך ותאריך כתיבת הקוד).
- הכותרת של ההערה הראשית תהיה כשם התוכנית/המערכת. אם שם המערכת מורכב ממספר מילים, יש להפרידן על ידי רווח. למשל עבור התוכנית MyProgram - הכותרת בהערה תהיה My Program.
- בין ההערה הראשית להגדרת התוכנית (void main), לא תבוא שורת רווח.
- שורות המקפים יהיו שוות זו לזו באורכן.
  - אורכן של שורות המקפים יהיה לפחות כאורך תוכן ההערה אך לא יותר מ90 תווים.
  - תוכן ההערה לא יחרוג (אופקית) מהגבולות שמגדירות שורות המקפים בתחילתה ובסופה.
- ההערה הראשית צריכה להיראות בדיוק כמו ההערה לדוגמה (מבחינת מיקום הכותרת, שורות רווח, רמת פירוט וכדומה).

הערה לשגרה

הערה לשגרה דומה בתפקידה להערה ראשית לתוכנית. גם היא נועדה לתת רקע כללי על קטע הקוד שכתבנו מבלי לדרוש מהקורא לרדת לפרטי פרטים.

דוגמה להערה לשגרה

```
//---------------------------------------------------------------------------
//                                Calculate
//                                ---------
//
// General      : The function calculates the math operation of
//                two numbers and an operator.
//
// Parameters   :
//       inFirstOperand  - First number (In)
//       inSecondOperand - Second number (In)
//       inOperator      - The operation to perform on the numbers (In)
//
// Return Value : The result of the operator and the two operands.
//
//---------------------------------------------------------------------------
int Calculate(int inFirstOperand,
              int inSecondOperand,
              int inOperator)
...
```

מבנה

- ההערה תופיע לפני כל שגרה במערכת ותורכב מהסעיפים הבאים:
  - **סעיף כללי (General)** שיסביר מה השגרה עושה ובמידה והיא מאוד מסובכת אז גם אופן פעולתה הכללי.
  - **סעיף פרמטרים (Parameters)** - הפרמטרים שהשגרה מקבלת, והסבר קצר על תפקיד כל אחד מהם. לכל פרמטר כתוב את סוגו (In, Out, I/O).
  - **סעיף החזר (Return Value)** שיסביר את משמעות ערך ההחזר בתוכנית.
    - אם ישנם מקרי קצה מסוימים בהם השגרה מטפלת בצורה שונה יש לציין זאת כאן.
    - כל סעיפי ההערה יופיעו תמיד, ובמידה ואין צורך באחד מהם, ייכתב בו ״None״.
  - **לדוגמה**, בסעיף ההחזר של שגרה שטיפוס החזרה הוא **void** (פרוצדורה), נכתוב ״None״.
- הכותרת של ההערה לשגרה תהיה כשם השגרה. אם שם השגרה מורכב ממספר מילים, יש להפרידן על ידי רווח. למשל עבור השגרה AddNumbers - הכותרת בהערה תהיה Add Numbers.
- בין ההערה להגדרת השגרה לא תבוא שורת רווח.
- **כל פרמטר** ייכתב בשורה נפרדת.
- לפני ההערה תבוא שורת רווח במידת הצורך (על פי כללי התיעוד של הערות לוגיות המופיעים בהמשך הפרק).

- שורות המקפים יהיו שוות זו לזו באורכן.
  - אורכן של שורות המקפים יהיה לפחות כאורך תוכן ההערה אך לא יותר מ90 תווים.

  - תוכן ההערה לא יחרוג (אופקית) מהגבולות שמגדירות שורות המקפים בתחילתה ובסופה.
- ההערה לשגרה צריכה להיראות בדיוק כמו ההערה לדוגמה (מבחינת מיקום הכותרת, שורות רווח, רמת פירוט וכדומה).

הערות לוגיות

כללי

באמצעות הערות בין שורות הקוד נוכל להסביר את הקוד, להדגיש נקודות חשובות, להסביר שורה מסובכת במיוחד ועוד. באמצעות הערות אלו תוכניתנים אחרים יוכלו לקרוא ולהבין את הקוד שלנו בקלות, וגם אנחנו נוכל לחזור בעתיד לקוד שכתבנו ולהבין אותו במהירות.

תוכניתן ממוצע מבין פקודות פשוטות והקשרן בתוכנית. הצורך בתיעוד נוסף נוצר בקטעים לוגיים מסובכים.

כשהערות הן סתמיות (חסרות תוכן, או שוות ערך בדיוק לקוד), הן גוזלות זמן ועלולות להיות מיושנות או שגויות.

