# קונבנציות C++

## הקדמה

### מטרת המסמך

קביעת הנחיות וסטנדרט של כתיבת קוד בשפת C++ על מנת לשמור על קוד נקי, קריא וקל לתחזוקה.

### חשיבות קונבנציות

קונבנציות משמשות כאבני-דרך וכבסיס אחיד לכתיבת קוד. ככאלה, הן תורמות לשמירה על סטנדרט קוד אחיד, להפחתת שגיאות, לתחזוקה קלה, להגברת שיתוף הפעולה בין חברי הצוות ולשיפור קריאות הקוד.

## הנחיות כלליות

### סטנדרט

הקוד ייכתב בסטנדרט 2020 (C++20).

### קריאות הקוד

חשוב להבטיח שהקוד יהיה אלגנטי, קריא וברור. יש להשתמש בשמות מתאימים ובתצורות קוד שניתן להבין בקלות.

### הערות (Comments)

הערות קוד בנות 3 שורות לכל היותר יסומנו ב-// או ב-/\*…\*/.  
הערות קוד בנות 4 שורות ומעלה יסומנו ב-/\*…\*/.

### Commit Messages

יש להקפיד על Commit Message מתומצת אך אינדיקטיבי, כך שהקורא יבין מה שונה בקוד באותו commit.

### Branches

יש לתחזק branch ראשי (master), שהוא הגירסה שרצה ב-production.  
כל עבודת קוד (feature חדש/תיקון באג/עידכון קונפיגורציה/refactoring) תיכנס במסגרת branch ייעודי לה לתוך master.  
אין לערבב בין עבודות בתוך branch בודד.

שם branch ייכתב בפורמט הבא: <job type>/<job description>.  
job type יהיה feature/fix/config/refactor עבור feature חדש/תיקון באג/עידכון קונפיגורציה/refactoring, בהתאמה.  
job description יהיה קצר ואינדיקטיבי, במלל חופשי.

### תיעוד

Flow-ים מרכזיים וקטעי קוד מורכבים יתועדו במסמך חיצוני, או בכלי תיעוד קוד אחר (כדוגמת Confluence).  
"קטעי קוד מורכבים": מכניזם תשתיתי, design pattern, מבנה נתונים מיוחד, וכדומה.  
קטעי קוד שאינם מורכבים: תיקונים של באגים, refactoring לא-משמעותי, שינויי quality-of-life, וכדומה.

### תשתית Version Control

התשתית שבה משתמשים היא Git, בצירוף עם BitBucket.

### ראייה עתידית

בעת תיכנון קוד (פשוט וקצר או מורכב וארוך), יש להקדיש מחשבה ולשים לב שהקוד יהיה תואם את העתיד הקרוב והסביר.

## אירגון הקבצים

### שיום (Naming) קבצים

קבצי source יהיו בעלי סיומות .cpp או .h.  
קבצי קונפיגורציה יהיו בעלי סיומת .cfg.  
קבצי טקסט יהיו בעלי סיומת .txt.

### מבנה עץ התיקיות

קבצי source יהיו תחת תיקיית src.  
קבצי test יהיו תחת תיקיית test.  
ככלל, למעט main.cpp, כל קובץ יהיה בתוך תיקייה בעלת שם אינדיקטיבי כלשהוא.

## שיום (Naming)

### משתנים (Variables) ופונקציות (Functions)

השמות יהיו אינדיקטיביים בהתאם למטרת המשתנה או הפונקציה.  
  
אין לקצר שמות על חשבון קריאותם, לדוגמה: יש לכתוב userCounter ולא userCnt.  
עם זאת, יש להפעיל שיקול דעת לגבי אורך השם, ולהיעזר בהקשר, לדוגמה: אין לכתוב numberOfItemsInShoppingCart, אלא itemCount, שהוא קצר יותר, ומובן בהתאם להקשר.  
  
ניתן לבחור בין שיטת camelCase לבין שיטת snake\_case בקוד חדש.  
בקוד ישן, יש להיצמד לשיטה הקיימת.

