מבוא לתכנות תחרותי

עקרונות בסיסיים בשפות התכנות C++, טיפוסים פרימיטיביים, מחרוזות,
מספרים גדולים, תחביר השפה, קלט ופלט, רפרנסים וקבועים.

קלט ופלט

קלט ופלט סטנדרטי

- ישנם דרכים שונים שבהם תוכנית יכולה לקבל קלט ולפלוט פלט
- הדרך הסטנדרטית והפשוטה היא בעזרת הזרמים הדיפולטיבים:
  1. Standard Input, או בקיצור *stdin*
  2. Standard Output, או בקיצור *stdout*
  3. Standard Error, או בקיצור *stderr*
- הספרייה הסטנדרטית של C++ מממשת את האובייקטים cin,cout,cerr שבעזרתם ניתן לקרוא ולכתוב לזרמים הדיפולטיבים
- נשתמש בendl בכדי להדפיס את תו ירידת השורה '\n' ולרוקן את כל הפלט למסך

שלום עולם!

```
#include <iostream>

int main() {
    std::cout << "Hello world!" << std::endl;
}
```

דגשים לתכנות תחרותי

- בתכנות תחרותי בדרך כלל נשתמש בהרבה אובייקטים מהספרייה הסטנדרטית המוגדרים בהרבה קבצי header שונים
- במערכת ההפעלה לינוקס קיים קובץ ה־header <bits/stdc++.h> המכיל בתוכו את כל ה־include־ים לקבצי ה־header הסטנדרטיים
- כל האובייקטים הסטנדרטיים נמצאים בתוך namespace ששמו std
- לשם הנוחות, בתכנות תחרותי נכתוב בתחילת הקובץ using namespace std בכדי לא לכתוב std בכל מקום שבו אנו משתמשים באובייקט מהספרייה הסטנדרטית

```
#include <bits/stdc++.h>
using namespace std;

int main() {
    cout << "Hello world!" << endl;
}
```

קצת בשר

```
#include <bits/stdc++.h>
using namespace std;

int main() {
    int n;
    cin >> n;
    int s = 0;
    while(n--) {
        int a;
        cin >> a;
        s += a;
    }
    cout << s << endl;
}
```

- התוכנית קולטת מספר n ואחריו רשימת מספרים בגודל n, ומדפיסה את סכומה.
- נשים לב שגם בשימוש ב־cin הזרם נמצא "בצד שמאל" ורק כיוון החצים משתנה ל־<< בהתאם. השורה n << cin לא תתקמפל!
- הביטוי n-- מקטין את הערך של המשתנה n באחד ומחזיר את הערך **הקודם** של המשתנה.
- הלולאה תיעצר כאשר n-- יחזיר 0, כלומר בדיוק אחרי n איטרציות.
- באופן דומה, נוכל לבצע פעולות כמו n++, n--, ++n.

מספרים

שלמים

int32_t

long long

int64_t

int

__int128_t

שלמים

- הסטנדרט של C/C++ לא מגדיר מהו הגודל של משתנה `int`. עם זאת, ברוב המחשבים לשימוש כללי (general-purpose) גודלו של int הוא 32 סיביות (4 בתים).
- הטווח של מספר (מסומן) בגודל 32 סיביות הוא $[-2^{31}, 2^{31}-1]$. זאת כאשר $2^{30} \approx 10^9$.
- בתכנות תחרותי, פעמים רבות נרצה דיוק גדול יותר. לשם כך נשתמש בטיפוס `int64_t` (או `long long`). טיפוס זה מורכב מ־64 סיביות.
- הטווח של מספר (מסומן) בגודל 64 סיביות הוא $[-2^{63}, 2^{63}-1]$. הרבה יותר ממשתנה `int` רגיל!
- לפעמים נשתמש בשורה `#define int int64_t` בתחילת הקובץ (אחרי ה־`include`־ים) בכדי להפוך את ה־`int`־ים בתוכנית ל־`int64_t`.
- יש גם את הטיפוס `__int128_t`, אבל הוא לא נתמך על ידי מעבדים בחומרה, באגי ואיטי מאוד. בדרך כלל אפשר להימנע מלהשתמש בו. בנוסף אי אפשר לקלוט ולהדפיס אותו בקלות.
- אם ננסה לבצע פעולה ולשמור אותה במשתנה שלא מכיל מספיק מקום לתוצאה, נקבל מספר שגוי. לתופעה (באג) קוראים overflow.

כל אחד ומה שנוח לו

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;

int main() {
    int x; // 32bit
    ll y;  // 64bit
}
```

```cpp
#include <bits/stdc++.h>
using namespace std;
#define int int64_t

int32_t main() {
    int x, y;  // 64bit
}
```

קלט ופלט INT128_T

```
#include <bits/stdc++.h>

using namespace std;

typedef __int128_t int128_t;

ostream& operator<<(ostream& out, int128_t n) {

    if (n < 0)
        return out << '-' << -n;

    if (n < 10)
        return out << char(n % 10) + '0';

    return out << n / 10 << char(n % 10) + '0';

}
```

```
istream& operator>>(istream& in, int128_t& n) {

    char c;

    do in >> c;

    while (!isdigit(c) && c != '-');

    bool neg = c == '-';

    if (neg) in >> c;

    n = 0;

    for (; isdigit(c); in >> c)

        n *= 10, n += c - '0';

    if (neg) n *= -1;

    return in;