קווים מנחים

חשוב לכתוב את ההערות בזמן כתיבת הקוד עצמו, כדי שהן תהיינה כמה שיותר נאמנות לקוד ומדויקות. הזמן המתאים ליצירת תיעוד הוא מיד לאחר כתיבת קטע מסוים. כך לא יהיו סתירות בין הקוד להערות ולא יגרם בלבול לתוכניתן הקורא את התוכנית. נשאלת השאלה - **כמה לתעד**? אין תשובה אחת. חייבים להפעיל פה שיקול דעת. ככל שתכתבו תוכניות רבות יותר, כך תפתחו חוש לכמות התיעוד הנדרשת לכל קטע קוד.

- בראש כל קובץ נכתוב את שם הקובץ בהערה ולאחריה שורת רווח.
- יש להסביר חלקים לוגיים בתוכנית, או פקודות מסובכות ולא שגרתיות. ניתן כמובן לכתוב הערה אחת למספר שורות המבצעות מטלה אחת משותפת.
  - **לכל מבנה בקרה** (תנאי, Else if, לולאה או פקודת ברירה) יש לכתוב הערה.
- ניתן להשמיט הערות לפקודות **פשוטות** במיוחד שניתן להבין בקלות גם ללא הערה.
- מומלץ לכתוב הערות הסבר בין חלקים של מבני בקרה (לפני כל case ולפני כל else).
  - במידה ונכתוב הערה עבור case מסויים בפקודת ברירה, נכתוב הערה גם עבור שאר הcaseים **כולל הdefault.** שימו לב שניתן להשאיר את הcaseים ואת הdefault ללא הערה לוגית, אבל **תמיד** מעל הswitch תגיע הערה.
- הערות תיכתבנה באנגלית.
- כל ההערות יתחילו בתו רווח ולאחריו אות גדולה (למעט שורות מקפים בהערות ראשיות והערות לשגרה).

דוגמאות להערות לוגיות

```
1    // Check if the array isn't sortable.
2    if (!bIsSortable)
3    {
       ניתן להשמיט הערות לפקודות פשוטות במיוחד שניתן להבין בקלות גם ללא הערה
4        cout << "Error" << endl;
5    }
6    // The array is sortable
7    else
8    {
9        // Sort the array
10       for (nCurrIndex = 0; nCurrIndex < NO_OF_CELLS; ++nCurrIndex)
11       {
         כאן יבוא הקוד למיון מלווה בהערות לוגיות המסבירות את תהליך המיון
12       }
13   }
14
15   // Read the first firstname
16   cin >> szFirstName;
17
                                        שימו לב כיצד ניתן לחלק הערה לוגית לשתי שורות
18   // Read the names and print them while the firstname is not the
19   // ending string
20   while(strcmp(szFirstName, NAME_LIST_END) != 0)
21   {
22       // Read the last name
23       cin >> szLastName;
24
25       // Print the full name
26       cout << szFirstName << " " << szLastName << endl;
27
28       // Read the next first name
29       cin >> szFirstName;
30   }
31
                                        הערה אחת לקטע לוגי המורכב מכמה פקודות
32   // Switch two numbers
33   nTempNum = nFirstNum;
34   nFirstNum = nSecondNum
35   nSecondNum = nTempNum;
```

הערות למשתנים

- שמות המשתנים צריכים להיות מספיק משמעותיים כדי שיהיה ניתן להבין מה תפקידם גם מבלי לקרוא הערה נוספת, ולכן לעולם לא נרשום הערה לוגית למשתנה.

מאגר כללי תיעוד

בחלק זה נאגד לנוחיותכם את הכללים שהופיעו בפרק זה. דוגמאות והסברים מפורטים תמצאו בתוך הפרק.

- לכל תוכנית נכתוב **הערה ראשית**.
  - ההערה תכיל:
    - סעיף כללי
    - סעיף קלט
    - סעיף עיבוד
    - סעיף פלט
    - פרטי התוכניתן
  - כל סעיפי ההערה יופיעו תמיד, ובמידה ואין צורך באחד מהם, ייכתב בו ״**None**״.
  - בין ההערה לשגרת ה-main לא תהיה **שורת רווח**.
  - לפני ואחרי כל אחד מארבעת הסעיפים בהערה הראשית תבוא שורת רווח.
  - שורות המקפים יהיו **שוות** זו לזו **באורכן**.
    - תוכן ההערה לא **יחרוג** (אופקית) מהגבולות שמגדירות שורות המקפים בתחילתה ובסופה.
- לכל שגרה נכתוב **הערה לשגרה**.
  - ההערה תכיל:
    - סעיף כללי
    - סעיף פרמטרים
    - ערך החזר
  - כל סעיפי ההערה יופיעו תמיד, ובמידה ואין צורך באחד מהם, ייכתב בו ״**None**״.
  - **הכותרת** של ההערה לשגרה תהיה כשם השגרה, כאשר המילים תופרדנה על ידי רווחים.
  - **כל פרמטר** ייכתב בשורה נפרדת.
  - בין הערת השגרה להגדרתה לא תהיה **שורת רווח**.
  - שורות המקפים יהיו **שוות** זו לזו **באורכן**.
    - תוכן ההערה לא **יחרוג** (אופקית) מהגבולות שמגדירות שורות המקפים בתחילתה ובסופה.
- בראש כל קובץ נכתוב את שם הקובץ בהערה ולאחריה שורת רווח.
- אין לכתוב הערות לוגיות כדי **להסביר משתנים**.
- יש לכתוב **הערה לוגית** לקטעים לוגים מסובכים או מורכבים. **לכל מבנה בקרה יש לרשום הערה לוגית שתסביר אותו.**
- ניתן **להשמיט** הערות לפקודות **פשוטות** במיוחד שניתן להבין בקלות גם ללא הערה.
- כל ההערות הלוגיות יתחילו בתו רווח ולאחריו אות גדולה.