### מחלקות (Classes) ומבנים (Structs)

שמות מחלקות ייכתבו בשיטת PascalCase, לדוגמה: MyClass.

### קבועים (Constants)

שמות קבועים ייכתבו באותיות גדולות, בשילוב עם קו תחתון, לדוגמה: SOME\_CONSTANT.

## תבנית קוד (Code Formatting)

### הזחה (Indentation)

רמת הזחה תהיה בגודל של tab אחד, או 4 רווחים.

### אורך שורה

אורך שורת קוד לא יעלה על 120 תווים.

### סוגריים מסולסלים

סוגריים מסולסלים פותחים יבואו בשורת הקוד הפותחת אותם, לדוגמה:

if (…) {  
 // code  
}

### ריווח (Spacing)

יש להוסיף רווח אחד מסביב לאופרטורים ולאחר ,.

### כוכבית

כוכבית תבוא בצמוד לסוג המשתנה, ולא לשמו, לדוגמה: int\* number, ולא int \*number.

### קבצי Header

בראשית כל קובץ Header ייכתב #pragma once.

## שיטות מועדפות (Best Practices)

### טיפול בשגיאות

יש להעדיף Exceptions במקום Return Codes.  
יש להעדיף Exceptions סטנדרטיים, לדוגמה: std::runtime\_error, std::logic\_error.  
אין להשתמש ב-exceptions עבור flow תקין.  
יש להשתמש ב-std::optional וב-std::expected על מנת להתייחס לערך שייתכן שאינו קיים.

### ניהול זיכרון

יש להשתמש במצביעים חכמים (Smart Pointers):  
std::unique\_ptr, std::shared\_ptr, std::weak\_ptr.  
יש להעדיף מבני נתונים מובנים בשפה (STL Containers), על פני מימוש עצמי.

### שימוש ב-const

יש להשתמש ב-const וב-constexpr בכל מקום רלוונטי.  
יש להשתמש ב-std::string\_view בכל מקום רלוונטי.

### שימוש ב-auto

אין להעדיף קריאות על פני קוד ברור, ולכן יש להשתמש ב-auto במקומות בהם סוג המשתנה ברור.  
במקומות בהם סוג המשתנה לא ברור, יש לכותבו באופן מפורש.

### שימוש ב-explicit

יש להשתמש ב-explicit עבור כל constructor בעלי ארגומנט יחיד.

### לולאות

יש להעדיף שימוש ב-ranged-based loops, לדוגמה:

for (auto num : vec)

ולא

for (auto it = vec.begin(); it != vec.end(); ++it)

### פונקציות Lambda

יש להשתמש בפונקציות lambda באופן קריא וקצר.  
לקטעי קוד ארוכים יותר, יש להעדיף פונקציות רגילות (named functions).

### ספרייה סטנדרטית

ככלל, יש להעדיף שימוש בספריית C++ הסטנדרטית וב-features המובנים של השפה, על פני ספריות חיצוניות.

### איתחול (Initialization)

ב-constructor, יש להעדיף שימוש ככל הניתן ב-initializer list, על פני איתחול בגוף ה-constructor.  
יש לאתחל משתנים בעזרת סוגריים מסולסלים ({}) בלבד.

## בדיקות (Testing)

### בדיקת יחידה (Unit Test)

יש לכתוב טסט עבור כל לוגיקת קוד חדשה ופונקציונאליות משמעותית.

### בדיקת מערכת (System Test)

יש לכתוב טסט עבור בדיקת מספר רכיבים או מודולים כאחד, כדי לוודא את ה-flow של הקוד ואת התקשורת ביניהם.

## סקירת קוד (Code Review)

יש לבצע Code Review איכותי, ע"י חבר צוות.  
"Code Review איכותי" יבחן: עמידה בקונבנציות, flow מרכזי תקין, מקרי קצה, השפעה על רכיבי מערכת אחרים ועמידות (קוד "צופה פני עתיד").