**ב-C++ הכרזת משתנה מחלקה מקצה אוטומטית מופע וקוראת לבנאי.**

**בשביל להקצות משתנה שיצביע למופע קיים או לכזה שיוקצה דינמית יש להכריז על פוינטר מסוג המחלקה.**

**המזהים המוגדרים בתוך המחלקה הם מעין "משתנים גלובליים" של מתודות המחלקה כל עוד אין התנגשות שמות.**

**בשפה הזו this הוא פוינטר המצביע למופע (ולא reference כמו ב-Java, C#).**

**התמודדות עם התנגשות שמות:**

1. **הכרת מזהים גלובליים: ניתן לבקש מהקומפילר להתעלם מהמזהים המוגדרים במחלקה ע"י האופרטור :: ללא קדם.**
2. **התנגשות בין הכרזה לוקלית לבין מזהה של המחלקה: ברירת המחדל היא ההכרזה הלוקלית.**
* **ניתן לעקוף את ההכרזה הלוקלית ע"י this-> או (\*this)..**
* **ניתן לעשות זאת גם עם**

**שם המחלקה::**

**העברת מופע של מחלקה כפרמטר.**

1. **אם מעבירים מופע של מחלקה כפרמטר "רגיל" המופע מועבר by value קרי: המופע משוכפל למשתנה כפיל שלו שנדחף למחסנית.**
2. **ההעתקה לכפיל היא בית בית.**
3. **במחלקות עם מופעים גדולים זה לא יעיל.**
4. **זה גם לפעמים עושה צרות רציניות.**
5. **העברת הכתובת בתור פרמטר או העברת יחס למופע פותרת את הבעיות הללו אבל מאפשרת לרוטינה הנקראת לשנות את המופע.**
6. **הפתרון היעיל ביותר שקיים ב-C++ הוא להעביר פרמטר reference שהוא const (אפשר גם פוינטר const). בשביל זה צריך לתכנת בהתאם.**

**על פרמטר שהוא const אפשר לממש קוד ש-:**

* **רק קורא שדות של המחלקה**
* **להפעיל עליו רק מתודות בעלות מאפיין const.**

**מתודות const הן מתודות (לא סטטיות) שהקומפילר מוודא שאינן משנות את מופע המחלקה.**

**הרגל תכנותי טוב: כל מתודה שאנחנו מממשים שאינו משנה מופע המחלקה, כדאי להגדיר אותו מתודת const אלא אם יש סיכוי שזה ישתנה בעתיד.**

**העמסת אופרטורים**

**(בניגוד ל-C#) כל סוגי העמסת אופרטורים הם במסגרת פורמלית אחת.**

**רוב האופרטורים ניתנים להעמסה בשתי צורות: כמתודה ופונקציה גלובלית.**

**חלק מהאופרטורים יכולים להיות מועמסים רק כמתודות.**

**(ב-C# השימוש ב-operator תמיד מגדיר מתודה סטטית מקביל פונקציה גלובלית כאן,**

**העמסת [ ], =, () מוגרות אחרת מקביל כמתודות כאן).**

**כאשר מעמיסים אופרטור כמתודה הפרמטר השמאלי משתמע אוטומטית הוא המופע של המחלקה**

**זה הופך העמסת אופרטור כמתודה כמשהו "פחות כללי" מהעמסה כפונקציה גלובלית.**

**Class Complex**

**{**

**};**

**int main()**

**{**

**Complex c1,c2, c3;**

**double x, y;**

**c2 = c1 +x; // ניתן לממש**

**גם כמתודה וגם כפונקציה גלובלית**

**c3 = y +c2; // ניתן למימוש**

**רק כפונקציה גלובלית**

**}**

**לעיתים קרובות מפתחי מחלקות חדשות אוהבים להדפיס תוכן של מחלקה ע"י האןפרטור ">>" כמו שעושים למספרים.**

**יש דרך מסורתית לעשות זאת:**

### **ostream& operator << ( ostream& stream, class שם המחלקה החדשה**)

}

 פירוט הקוד פלט

return stream;