}
```

ממשיים

- בניגוד למספרים שלמים, מספרים ממשיים קשה לתחזק במחשב בצורה טובה.
- הדרך הקלאסית לייצוג מספרים ממשיים הם הטיפוסים `float` (32 סיביות) ו־`double` (64 סיביות).
- בשניהם, יהיו בעיות דיוק, במיוחד בהשוואה בין שני מספרים לאחר הרבה חישובים.
- בתכנות תחרותי, ננסה להימנע ככל הניתן מהטיפוסים `float` ואפילו `double`.
- נעדיף לא להשתמש בפעולות סטנדרטיות כמו pow או `log2` העובדות עם ממשיים, אלא נממש אותן בעצמנו בעבור שלמים.

בעיות דיוק בממשיים

```
#include <bits/stdc++.h>
using namespace std;

int main() {
    cout << fixed << setprecision(20);

    double a = 0.1, b = 0.2;

    cout << "a   = " << a << endl;
    cout << "b   = " << b << endl;
    cout << "a+b = " << a + b << endl;

    if (a + b == 0.3)
        cout << "a+b == 0.3" << endl;
    else
        cout << "a+b != 0.3" << endl;
}
```

בעיות דיוק בממשיים

```
#include <bits/stdc++.h>
using namespace std;

const double eps = 1e-8;  // 0.00000001

bool epseq(double a, double b) {
    return abs(a - b) <= eps;
}

int main() {
    double a = 0.1, b = 0.2;
    if (epseq(a + b, 0.3))
        cout << "a+b == 0.3" << endl;
    else
        cout << "a+b != 0.3" << endl;
}
```

פעולות מתמטיות לשלמים

```
int ilog2(int a) {
    if (a == 1) return 0;
    return ilog2(a / 2) + 1;
}
```

```
int ipow(int a, int b) {
    if (b == 0) return 1;
    return a * ipow(a, b - 1);
}
```

מימוש לדוגמה בלבד - יש מימושים יעילים יותר.
תרגיל: האם ישנו מימוש טוב יותר (סיבוכיות) לפעולת החזקה?
נסו לממש את פעולת ה־log עם הבסיס כמשתנה, ועם עיגול כלפי מעלה.

מחרוזות ותווים

טיפוס התו

```
int main() {
    char c = 'a';
    while (c <= 'z')
        cout << c++;
}
```

- סטנדרט ה־ASCII מאפשר לנו לייצג תווים בסיסיים כמו אותיות לטיניות, ספרות וסימני פיסוק כל אחד בעזרת קידוד יחיד באורך 8 סיביות – בית אחד.
- בית אחד הוא יחידת הזיכרון הקטנה ביותר שמחשב מודרני מספק לנו. ניתן להקצות בית אחד בעזרת הטיפוס `char`.
- בפועל, משתנה מהטיפוס `char` מכיל מספר בתווך $[0,255]$.
- הקידוד של אותיות לטיניות עוקבות ושל ספרות עוקבות עוקב גם הוא.

תו הוא בעצם מספר

```
int main() {
    char c = 'a';
    while (c <= 'z')
        cout << c++;
}
```

```
int main() {
    char c = 97;
    while (c <= 122)
        cout << c++;
}
```

אובייקט המחרוזת

```
int main() {
    string s;
    cin >> s;

    for (char c : s)
        cout << "got " << c << endl;

    for (int i = 0; i < s.size(); i++)
        cout << "got " << s[i] << endl;
}
```

- הספרייה הסטנדרטית של C++ מממשת אובייקט המייצג רצף של תווים ושמו `std::string`.
- קליטת מחרוזת תתבצע עד תו מרווח (whitespace) כלשהוא, כמו תו הרווח או תו ירידת השורה.
- ניתן לגשת לתו ה־i במחרוזת s ע"י `s[i]`.
- ניתן לעבור על כל התווים במחרוזת ע"י הלולאה `for (char c : s)`
- למחלקה `string` מטודות שימושיות כמו `size`, `reverse` וכו'.

משתנים ומערכים גלובליים

משתנים גלובליים

```
#include <bits/stdc++.h>
using namespace std;

void print(int x) {
    cout << x << endl;
}

int main() {
    int n;
    cin >> n;
    print(n);
}
```

```
#include <bits/stdc++.h>
using namespace std;

int n;  // outside main!

void printn() {
    cout << n << endl;
}

int main() {
    cin >> n;
    printn();
}
```

מערכים גלובליים

```
#include <bits/stdc++.h>
using namespace std;

const int maxn = 2e5;  // 200,000
int array[maxn];

int main() {
    int n;
    cin >> n;
    for(int i=0; i<n; i++)
        cin >> array[i];
}
```

- בשאלות הדורשות שימוש במבנה נתונים כמו מערך, תופיע הגבלה על גודלו של הקלט.
- נוכל תמיד ליצור מערך בגודל המקסימלי הנתון בשאלה.
- נוכל להיעזר במשתנה קבוע (const) כמו maxn בכדי לשמור את גודל הקלט המקסימלי.
- ניצור מערך גלובלי בדיוק כמו משתנה גלובלי, אך עם הסימון [] ובתוכו גודל המערך.
- גישה לאיבר ה־i במערך תתבצע בעזרת array[i] כאשר array הוא שם המערך.

מערכים דו ממדיים

```
#include <bits/stdc++.h>
using namespace std;
#define fori(n, i) for (int i = 0; i < n; i++)

const int maxn = 1000;
char array[maxn][maxn];

int main() {
    int n, m;
    cin >> n >> m;
    fori(n, i) fori(m, j)
        cin >> array[i][j];
}
```

- אותם העקרונות תקפים גם במערכים עם שני ממדים או יותר!
- נשים לב כי ניתן לשנות את טיפוס הערכים במערך והוא אינו חייב להיות int (במקרה הזה הוא char).
- כדי לחסוך את לולאות ה־for הארוכות ניתן להשתמש ב־define ולהגדיר "סוג לולאה" חדש בעצמנו.