**טמטפלטים – תכנות המשאיר פתוח את סוג המשתנה**

**קוד טמפלטי הוא מקרו - אין לו השתקפות בינארית.**

**לפיכך אם רוצים לממש טמפלטים בקובץ נפרד, חייבים לעשות לו #include, המסורת לעשות בקובץ.hpp**

**יש 2 סוגים עיקריים של טמפלטים:**

* **טמפלטים של פונקציות**
* **טמפלטים של מחלקות**

**דוגמא לטמפלט של פונקציה:**

**template<class T>**

**inline T my\_sqr(T x)**

**{**

 **return x\*x;**

**} // my\_sqr**

**אפשר גם**

**template<typename T>**

**inline T my\_sqr(T x)**

**{**

 **return x\*x;**

**} // my\_sqr**

**טםמךטים של מחלקות: לבדיל מטמפלטים של פונקציות קוד הלקוח מעיד על כך שאנחנו משתמשים בטמפלטים.**

**לדוגמא:**

 **stack<float> Float\_Stack(5);**

 **stack<int> Int\_Stack(5);**

**פונקציות טמפלטיות – מאפשרת כתיבת אב טיפוס לאלגוריתם.**

**למשל**

**template<class T>**

**void bublesort(T vec[], int n) /\*\*\*\*\*/**

**{**

**חריגות ב-C++**

* **תוספת מאוחרת לשפה**
* **חריגות מערכת (כמו חלוקה באפס) לא נכללות בקטגורה הזו (אלא בסיגנים של C). יוצא מן הכלל הוא כשלון של new.**
* **בשפה הזו, אפשר לזרוק כל דבר (int, float ...)**
* **חריגות ב-C++ כמעט תמיד ביוזמת המתכנת.**

**חברי מחלקה סטטיים**

**מי שרוצה יכול לעבוד ב-C++ בדיוק כמו ב-Java, C#.**

**זה מקל על תרגום תוכניות Java, C# ל-C++.**

**למשל:**

  **static complex complex\_add(complex x,**

 **complex y);**

 **c = complex::complex\_add(a,b);**

**תאור מאוחר**:

**complex complex::complex\_add(complex x,
 complex y)
בלי מילת המפתח static**

**שדות mutable**

**מחריגים משתנים מההגנה של const.**

**התמרות משתמעות implicit promotions**

**אם הקומפילר נתקל בקוד שלא מומש בכל הקשור להעברת פרמטרים לפונקציה או מתודה, הוא מנסה לממש את הקוד על ידי חיפוש בנאי היכול לקבל פרמטר אחד.**

**זה לא במיוחד שווה משום זה יכול ליצור קוד דו משמעי.**

**אפשר לוותר על המנגנון הזה ע"י הגדרת בנאים עם מילת המפתח explicit.**

**בנאים ב-java, C# באופן משתמע תמיד explicit.**