פרק 3 - משתנים, קבועים וטיפוסים

כללי

שמות משתנים טובים הם סוג של תיעוד, הם הופכים את הקוד לקריא יותר ומקלים את קריאתו.

קווים מנחים

- חשוב שהקורא יבין בקלות 3 דברים לגבי כל משתנה:
  - **תפקידו** (למשל, זה מונה).
  - **הקשרו** (למשל, מונה טנקים).
  - **טיפוסו** (למשל, int).
- שימוש במשתנה אחד לכמה מטרות יוצר בלבול.

הסבר

- כדי ש**תפקיד** המשתנה יהיה ברור בקריאה ראשונה, יש לציין את התפקיד בשם המשתנה. בדוגמה שלנו, שמו של מונה האורחים נגמר במילה Counter.
- לא נאגלז שמות משתנים. כלומר שם מונה האורחים יהיה Counter ולא Mone.
- לא נשתמש בספרות בשמות משתנים. לדוגמא נקרא למשתנה nFirstInput ולא nInput1,
- לא נשתמש בתווים בשמות משתנים. לדוגמא נקרא למשתנה nFirstInput ולא n_First_Input, (שימו לב שבקו תחתון ניתן להשתמש בשמות קבועים, אך לא בשמות משתנים). באופן כללי שמות משתנים יכילו רק אותיות באנגלית.
- **הקשרו** של המשתנה לעולם התוכן ולתוכנית גם הוא יצוין בשמו. בדוגמה שלנו, האורחים מיוצגים על ידי המילה Residents וכך ניתן להבין מה מטרת המונה nResidentsCounter.
- נשתמש במשתנה רק בהקשר אחד. כיום, כאשר הזיכרון הפנימי מגיע לגדלים עצומים, אנו יכולים להרשות לעצמנו להגדיר משתנה נוסף במקום להשתמש במשתנה קיים, גם אם משמעותם דומה.
  - **לדוגמה**, בתוכנית שמטפלת בביקור אצל רופא שיניים, המשתנה nPaymentSum יכול להכיל סכום של סוגי ערכים שונים (למשל התשלום על ביקור אחד, או התשלום לאורך כל הביקורים בשנה לשלמה), אבל נגדיר שני משתנים: nAppointmentPaymentSum ו-nYearlyPaymentSum, שהם חד-משמעיים.
- על מנת ש**טיפוס** המשתנה יהיה ברור, האות הראשונה בשם המשתנה תציין את טיפוסו. בדוגמה שלנו nNewResidentsNum הוא מספרי (Number) ובמימוש ב-C הוא מסוג Integer. בדומה, bIsHotelClosed הוא בוליאני.

- טבלת התחיליות המלאה נמצאת בנספח 2.

- שמו של משתנה בוליאני ינוסח בצורת שאלה כאשר התשובה לשאלה תהיה true או false. בדוגמה שלנו, הבוליאני שקובע האם המלון סגור, נקרא bIsHotelClosed.
- **דוגמה נוספת**: בוליאני שקובע האם המשתמש יצא מהתוכנית, יקרא bDidUserExit.
- אין להגדיר **שני משתנים** בשורה **אחת** (על ידי הפרדה עם פסיק).

שימו לב כמה קל להבין את התוכנית שלהלן כאשר שמות המשתנים ניתנים לפי הסטנדרטים. גם אתם צריכים להגיע למצב שבו הקוד שלכם מסביר את עצמו.

