# זרמי קלט ופלט

## Stream

קלט/פלט של C ++ מתרחש בזרמים (Streams) של מידע, שהם רצפי בתים (bytes), כל בית בגודל 8-bit. פעולות קלט/פלט מתבצעות דרך אובייקטים המייצגים "זרימה" של מידע מסוגים שונים. כיוון שקשה לדמיין אובייקט שמייצג זרימה של מידע, מומלץ לדמיין אותו כאובייקט המייצג צינור שדרכו המידע עובר. באמצעות האופרטורים והשיטות המוגדרים על אובייקטים אלו נוכל להזרים מידע ממקום למקום. אם בתים זורמים מהתקן כמו מקלדת, כונן דיסק, או חיבור רשת וכו' לזיכרון הראשי, זה נקרא פעולת קלט. ואם בתים זורמים מהזיכרון הראשי להתקן כמו מסך, מדפסת, וכו', זה נקרא פעולת פלט.

## היררכיית זרימה

הספרייה התקנית של ++C מגדירה היררכיה שלמה של זרמי קלט ופלט . 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 המייצגת זרם קלט, ו-ostream המייצגת זרם פלט. ארבעת האובייקטים המרכזיים המייצגים זרימה תקנית מסוג מחלקות אלו: cin, cout, cerr, clog.
  + המחלקה iostream יורשת גם מ-istream וגם מ-ostream, ולכן מייצגת זרם אשר יכול לשמש גם לקלט וגם לפלט. כדי להשתמש בפקודות קלט ופלט יש לכלול ספרייה זו <iostream>.
  + משלושת המחלקות לעיל יורשות ifstream ו-ofstream, המייצגות קלט ופלט לקבצים. המחלקה fstream יורשת מ-iostream ומכילה גם קלט וגם פלט לקבצים. נלמד עליהם בהמשך.
  + משלושת המחלקות לעיל יורשות istringstream ו-ostringstream, המייצגות קלט ממחרוזת ופלט למחרוזת. המחלקה stringstream יורשת מ-iostream ומכילה גם קלט וגם פלט למחרוזות.
  + המחלקה iomanip מכילה אובייקטים ושירותים שמועילים לביצוע קלט/פלט מובנים עם זרימות מיוחדות הנקראות מניפולטורים, כגון setw ו-setprecision.

## זרמים תקניים

מערכת ההפעלה מקצה לכל תוכנית זרם קלט אחד הנקרא "קלט תקני" (stdin), ושני זרמי פלט הנקראים "פלט תקני" (stdout), ו"פלט שגיאה תקני" stderr)).

### קלט תקני - cin

הקלט התקני מחובר למכשיר הקלט התקני שהוא בדרך כלל המקלדת. כאמור לעיל, האובייקט המייצג זרימת קלט זו הוא **cin**. כדי לחלץ את הערך מתוך אובייקט זה יש להשתמש באופרטור החילוץ '>>'. מוגדרת העמסה על אופרטור זה כך שידע לקלוט כל קלט מסוג בסיסי. אנו יכולים גם להרחיב אופרטור זה כדי לקרוא מחלקות שאנחנו בונים (ראה קובץ פונקציות ואופרטורים). ניתן להשתמש באופרטור החילוץ יותר מפעם אחת בהצהרה יחידה, כדי לקלוט מספר ערכים.

std::cin >> name >> age;

### פלט תקני - cout

הפלט התקני מחובר למכשיר הפלט התקני שהוא בדרך כלל מסך התצוגה. האובייקט המייצג זרימת פלט זו הוא **cout**. כדי להכניס ערך לתוך אובייקט זה יש להשתמש באופרטור ההכנסה '<<'. מוגדרת העמסה על אופרטור זה כך שידע לפלוט כל פלט מסוג בסיסי. אנו יכולים גם להרחיב אופרטור זה כדי להדפיס לפי הגדרתנו מחלקות שאנחנו בונים (ראה קובץ פונקציות ואופרטורים). ניתן להשתמש באופרטור ההכנסה יותר מפעם אחת בהצהרה יחידה, כדי לפלוט מספר ערכים. כדי לרשת שורה נפלוט endl.

std::cout << "Hello world!" << endl;

### פלט שגיאה תקני - cerr, clog

מחובר למכשיר פלט השגיאה התקני שהוא גם כן בדרך כלל מסך התצוגה. האובייקטים המייצגים זרימת פלט זו הם **cerr** ו-**clog**. ההבדל ביניהם הוא ש-cerr לא משתמש בבאפר (buffer), ולכן כל פעם שמכניסים אליו ערך הוא מועבר ישירות למכשיר הפלט. ואילו clog כן משתמש בבאפר, ולכן כל מה שמכניסים אליו ממתין עד שהבאפר יתמלא או עד שירוקנו אותו (flush). ההכנסה לתוך אובייקטים אלו זהה ל-cout, כלומר יש להשתמש באופרטור ההכנסה '<<'.

std::cerr << "Error message" << endl;

## הדפסה לקובץ בלינוקס

כשמריצים תוכנית בלינוקס אנו יכולים 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 השנייה 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 מידי). הפקודה השלישית כותבת את פלט השגיאה התקני לקובץ ואת הפלט הרגיל למסך. והפקודה הרביעית כותבת את שני סוגי הפלטים לשני קבצים שונים.

