המרות סוגים ומידע על סוגים בזמן ריצה

בשפת סי, כשרוצים להמיר סוג, משתמשים בסוגריים. יש הרבה סוגים של המרות וכולן מתבצעות באותו אופן. למשל:

```
double d = 3.0;  int i = (int) d;   // המרת מספרים

const int* cp = &i;   int *ncp = (int*)cp;  // המרה עוקפת קונסט

double* dP = (double*)ncp;  // המרת פוינטר מסוג אחד לפוינטר מסוג אחר

Base *baseP1 = new Derived;  Derived *derP1 = (derP*)baseP1; // המרת
פוינטר בסיס שמצביע למחלקה יורשת, לפוינטר מהסוג הנכון.

Base *baseP2 = new Base;  Derived *derP2 = (derP*)baseP2; // המרת פוינטר
בסיס שמצביע למחלקת בסיס, לפוינטר מהסוג הלא-נכון.
```

בשפת ++C, המרות מתבצעות ע"י אופרטורים מיוחדים - לכל סוג של המרה יש אופרטור עם שם משמעותי, שעוזר לקורא להבין איזו המרה בדיוק מתבצעת כאן. אנחנו נציג אותם מהנדיר לנפוץ.

---

const_cast

האופרטור const_cast משמש להמרת מצביע או רפרנס קבוע (const) למצביע/רפרנס רגיל (לא const). הוא למעשה אומר לקומפיילר להתעלם מהבדיקה של const.

אופרטור זה משמש במקרים נדירים מאד. בדרך-כלל, שימוש באופרטור זה מראה על שגיאה בתיכנון המחלקה. צריך לתכנן את המחלקה נכון כך שכל מה שצריך להיות const אכן יהיה const.

מתי בכל-זאת משתמשים בו? כשאנחנו מקבלים קוד ישן של מתכנת שעשה שגיאה ולא סימן פונקציה מסויימת כ-const למרות שבפועל היא כן const (לא משנה את העצם שהיא מקבלת כפרמטר). אם יש לנו גישה לקוד המקור - נשנה אותו ונסמן את הפונקציה כ-const. אבל לפעמים אין לנו גישה לקוד המקור אלא רק לקובץ הבינארי המקומפל. במקרה זה, אנחנו משתמשים בconst_cast כדי להגיד לקומפיילר "סמוך עלינו, בדקנו ואנחנו יודעים שהפונקציה לא משנה את הארגומנט שלה, יהיה בסדר".

כמו במקרים רבים אחרים בחיים, כשמישהו אומר "סמוך עליי" זה עלול להיות פתח לצרות... לכן בדרך-כלל לא נשתמש בהמרה זו.

reinterpret_cast

האופרטור reinterpret_cast משמש להמרת פוינטר או רפרנס מסוג אחד לסוג אחר. האופרטור לא מבצע שום בדיקה - לא בזמן קימפול ולא בזמן ריצה. כמו const_cast, הוא למעשה אומר לקומפיילר להתעלם מבדיקות הטיפוסים הרגילות שהוא מבצע, ולסמוך עלינו שאנחנו יודעים מה אנחנו עושים. זה מסוכן ועלול לגרום לשגיאות לוגיות וערכים לא מוגדרים. ראו תיקיה 4.

אם reinterpret_cast כל-כך מסוכן, למה בכל-זאת משתמשים בו? שימוש לגיטימי הוא כשרוצים לתרגם מחלקה כלשהי לבינארית לצורך כתיבה לקובץ בינארי (למשל קובץ תמונה). ראו דוגמה בתיקיה 5.

static_cast

האופרטור static_cast משמש להמרה המתבצעת ע"י פונקציה ידועה בזמן קומפילציה. לדוגמה:

```
double d = 12.45;
int i = static_cast<int>(d);
```

זו המרה של מספר ממשי למספר שלם, המתבצעת ע"י פעולה ידועה - לקיחת החלק השלם בלבד. באותו אופן אפשר להמיר מספר שלם למספר ממשי.

למה זה עדיף על המרה בעזרת סוגריים כמו בסי -

```
int i = (int)d;
```

? משתי סיבות:

א. קל יותר למצוא המרות בקוד - פשוט מחפשים את המחרוזת static_cast.

ב. ההמרה בעזרת סוגריים עלולה להיות שגויה. למשל, בעזרת סוגריים אפשר להמיר מצביע למספר ממשי למצביע למספר שלם:

```
int* ip = (int*)dp;
```

ההמרה נראית בדיוק כמו קודם אבל היא שגויה - התוצאה תהיה זבל. לעומת זאת, אם נכתוב:

```
int* ip = static_cast<int*>(dp);
```

הקומפיילר יציל אותנו מנפילה לפח הזבל בכך שיכריז על שגיאת קומפילציה - אין המרה סטטית המאפשרת להמיר מצביע לממשי למצביע לשלם (ראו תיקיה 4).