תוכנית לדוגמה עם שמות משתנים תקניים

```
1    bool bIsHotelClosed;
2    int  nNewResidentsNum;
3    int  nResidentsCounter;
...
4   if (!bIsHotelClosed)
5   {
6    cin >> nNewResidentsNum;
7    nResidentsCounter += nNewResidentsNum;
8   }
```

סוגי משתנים

על מנת להבדיל בין סוגי המשתנים השונים (משתנה רגיל, קבוע, גלובאלי וכדומה) נגדיר כמה סטנדרטים נוספים.

סוג	הסבר	דוגמה
משתנה מקומי	האות הראשונה הינה תחילית המסמלת את סוג המשתנה. כל מילה בשם המשתנה מתחילה באות גדולה.	int nNoOfBalloons = 0;
משתנה גלובאלי	למשתנים גלובאליים נוסיף תחילית (g_).	extern g_nHorseNo;
קבוע	קבועים יירשמו תמיד באותיות גדולות עם '_' בין מילה ולמילה	#define VEC_SIZE 100 const int MIN_AGE = 18;
קבוע גלובאלי	יירשמו כמו קבועים רגילים, וללא תחילית g.	const int MAT_SIZE = 1222;

Enum	שם ה-enum יירשם עם אות גדולה בתחילת כל מילה. שמות הערכים הם כמו קבועים ולכן יכתבו באותיות גדולות.	enum DaysOfWeek
		{
		SUNDAY = 1,
		MONDAY,
		..
		};

Struct	כאן יחולו אותם כללים כמו ב-Enum, כאשר על השדות הפנימיים חלים אותם התקנים כמו על משתנים רגילים.	struct PersonDetails
		{
		int nAge;
		char* pszName;
		};

- כאשר נגדיר מופע של מבנה שנוצר על ידינו, התחילית שתייצג אותו תורכב מקיצור של שמו. בדרך כלל נשתמש במספיק אותיות על מנת להבדיל בין המבנים השונים. ככלל, ניתן להשמיט אותיות A, E, I, O,U ועיצורים פחות חשובים, ולהשתמש בעיצורי המבנה העיקריים בלבד.
  - לדוגמה תחילית של מבנה בשם StudentDetails תקרא sdtl.
  - רצוי שהתחילית לא תעלה על 4 תווים.

מאגר כללי משתנים, קבועים וטיפוסים

בחלק זה נאגד לנוחיותכם את הכללים שהופיעו בפרק זה. דוגמאות והסברים מפורטים תמצאו בתוך הפרק.

- **תפקיד** המשתנה, **הקשרו וטיפוסו** צריכים להיות ברורים **משמו**.
  - בתחילת שם המשתנה תבוא **תחילית** בהתאם לטיפוסו.
    - התחילית שתייצג **מבנה** שנוצר על ידינו תורכב מקיצור של שמו.
    - טבלת התחיליות המלאה נמצאת בנספח 2.
  - לא נשתמש בספרות בשמות משתנים.
  - כל שמו של משתנה בוליאני ינוסח **בצורת שאלה**.
  - כל **אות ראשונה** בכל מילה בשמו של משתנה מקומי תהיה אות גדולה.
  - למשתנים **גלובאליים** נוסיף את התחילית ״g_״.
  - שמם של **קבועים** ייכתב באותיות גדולות.
    - אין צורך להשתמש בתחילית ״g_״ עבור **קבועים גלובאליים**.
  - שמם של **Enums** יירשם עם אות גדולה בתחילת כל מילה.
    - שמות ה**ערכים** יירשמו עם אותיות גדולות בלבד.
  - שמם של **מבנים** (Struct) יירשם עם אות גדולה בתחילת כל מילה.
    - אין להשתמש באותו המשתנה ביותר **מהקשר** אחד.

פרק 4 - מבני בקרה

כללי

כפי שמבני בקרה שונים נותנים לשפה כוח וגמישות, כך הם גם מהווים פוטנציאל להסתבכות מיותרת, ולכן חשוב להגדיר כללים נוקשים בכל הנוגע למבני הבקרה.

קטעים ביצועיים - בלוקים

קיימת שגיאה נפוצה שיכולה להיות די קשה לגילוי.

ננ יח שאנו רוצים להדפיס מספרים

```
for (nCurrNumber = 0;
     nCurrNumber < MAX_NUMBER;
     ++nCurrNumber)
    cout << nCurrNumber << endl;
```

עכשיו ננ יח שאנו רוצים גם לסכום אותם

```
for (nCurrNumber = 0;
     nCurrNumber < MAX_NUMBER;
     ++nCurrNumber)
         nSum += nCurrNumber;
         cout << nCurrNumber << endl;
```

מה הבעיה? איך הקומפיילר ידע האם ה-cout הוא בתוך הלולאה או לא? מבחינת הקומפיילר, רק הפקודה הראשונה שייכת ללולאה. מבחינת קורא אנושי, העימוד קובע.

