**שפת התכנות C++**

* **שפה די עשירה בכל הקשור לאמצעי תכנות.**
* **שפה שתומכת ב-OOP אך אינה נחשבת לשפה OOP.**
* **שפה מורחבת של C, כוללת את אמצעי התכנות של C. תוכנית חוקית ב-C תוכל לעבור להיות תוכנית חוקית ב-C++ במאמץ מינימלי.**
* **השפה כוללת פוינטרים, כירושה מ-C, אבל תכנות OOP עשוי להשתמש בפוינטרים. למשל הקצאה דינמית מחייבת שימוש בפוינטרים.**
* **השפה מלאה מלכודות מנקודת ראות של הנדסת תוכנה למי שלא לומד אותה בצורה מסודרת.**
* **בדומה לשפת C, לשפת C++ יש סטנדרד, קומפילרים ל-C++ יש כמעט בכל מסגרת נפוצה, והם מקיימים את הסטנדרד, אבל למען האמת השפה, כמו C, איננה פורטבילית באמת.**
* **הסטנדרד אינו קובע עמדה בכל הקשור ל-GUI ולהקצאת משאבים מהמערכת.**
* **בניגוד ל-Java שיש לו את ה-JVM/JRE ל-C# שיש לו את ה-.NET ל-C++ אין סביבת ריצה הוא מדבר ישירות עם המערכת דרך קריאות מערכת.**

**משתני יחס**

**כמו בשפות הקודמות, משתני יחס הם מצביעים מוגבלים אולם:**

**בשפה הזו משתני יחס מקבלים ערך רק באתחול (כאשר המשתנה נוצר).**

**מאותו רגע, השמה למשתנה יחס הוא רק לזיכרון עליו הוא מצביע.**

**זה הופך משתני יחס למשהו שימושי רק כפרמטרים (שם הוא נוצר מחדש בכל קריאה(.**

**ב-C++ מערכים הם בסך הכול כתובת התחלה. בגישה למערך לא נבדק אם אתה חורג ממנו.**

**פונקציה שמחזירה יחס, נראית כמי מחזירה נתון בהשמה מימין (מקור) ומחזירה פוינטר בהשמה משמאל (יעד).**

**ב-C++ כמו בשפות הקודמות חתימה של פונקציה כולל את רשימת סוגי הפרמטרים.**

**כאשר אנחנו רוצים לקרוא מקוד C++ רוטינה שנכתבה ב-C או באסמבלי בדרך כלל נכריז**

**extern "C" int fun(int);**

**ה-"C" אומר שהפונקציה מקיימת את מוסכמות הקריאה של C ולא C++.**

**אם עושים את זה, לא ניתן להעמיס את שם הפונקציה יותר.**

**כאשר יש פרמטר עם ברירת מחדל, לכל הפרמטרים שבאים אחריו חייבים להיות גם ערכי ברירות מחדל.**

**פונקציות inline**

**מעין מנגנון מקרו משופר לעומת המקרו של C.**

**בדרך כלל קומפילרים מגבלים מה יכול להופיע בפונקצית inline.**

**אם הקוד לא עונה למגבלות, הפונקציה תמומש כפונקציה רגילה.**

**תכנות OOP ב-C++**

**מקובל ע"י הגדרת class-ים.**

**למרות להשתמש גם ב-struct.**

**לעומת C,**

**הכרזת class או struct יוצרת ישר מעין typedef.**

**אם ב-C**

**struct myrec**

**{**

**…**

**};**

**struct myrec rec1;**

**ב-C++ אוכל לעשות**

**myrec rec1;**

**בגלל זה, הקוד הבא חוקי ב-C ולא ב-C++:**

**typedef struct myrec**

**{**

**…**

**} myrec;**

**myrec rec1;**

* **ב-C++ ניתן בהגדרת class או struct להכריז (ואופציונלית לממש) מתודות.**

**הבדלים מהותיים נוספים מהשפות הקודמות (Java, C#):**

* **רמת הנגישות מוגדרת ע"י ליבלים:**

**public:**

**private:**

**protected:**

* **ההבדל הטכני היחיד בין struct ל-class הינו ברירת המחדל של רמת הנגישות: ב-struct זה public וב-class זה private.**
* **ב-C++ אפשר בהגדרת המחלקה רק להכריז על מתודות ולממש אותם בנפרד.**
* **שיטת העבודה הקלאסית:**

**הכרזת המחלקה בקובץ .h או .hpp**

**פירוט המתודות בקובץ cpp.**

**קוד לקוח בקובץ .cpp**

**קוד טמפלטים (במושגים של השפות האחרות קוד ג'נרי) בקבצי .hpp**

* **מופעים של מחלקה בדרך כלל לא מוקצים דינמית אלא על המחסנית:**

**stack My\_Stack(5);**

**גם הכרזת משתנה, גם הקצאת המופע וגם קריאה לבנאי עם פרמטר שלם 5.**

* **אם מקצים שטח דינמי במהלך חייה של מופע של מחלקה אין משחרר אוטומטי של שטחי זיכרון דינמיים (גם אם אחד לא מצביע עליהם יותר). שחרור שטחים דינמיים באחריות המתכנת ע"י מימוש פונקציה הורסת destructor:**

**~stack(); // destructor**

**השפה מחויבת לקרוא לפונקציה ההורסת עם שחרור המופע.**

* **מערכים דינמיים חייבים לשחרר יחד עם האופרטור [].**
* **ברוב הקומפילרים טעות בפקודת השחרור איננה גורמת לשום תגובה.**
* **הקצאות דינמיות מחייבות שימוש בפוינטרים.**

**פונקציות גלובליות:**

* **אלטרנטיבה לפונקציות סטטיות של Java ו-C#.**
* **פונקציות שלא קשורות ישירות למחלקות, אפשר לקרוא להם שלא דרך מופע כלשהוא.**
* **אם מוכרחים מסיבה כלשהיא לאפשר נגישות לשדות פרטיים אפשר להגדיר פונקציה גלובלית כ-friend.**