איך זה עובד עם מחלקות?

- static_cast חוסם המרות בין מצביעי-מחלקות שאין ביניהן קשר (אף אחת לא יורשת מהשניה).
- static_cast מאפשר המרה של מצביע למחלקה יורשת אל מצביע למחלקת הבסיס - והמרה כזאת היא בטוחה (אבל היא מתבצעת אוטומטית גם בלי static_cast).
- static_cast מאפשר גם המרה של מצביע למחלקת בסיס אל מצביע למחלקה יורשת - והמרה כזאת היא **מסוכנת** - היא נכונה רק אם המצביע המומר אכן הצביע לעצם מהסוג של המחלקה היורשת. **לא מתבצעת** כל בדיקה לנושא זה ולכן זו סכנה - עדיף במקרה זה להשתמש ב-dynamic_cast שנלמד בהמשך.

שימו לב: כשמשתמשים ב-static_cast, בדרך-כלל הקומפיילר מכניס פקודה כלשהי שתתבצע בזמן ריצה. כשמשתמשים ב-const_cast או reinterpret_cast, הקומפיילר לא מכניס שום פקודה לביצוע בזמן ריצה - ההוראה משפיעה על הקומפילציה בלבד. לא מאמינים? בדקו ב-godbolt.org.

dynamic_cast

האופרטור dynamic_cast משמש להמרה של פוינטר/רפרנס למחלקת-בסיס עם שיטות וירטואליות, אל פוינטר/רפרנס למחלקה יורשת שלה. כשמשתמשים באופרטור זה, הקומפיילר מכניס בדיקה, המתבצעת בזמן ריצה, אם העצם שמצביעים אליו אכן מתאים לסוג שאליו רוצים להמיר.

לדוגמה, אם יש לנו מצביע מסוג Shape לעצם מסוג Circle, אפשר להמיר אותו ע"י המרה דינמית למצביע מסוג Circle. אבל, אם ננסה להמיר אותו למצביע מסוג Square, האופרטור יחזיר null.

אם ננסה להמיר רפרנס במקום מצביע - לא נקבל null (כי אין רפרנס כזה) אלא נקבל חריגה - bad_cast.

הבדיקה מתבצעת לפי סוג העצם בזמן ריצה. לכן, אפשר לבצע בדיקה זו רק אם העצם הוא ממחלקה שיש לה מידע על סוגים בזמן ריצה - כלומר מחלקה שיש לה טבלת שיטות וירטואליות. אם למחלקת הבסיס אין שיטות וירטואליות - הקומפיילר לא ייתן לנו להשתמש ב-dynamic_cast.

אפשר להשתמש ב-dynamic_cast כדי לדמות את אופרטור instanceof. ראו דוגמה בתיקיה 6.

שימו לב: dynamic_cast היא פעולה "יקרה" בזמן ריצה - היא צריכה לעבור על כל עץ הירושה כדי לקבוע אם ההמרה תקינה או לא.

מידע על סוגים בזמן ריצה - RTTI

האופרטור dynamic_cast משתמש במידע על סוג העצם בזמן ריצה.

אנחנו יכולים לגשת למידע הזה ישירות בעצמנו, בעזרת האופרטור typeid.

כשמריצים typeid על עצם, מקבלים מצביע למבנה המכיל מידע על סוג העצם, כגון השם שלו (השיטה name), האינדקס שלו, ועוד.

אם העצם הוא מסוג סטטי - המידע נקבע בזמן קומפילציה. אם העצם הוא מסוג פולימורפי (- צאצא של עצם עם פונקציות וירטואליות), אז הקריאה ל-typeid יוצרת פעולה בזמן ריצה הפונה לטבלה הוירטואלית ושולפת משם את סוג העצם. ראו דוגמה בתיקיה 6.

זמן הריצה - O(1) - לא צריך לעבור על כל עץ הירושה אלא רק ללכת לטבלה הוירטואלית של המחלקה הנוכחית.

שימו לב: אפשר להגיד לקומפיילר שלא ישמור בכלל מידע על סוגים בזמן ריצה, כך שלא יהיה לנו dynamic_cast ולא typeid. בקומפיילר של ויז'ואל סטודיו זו ברירת המחדל - כדי לחסוך זמן ומקום בזיכרון. אם רוצים להשתמש באפשרויות האלו צריך לשנות את ההגדרות של הקומפיילר.

מקורות

- מצגות של אופיר פלא ומירי בן-ניסן.
- http://en.cppreference.com/w/cpp/language/reinterpret_cast
- https://stackoverflow.com/q/103512/827927 - למה להשתמש ב static_cast?
- http://en.cppreference.com/w/cpp/language/typeid

סיכם: אראל סגל-הלוי.