- תמיד נגדיר קטע ביצועי במבנה בקרה, על ידי הגדרת בלוק בעזרת שימוש בסוגריים מסולסלים.
  - אם אין שום פקודות בקטע הביצוע, עדיין נגדיר בלוק, והוא פשוט יהיה ריק.

```
for (nCurrNumber = 0;
     nCurrNumber < MAX_NUMBER;
     ++nCurNumber)
{
    cout << nCurrNumber << endl;
}
```

- **תזכורת משיעור לולאות:**
  - לעולם לא נשנה את המונה של לולאת מונה בקרה בתוך הקוד שבתוכה (אלא רק בחלק השלישי של הגדרתה).
  - לא נשתמש במונה הלולאה מחוצה לה.

תנאים וביטויים לוגיים

- במידה ונרצה להשתמש בתנאים מורכבים נפריד כל ביטוי לוגי בסוגריים משלו.
  - גם משתנה בוליאני נחשב לביטוי חשבוני בפני עצמו ולכן יתוחם בסוגריים.
- מצידם של אופרטורים (השמה, השוואה, שרשור וכדומה) יהיו תווי רווח.
  - קטע הביצוע הקצר תמיד יופיע ב - if כאשר הארוך יותר יופיע ב – else **מבחינה ויזואלית.** מה הכוונה? אם יש לי פקודה אחת בכל בלוק, אבל אחת מהפקודות מועמדת בצורה כזו שהיא מתפרשת על פני 2 שורות בעוד שהשנייה מתפרשת על שורה אחת בלבד, הפקודה הראשונה היא זו שתופיע בחלק ה-else.
  - יוצא דופן מכלל זה הוא האופרטור פסיק, שלפניו אין תו רווח ולאחריו יש.
- אין להשוות משתנה בוליאני ל-true או ל-false בביטויים לוגיים. יש להשתמש במשתנה עצמו כביטוי לוגי בפני עצמו (עם תוספת של אופרטור "!" במקרה של שלילה).
- השימוש באופרטור הטרינארי ? : אינו סטנדרטי.

דוגמה לשימוש נכון בסוגריים וברווחים

```
if ((nCurrAmount < 10) && (vpow(nNumber, 2) > 5) && (!bIsValid))
```

פקודות בריה

- לכל פקודת ברירה יהיה מקרה **default**, גם אם הוא יהיה ריק.
- חלק ה-default ישמש כחלק ברירת מחדל **בלבד**.
- בסוף כל מקרה תהיה שורת break.
- לפני פקודת break תבוא שורת רווח.
- ניתן (ואף מומלץ) לרשום הערה לפני מקרה כלשהו.
- חובה להקיף את בלוק הקוד השייך לcase בסוגריים מסולסלים

דוגמה סטנדרטית:

```
9          switch (arrnGrades[0])
10         {
11              case (10):
12              {
13                   nGrade += 20;
14                   cLetter = 'A';
15
16                   break;
17              }
18              case (8):
19              {
20                   nGrade += 15;
21                   cLetter = 'B';
22
23                   break;
24              }
25              default (6):
26              {
27                   nGrade += 0;
28                   cLetter = 'F';
29
30                   break;
31              }
32     }
```

מאגר כללי מבני בקרה

בחלק זה נאגד לנוחיותכם את הכללים שהופיעו בפרק זה. דוגמאות והסברים מפורטים תמצאו בתוך הפרק.

- נגדיר קטע ביצועי במבנה בקרה, על ידי הגדרת **בלוק** בעזרת שימוש בסוגריים מסולסלים.
  - אם אין שום פקודות בקטע הביצוע, עדיין נגדיר בלוק, והוא פשוט יהיה **ריק**.
    - כל ביטוי לוגי יתוחם **בסוגריים** משלו.
  - גם משתנה **בוליאני** יוקף בסוגריים.
    - מצידם של **אופרטורים** יהיו תווי רווח.
  - יוצא דופן מכלל זה הוא האופרטור **פסיק**, שלפניו אין תו רווח ולאחריו יש.
- אין להשוות משתנה בוליאני ל-true או ל-false בביטויים לוגיים. יש להשתמש במשתנה עצמו כביטוי לוגי בפני עצמו.
- השימוש באופרטור הטרינארי ? : **אינו סטנדרטי**.

פרק 5 - מערכים

כללי

- בכל מקום בו אנו מתייחסים בפרק זה למערך, הכוונה גם למערך רב מימדי, כמו מטריצה.
- שם של מערך יהיה כמו שם של משתנה רגיל, מלבד תוספת של תחילית ״arr״ לפני תחילית טיפוס המערך.
- מערכים יופיעו תחת איזור ה-Array definition (מלבד מחרוזות שעדיין יופיעו תחת איזור ה-Variable definition).

