זרמי קלט ופלט

נלמד על זרמי קלט ופלט (iostreams) בספריה התקנית של C++. זה גם נושא חשוב ושימושי בפני עצמו, וגם דוגמה מעולה לירושה רגילה וכפולה והעמסת אופרטורים.

היררכיית הזרמים

הספריה התקנית של C++ מגדירה היררכיה שלמה של זרמי קלט ופלט (ראו תרשים במצגת). לצורך הדיון נתמקד במחלקות העיקריות:

- בשורש של עץ הירושה נמצאת המחלקה ios_base והיורשת שלה – ios. המחלקות האלו כוללות משתנים המשותפים לכל הזרמים, כגון: הפורמט ורמת הדיוק של מספרים ממשיים.
- יורשות ממנה המחלקות istream – זרם קלט, ו-ostream – זרם פלט.
- המחלקה iostream מייצגת זרם אשר יכול לשמש גם לקלט וגם לפלט. היא יורשת גם מ istream וגם מ ostream – זו דוגמה לירושה כפולה (אחת הדוגמאות היחידות לירושה כפולה שהיא גם שימושית – בדרך-כלל ירושה כפולה נחשבת לבעייתית ולא שימושית במיוחד).
- מהמחלקה istream יורשות כמה מחלקות שונות של זרמי קלט, במיוחד ifstream (קלט מקובץ) ו istringstream (קלט ממחרוזת). כמו כן, המשתנה cin המשמש לקלט מהמקלדת הוא מסוג זה.
- מהמחלקה ostream יורשות כמה מחלקות שונות של זרמי פלט, במיוחד ofstream (פלט לקובץ) ו ostringstream (פלט למחרוזת). כמו כן, המשתנה cout המשמש לפלט למסך הוא מסוג זה. גם המשתנה cerr משמש לקלט למסך.
  - מה ההבדל? – ההבדל הוא ברמת מערכת ההפעלה: מערכת ההפעלה מקצה לכל תוכנית זרם-קלט אחד (נקרא גם "קלט תקני" או stdin), ושני זרמי-פלט (נקראים גם "פלט תקני" או stdout, ו"פלט-שגיאה תקני" או stderr). כשמריצים תוכנית בלינוקס, ניתן להפנות את הפלט התקני לקובץ אחד ואת פלט-השגיאה התקני לקובץ אחר. למשל, בפקודות הבאות:
    - `./a.out`
    - `./a.out > out.txt`
    - `./a.out 2> err.txt`
    - `./a.out > out.txt 2> out.txt`
- הפקודה הראשונה כותבת גם את הפלט התקני וגם את פלט-השגיאה התקני למסך. הפקודה השניה כותבת את הפלט התקני לקובץ, ואת פלט-השגיאה התקני למסך (שימושי כשיש הרבה פלט שלא רוצים לראות על המסך אלא לשמור בקובץ לעיון מאוחר יותר, אבל עדיין רוצים שהודעות-שגיאה יגיעו למסך באופן מיידי). הפקודה השלישית כותבת את פלט-השגיאה התקני לקובץ ואת הפלט הרגיל למסך, והפקודה הרביעית כותבת את שני סוגי הפלטים לשני קבצים שונים. ראו דוגמה בתיקיה 1.

- מהמחלקה iostream יורשות כמה מחלקות שונות של זרמים המשמשים גם לקלט וגם לפלט, למשל fstream, stringstream. ראינו דוגמאות בשיעור קודם, כשלמדנו על אופרטורים של קלט ופלט.

מניפולטורים

אפשר "לכתוב" לזרמים גם עצמים מיוחדים שנקראים "מניפולטורים" – io manipulators. כשכותבים עצם כזה, למשל, ל-cout, לא מודפס שום דבר למסך, אלא המצב של הזרם cout משתנה. למשל, אפשר "להדפיס" לשם עצם שנקרא boolalpha, האומר שיש להדפיס ערכים מסוג bool באותיות true או false, ולא במספרים 0 או 1. ראו דוגמה בתיקיה 3.

איך זה עובד? פשוט, ע"י העמסת אופרטורים. ראו כאן: https://en.cppreference.com/w/cpp/io/basic_ostream/operator_ltlt

מניפולטור שימושי במיוחד הוא setprecision, המשנה את רמת-הדיוק בכתיבת מספרים ממשיים. כידוע, מספרים ממשיים עלולים להיות אינסופיים, ואנחנו לא רוצים להדפיס מספרים אינסופיים. לכן כשמדפיסים מספרים ממשיים, המחשב מעגל אותם בהתאם ל**רמת הדיוק** שבחרנו. המשמעות של "רמת הדיוק" (precision) היא מספר הספרות המשמעותיות שמודפסות – מספר הספרות בין הספרה השמאלית ביותר שאינה 0 לספרה הימנית ביותר שאינה 0. לדוגמה:

- במספר 7600 יש שתי ספרות משמעותיות – 7, 6.
- גם במספר 0.0076 יש שתי ספרות משמעותיות – 7, 6.
- במספר 7600.0076 יש שמונה ספרות משמעותיות.