## מניפולטורים

ישנם כל מיני אובייקטים מיוחדים שנקראים "מניפולטורים" (io manipulators), המוגדרים במחלקה iomanip. שאם מכניסים אותם לאובייקט המייצג זרימה, הוא לא מזרים כלום אך המצב שלו משתנה. אחד המניפולטורים הנפוצים הוא setprecision.

### רמת הדיוק - setprecision

כשמדפיסים מספרים ממשיים, המחשב מעגל אותם בהתאם לרמת הדיוק מסוימת. רמת הדיוק (precision) היא מספר הספרות המשמעותיות, שהן מספר הספרות בין הספרה השמאלית ביותר שאינה 0 לספרה הימנית ביותר שאינה 0. לדוגמה, במספר 7600 יש שתי ספרות משמעותיות, ובמספר 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 מתוכם. למשל: cout << 1234.5678 ידפיס רק **1234.57** המחשב חותך 2 ומעגל. ועבור cout << 12345678 מדפיס 1.23457e+07. 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 של cout על ידי המניפולטור setprecision. למשל, אם נשנה את רמת הדיוק ל-4, נקבל עבור הדוגמה האחרונה: 1.235e+07. אם רמת הדיוק היא גבוהה מאד, אנחנו עלולים לקבל תוצאות לא צפויות. זאת מפני שיש אינסוף מספרים ורק מספר סופי של ייצוגים של מספרים בסיביות. לכן, לרוב המספרים אין ייצוג בסיביות, והמחשב מעגל אותם לייצוג הקרוב ביותר.

cout << setprecision(8) << 12345678 << endl;

### הדפסת ערכים בוליאניים - boolalpha

כשמדפיסים ערכים בוליאניים, במצב הרגיל יודפס 0 עבור false ו-1 עבור true. אם נרצה שבמקום זאת יודפס true"" ו-"false", נשתמש במניפולטור boolalpha. cout << boolalpha << bool statement

# קלט ופלט - קבצים

# קבצים בינאריים

## יצירת תמונות

קבצים בינריים יכולים לשמש לייצוג תמונות. לא נרצה לייצג תמונה בקובץ טקסט, מפני שיכול להשתנות בין מערכות הפעלה, ובכך לשנות את התמונה. סיבה נוספת מפני ששימוש בקובץ טקסט יהיה מאוד בזבזני וכל תמונה תצרוך המון זיכרון. לעומת זאת, קבצים בינאריים נשארים זהים בכל מערכת הפעלה וצורכים פחות זיכרון.

ישנן דרכים רבות לייצג תמונה בקובץ. ברמה הבסיסית ביותר, תמונה בשחור לבן כל בית יהיה פיקסל אחר בתמונה, כך שלכל פיקסל יש 256 גוונים אפשריים של אפור מ-0 (שחור) ל-255 (לבן). בתמונה צבעונית אפשר לייצג כל פיקסל בעזרת שלושה בתים, שכל אחד מהם מייצג את עוצמת האור בערוץ צבע אחר (למשל: אדום, ירוק, כחול).

כדי ליצור קובץ בינארי כזה שמכיל בתוכו תמונה, נבנה תוכנית ונגדיר בה מבנה RGB עם שדות אדום ירוק וכחול. שדות אלו יהיו מסוג uint8\_t שהוא טיפוס המכיל בית אחד ומשמש לכתיבה בינארית. נבנה מערך image שבו כל תא הוא RGB. ניתן לכל תא כזה ערכים. כעת באמצעות פקודה אחת write, המקבלת מערך של בתים ואת גודל המערך נוכל להדפיס את כל המערך לקובץ בינארי. בקובץ וידאו מה שקורה הוא שמפעילים את write מספר פעמים בשנייה, בדרך כלל 60. דוגמאות בתיקייה של שיעור 7 תיקייה 5.

### בעיות לוגיות אפשריות

* אם המערך של image מכיל מצביעים ל-RGB בערימה נקבל תמונה לא כמו שרצינו. הסיבה לכך היא שבערימה המבנים לא מסודרים באופן רציף ולכן גם הכתיבה לקובץ לא רציפה.
* אם ב-RGB תהיה שיטה וירטואלית, אזי כל אובייקט מסוג זה יכיל שדה נסתר של מצביע לטבלת השיטות הווירטואליות. ולכן כאשר נכתוב אובייקטים אלו לקובץ שוב נקבל אי רציפות בזיכרון. לכן מבנה RGB כזה חייב להיות ללא שיטות וירטואליות. אם בכל זאת נרצה להשאיר את השיטות הווירטואליות נהיה חייבים לכתוב את הפיקסלים בלולאה כפולה אחד אחר השני.