אתחול בהגדרה

- באתחול מערכים בהגדרה **תמיד יש לציין את גודל המערך**, וכן יש לציין ערכים לאתחול עבור כל תא.

דוגמה סטנדרטית לאתחול מערך

```
int arrnNums[VEC_SIZE] = {1, 2, 3, 4, 5};
```

דוגמה לא סטנדרטית לאתחול מערך

```
int arrnNums[VEC_SIZE] = {0};
```

אפילו שהדרך השנייה חוסכת לנו לולאת אתחול בהמשך, אנו לומדים לתכנת בכלליות ולא ספציפית לקומפיילר מסוים, ולכן לא נסתמך על הקומפיילר.

מערכים קבועים

כללי

- מערכים קבועים יוגדרו תחת ה-**Const definition** (ולא תחת Array definition).

שמות

- שמות של מערכים קבועים יכתבו ללא תחיליות כלל (לא של סוג הטיפוס, ולא תחילית ״arr״).
- שמות של מערכים קבועים יכתבו באותיות גדולות, כשמילים מופרדות באמצעות קו תחתון.

אתחול בהגדרה

- **חובה** לאתחל את כל התאים במערך קבוע בעת הגדרתו (אחרת לא נעבור קומפילציה), זאת בניגוד למערך רגיל אותו כמובן ניתן שלא לאתחל בעת ההגדרה.

הגדרת מערך קבוע

```
const int EXAMPLE_NUMBERS[VEC_SIZE] = {1, 2, 3, 4, 5};
```

מאגר כללי מערכים

בחלק זה נאגד לנוחיותכם את הכללים שהופיעו בפרק זה. דוגמאות לדברים מורכבים תמצאו בתוך הפרק.

- מערכים (גם רב מימדיים) יוגדרו תחת ה-**Array definition** (מלבד מחרוזות שעדיין יוגדרו תחת איזור ה-Variable definition).
- באתחול מערכים בהגדרה יש לציין את **גודל המערך**.
  - גודל המערך יהיה **קבוע**.
- כאשר מאתחלים מערך בהגדרה יש **לציין ערכים** לאתחול עבור כל תאי המערך (בנפרד).
- שם של מערך יהיה כמו שם של משתנה **רגיל**, מלבד תוספת של תחילית ״**arr**״ לפני תחילית טיפוס המערך.
  - **מערכים קבועים** יוגדרו תחת ה-**Const definition**.
- שמות של מערכים קבועים יכתבו **ללא תחיליות** כלל (לא של סוג הטיפוס, ולא תחילית ״arr״).
- שמות של מערכים קבועים יכתבו **באותיות גדולות**, כשמילים מופרדות באמצעות **קו תחתון**.
- חובה **לאתחל** את כל התאים במערך קבוע בעת הגדרתו.

פרק 6 - שגרות

כללי

השגרות הן החלק העיקרי ביותר בתוכנית אמיתית. לא נוכל להבין באמת את מהלכה של תוכנית, מבלי להבין מה השגרות שבתוכה מבצעות בכלליות, ולכן הגדרת כללים לשגרות הינה חיונית מאוד לקריאות של התוכנית כולה.

שם שגרה

לעיתים (כמו במקרים שמקבלים מודולים מקומפלים), שמה של השגרה יהיה הדבר שמייצג אותה, ולכן חובה שהוא יבהיר את פעולתה.

- שם השגרה צריך להיות כמה שיותר קצר, קולע וברור.
- האות הראשונה בכל מילה בשם השגרה תהיה אות גדולה.
  - שם שגרה **לדוגמה** : PerformCalculation.

פרמטרים לשגרה

כדי להשתמש בשגרה חשוב להבין את משמעות הפרמטרים שנשלחים אליה ואת השפעתם על פעולתה. לכן חשוב ששגרה תקבל פרמטרים שמשמעותם ברורה (שמה של השגרה ושמות הפרמטרים הם קריטיים בהבנתה). נקפיד על כך ששגרה תקבל רק את מספר הפרמטרים שדרושים לה.

סוגי פרמטרים

טעות נפוצה שכתיבה סטנדרטית יכולה למנוע היא שינוי ערך משתנה in בשגרה.

- כאשר נכתוב פרמטרים לשגרה, תמיד האות הראשונה תציין האם הפרמטר הוא מסוג in או out.
  - למשל שם סטודנט כפרמטר קלט **(in)** לשגרה יקרא iszStudentName.
  - מונה ספירה לאחור שמשתנה בתוך השגרה **(out)** יקרא onFinalCountdown.
  - מספר **in/out** יקרא ionLastNumber.
  - **שימו לב!**
- שמות הפרמטרים מכילים גם את התחיליות הרגילות של הטיפוס.
- שמות פרמטרים קבועים יהיו לפי הסטנדרט המתאים לקבועים ולא לפי החוקים הנ״ל!