ברירת-המחדל של רמת-הדיוק בהדפסה ל-cout היא **6**. אם מדפיסים מספר עם יותר מ-6 ספרות משמעותיות – נראה רק 6. למשל:

- `cout << 1234.5678;`

מדפיס **1234.57**: במספר המקורי יש 8 ספרות משמעותיות, המחשב חותך 2 ומעגל (במקרה זה מעגל למעלה)

- `cout << 12345678.;`

מדפיס **1.23457e+07**: שוב המחשב חותך שתי ספרות ומעגל, אבל במקום לכתוב 12345700, כדי לחסוך באפסים, הוא כותב רק את 6 הספרות המשמעותיות, ומוסיף את האקספוננט e+07 (שמשמעותו: "כפול 10 בחזקת 7").

אפשר לשנות את רמת-הדיוק ע"י הפקודה setprecision שנמצאת בכותרת <iomanip> - ראו בתיקיה 3. למשל, אם נשנה את רמת הדיוק ל-4, נקבל **1235** ו **1.235e+07**; אבל אם נשנה ל-10, נקבל רק 8 ספרות משמעותיות – **1234.5678** ו **12345678**.

אם רמת-דיוק היא גבוהה מאד, אנחנו עלולים לקבל תוצאות לא צפויות. למשל, ברמת דיוק 100 מקבלים:
1234.5678000000000033833202905952930450439453125
מדוע? - כי יש אינסוף מספרים ורק מספר סופי של ייצוגים של מספרים בסיביות. לכן, לרוב המספרים אין ייצוג בסיביות, והמחשב מעגל אותם לייצוג הקרוב ביותר. למשל, למספר 1234.5678 אין ייצוג בסיביות, והמספר הקרוב ביותר שיש לו ייצוג בסיביות (לפחות על המחשב שלי) הוא המספר הנ"ל.
כשרמת הדיוק היא נמוכה, אנחנו לא שמים לב לזה, כי בתצוגה המספר מתעגל ל 1234.5678. אבל כשרמת הדיוק גבוהה, אנחנו רואים את המספר כפי שהוא באמת.

מקורות

- 1) Floating-Point Determinism ,
- 2) What Every Computer Scientist Should Know About Floating-Point Arithmetic ,
- 3) Is floating point math broken? .
- https://en.cppreference.com/w/cpp/io/ios_base/precision

קבצים בינריים

עד עכשיו עבדנו עם קבצי טקסט, שבהם כל בית מייצג (בדרך-כלל) אות הניתנת לקריאה ע"י אדם. יש גם קבציס בינאריים, המשמשים למשל לייצוג תמונות. ישנן דרכים רבות לייצג תמונה בקובץ. ברמה הבסיסית ביותר, כדי לייצג תמונה צריך לייצג את עוצמת האור בפיקסלים של התמונה. נחשוב לדוגמה על תמונה בגווני אפור. לכל פיקסל יש, נניח, 256 גוונים אפשריים של אפור – מ-0 (שחור) ל-255 (לבן). יכולנו לייצג את התמונה כקובץ טקסט ובו רשימה של מספרים מופרדים בפסיקים, אבל זה מאד בזבזני. מקובל יותר לייצג תמונה כקובץ בינרי ובו כל בית מייצג פיקסל אחד, באופן דחוס.

אם רוצים לייצג תמונה צבעונית, אפשר לייצג כל פיקסל בעזרת שלושה בתים, שכל אחד מהם מייצג את עוצמת האור בערוץ-צבע אחר (למשל: אדום, ירוק, כחול). ישנן דרכים רבות לעשות זאת, והן מעבר להיקף של קורס זה. אנחנו לומדים על הנושא רק כדוגמה לשימוש בקובץ בינארי. בתיקיה 4 ניתן למצוא דוגמה לקובץ-תמונה בפורמט פשוט ביותר – פורמט ppm. שימו לב איך התמונה נוצרת באופן אוטומטי ע"י נוסחה בתוך מערך בזיכרון, ואיך היא נשמרת לקובץ בפעולה אחת write.

מקורות

- מצגות של אופיר פלא ומירי בן-ניסן.

סיכם: אראל סגל-הלוי.