{

**כנ"ל לגבי קלט עם "<<"**

### istream& operator >> ( istream& stream, class& שם המחלקה החדשה)

}

 פירוט הקוד קלט

return stream;

{

**ב-C++ לפחות לכל הקשור לאופרטור ההשמה "=" העמסת היא לא רק צעצוע.**

**לפעמים היא חובה המוטלת על מתכנתים לפחות בכל הקשור למחלקות שמכילות הצבעות לשטחים דינמיים.**

**מקור הבעיה היא שברירת המחדל של ההשמה בין משתני מחלקה הן העתקות בית בית.**

**למשל**

**Class A**

**{**

**}**

**Int main()**

**{**

 **A a1, a2;**

 **a1 = a2; // מתבצע העתקה**

**בית בית**

**}**

**מחלקות המכילות מצביעים לשטחים דינמיים:**

* **אופרטור ההשמה ברירת מחדל יגרום לשטחים הללו להיות משותפים גם כאשר איננו מעוניינים בכך.**
* **הפונקציה ההורסת של אחד מהם ישחרר את הטחים המשותפים.**
* **הבעיה הזו קיימת גם בהעברת פרמטרים by value: קיים מה שנקרא "בנאי העתקה ברירית מחדל" שיוצר כפיל ע"י העתקה בית בית.**
* **בנאי העתקה ברירת מחדל נקרא ב-:**
1. **העברה by value**
2. **החזרת מופע מחלקה כתוצאת פונקציה**

**return a;**

1. **קריאה מפורשת**

**Vector(int n)**

**{**

 **len = n;**

 **buff = new float[n];**

**} // Vector(int n)**

**שימוש ברשימות אתחול**

**Vector(int n): len(n), buff(new float [n]) {};**

**class A**

**{**

**};**

**main()**

**{**

 **A a1;**

 **A a2(a1); //קריאה מפורשת**

 **לבנאי העתקה**

**}**

**כלל:**

**אם מחלקה מכילה מצביעים לשטחים מוקצים דינמית, על המתכנת מוטלת אחריות אוטומטית:**

* **לממש פונקציה הורסת**
* **לדרוס את אופרטור ההשמה ברירת מחדל**
* **לדרוס את בנאי העתקה ברירת מחדל.**
* **ב-C++ שום דבר לא משתחרר אוטומטית: על התוכניתן לממש פקודת שחרור לכל שטח שהקצה.**
* **צריך תמיד לבדוק לפני שחרור אם יש מה לשחרר.**

**הורשה ב-C++**

**ב-C++ יש מאפיין שנקרא סוג ההורשה.**

**הוא מגדיר את רמת הנגישות המרבית של השדות של האב עבור קוד הלקוח (והמחלקות הנגזרות) של הבן.**

**למה זה לא שווה?**

**כי כל סוג הורשה למעט public לא יאפשר לבן לדרוס מתודות של האב.**

**שפת C++ תומכת ב"הורשה מרובה" – מחלקה יכולה לרשת מכמה מחלקות מלאות.**

**בגלל זה אין בשפה הזו מילת מפתח המקבילה ל-super או base: ההתייחסות למחלקות האב לפי שם המחלקה.**

**דריסה סטטית או ויטואלית:**

* **פוינטר או משתנה יחס מסוג האב יכול להצביע בפועל על מופע מסוג הבן.**
* **במידה ופוינטר/משתנה יחס מסוג האב קורא למתודה קיימת באב ונדרסת בבן, יתבצע קריאה:**
1. **אם המתודה מוגדרת באב כ-virtual הקריאה תהיה לפי המופע (המתודה של הבן).**
2. **אם המתודה מוגדרת לא באב כ-virtual הקריאה תהיה לפי ההגדרה של המשתנה (כלומר זה של האב).**
* **ברגע שמתודה מוגדרת באב הקדמון כ-virtual היא תהיה virtual לכל הצאצאים.**
* **אם המתודה לא מוגדרת כ-virtual באב הקדמון, לא ניתן להתחרט על כך.**

**שימוש ברשימות אתחול**

**Vector(int n)**

**{**

 **len = n;**

 **buff = new float[n];**

**} // Vector(int n)**

**Vector(int n): len(n), buff(new float [n]) {};**