עימוד פרמטרים

רק במידה ויש צורך בירידת שורה בהגדרת שגרה ובהצהרה עליה (כלומר מעל 90 תווים), כל פרמטר יועמד בשורה נפרדת ובדיוק מתחת לפרמטר הקודם. כנ״ל לגבי הארגומנטים בקריאה לשגרה.

ערכי החזר של שגרה

- לפני פקודת return תבוא שורת רווח.
- ערך ההחזר יתוחם בסוגריים.

לפניכם **שגרה לדוגמה** - שימו לב שניתן להבין בכלליות את פעולתה כבר לפי ערך ההחזר שלה, שמה ושמות המשתנים שלה. כמו כן, שימו לב לאופן **עימוד הפרמטרים** ואיך נראית **פקודת ההחזר**.

שגרה לדוגמה

```
1    //---------------------------------------------------------------------------
2    //                                Sum Grades
3    //                                ----------
4    //
5    // General      : Summing all numbers of an array.
6    //
7    // Parameters   :
8    //        iarrnGrades   - The numbers array to sum (In)
9    //        onAverage     - The numbers average (Out)
10   //
11   // Return Value : The sum.
12   //
13   //---------------------------------------------------------------------------
14   int SumGrades(int iarrnGrades[NUM_OF_GRADES],
15                 int onAverage)
16   {
17        // Variable definition
18        int nGradesCounter;
19        int nSumOfGrades = 0;
20
21        // Code section
22
23        // Running through all the grades and summing them.
24        for (nGradesCounter = 0; nGradesCounter < NUM_OF_GRADES; ++nGradesCounter)
25        {
26             nSumOfGrades += iarrnGrades[nGradesCounter];
27        }
28
29        // Calculates average to return via out parameter
30        onAverage = nSumOfGrades / NUM_OF_GRADES;
31
32        return (nSumOfGrades);
33   }
```

מאגר כללי שגרות

בחלק זה נאגד לנוחיותכם את הכללים שהופיעו בפרק זה. דוגמאות לדברים מורכבים תמצאו בתוך הפרק.

- שם השגרה צריך להיות כמה שיותר **קצר**, קולע וברור.
  - האות **הראשונה** בכל מילה בשם השגרה תהיה אות **גדולה**.
- כאשר נכתוב פרמטרים לשגרה, תמיד **האות הראשונה** תציין האם הוא מסוג in או out.
- לפני פקודת **return** תבוא שורת רווח.
- ערך **ההחזר** יתוחם בסוגריים.

נספחים

נספח 1 - סדר הגדרת מודול

1. הוראות #include, קודם כל של המערכת, ולאחר מכן של מודולים אחרים בתוכנית.
2. שימוש ב-namespaces (using namespace)
3. הוראות #define
4. הצהרת typedef
5. הצהרת enum
6. הצהרת struct
7. הגדרת קבועים גלובאליים
8. הגדרת משתנים גלובאליים
9. prototypes לשגרות
10. הערה ראשית
11. תוכנית ראשית
    - 11.1. הגדרת קבועים
    - 11.2. הגדרת קבצים
    - 11.3. הגדרת מערכים
    - 11.4. הגדרת משתנים
    - 11.5. חלק ביצועי
12. שגרות (לפי סדר הצהרתן)
    - 12.1. הגדרת קבועים
    - 12.2. הגדרת קבצים
    - 12.3. הגדרת מערכים
    - 12.4. הגדרת משתנים
    - 12.5. חלק ביצועי

- במקרה הצורך (כמו typedef שמשתמש בטיפוס שהוא struct) ניתן לשנות את סדר ההגדרות.
- שימו לב בדוגמה שבנספח 3, שאין לרדת שורה אחרי ההערות להגדרת אזור (מלבד אחרי Code section, שלפניו ואחריו יש שורת רווח).
- במידה ונשתמש בקבוע גלובלי במבנה נהפוך את ההגדרות ונגדיר מבנה אחרי קבוע גלובלי.

נספח 2 - קידומות סטנדרטיות

Regular Prefixes						
Type	**Prefix #1**	**Info**	**Prefix #2**	**Info**	**Prefix #3**	**Info**
short	**s**					
int	**n**					
long	**l**					
float	**f**					
double	**d**					
unsigned int	**un**					
char	**c**					
unsigned char	**uc**					
bool	**b**					
int[]	**arrn**					
char[]	**arrc**	Array of chars	**sz**	String (zero-terminated)		
int*	**pn**	Pointer to integer	**arrn**	Array of integers		
int**	**ppn**	Pointer to Pointer to integer	**parrn**	Pointer to array of integers		
char*	**pc**	Pointer to char	**arrc**	Array of chars	**sz**	String (zero-terminated)
char**	**ppc**	Pointer to Pointer to char	**parrc**	Pointer to array of chars	**psz**	Pointer to string
fstream	**fs**					
fstream*	**pfs**					
Global	**g_**					
Struct Example						
Position	**pos**	struct				
Position*	**ppos**	p->struct				
Location	**loc**	struct				
Location*	**ploc**	p->struct				
Function Parameters (to be added in addition to regular prefix)						
in	**i**					
out	**o**					
in / out	**io**					

- יש להתייחס למטריצות כמו מערכים מבחינת תחיליות.

נספח 3 - מודול לדוגמה

מודול לדוגמה

```
1   // FileName.cpp
2
3   #include <iostream>
4   #include <fstream>
5   #include "OtherFile.h"
6
7   using namespace std;
8
9   #define EXAMPLE_DEF         1800
10  #define SOME_STRING_EXAMPLE "This is an example"
11
12  // Typedef declaration
13  typedef int  NewInt;
14  typedef char NewChar;
15
16  // Enum declaration
17  enum EnlistedRanks
18  {
19    PRIVATE                = 91,
20    CORPORAL               = 85,
21    SERGEANT               = 82,
22    STAFF_SERGEANT         = 79,
23    SERGEANT_FIRST_CLASS   = 76,
24    MASTER_SERGEANT        = 73,
25    FIRST_SERGEANT         = 71,
26    SERGEANT_MAJOR         = 70,
27    COMMAND_SERGEANT_MAJOR = 69
28  };
29
30  enum OfficerRanks
31  {
32    SECOND_LIEUTENANT  = 30,
33    LIEUTENANT         = 27,
34    CAPTAIN            = 21,
35    MAJOR              = 18,
36    LIEUTENANT_COLONEL = 15,
37    COLONEL            = 12,
38    BRIGADIER_GENERAL  = 9,
39    MAJOR_GENERAL      = 6,
40    LIEUTENANT_GENERAL = 3
41  };
42
43  // Struct declaration
44  struct Officer
45  {
46    char*         pszName;
47    OfficerRanks  orRank;
48  };
49
50  struct Enlisted
51  {
52    char*             pszName;
53    EnlistedRanks     erRank;
```

```
54  };
55
56  // Global const definition
57  const char NEW_MESSAGE[] = "Programming is like hot water";
58
59  // Global variable definition
60  const int g_nOne = 1;
61  const int g_nTwo = 2;
62
63  // Function prototypes
64  void FirstProc();
65  char SecondFunc(int  inFirstNum,
66                  int* iopnSecondNum);
67
68  //*****************************************************************************
69  //                                 Example Program
70  //                                 ---------------
71  //
72  // General : This program does absolutely nothing.
73  //
74  // Input   : None.
75  //
76  // Process : Nothing real.
77  //
78  // Output  : Useless messages.
79  //
80  //*****************************************************************************
81  // Programmer   : Sergeant G.C Salah
82  // Student No   : 1203
83  // Date         : 31.03.1988
84  //*****************************************************************************
85  void main()
86  {
87    // Const definition
88    const int ARRAY_SIZE = 1203;
89
90    // File definition
91    fstream fsNames;
92
93    // Array definition
94    int arrnJustNumbers[ARRAY_SIZE];
95    int arrnNotOnlyNumbers[ARRAY_SIZE];
96
97    // Variable definition
98    bool bIsExampleOver = false;
99
100   // Code section
101
102   FirstProc();
103 }
104
105 //*****************************************************************************
106 //                                 First Proc
```

```
107 //                                  ----------
108 //
109 // General      : This procedure says that it's a procedure.
110 //
111 // Parameters   : None.
112 //
113 // Return Value : None.
114 //
115 //---------------------------------------------------------------------------
116 void FirstProc()
117 {
118   // Const definition
119   const int ARRAY_SIZE = 3;
120
121   // Array definition
122   int arrnResults[ARRAY_SIZE] = {1, 24, 55};
123
124   // Code section
125
126   cout << "I'm a procedure" << endl;
127 }
128
129 //---------------------------------------------------------------------------
130 //                                  Second Func
131 //                                  -----------
132 //
133 // General      : An example of a second function in a row.
134 //
135 // Parameters   :
136 //     inFirstNum    - A meaningless number (In)
137 //     iopnSecondNum - A pointless pointer (In/Out)
138 //
139 // Return Value : Nothing real...
140 //
141 //---------------------------------------------------------------------------
142 char SecondFunc(int  inFirstNum,
143                 int* iopnSecondNum)
144 {
145   // Code section
146
147   return ('\0');
148 }
```