@blackcurrant 2016年12月09日に投稿

eeic （東京大学工学部電気電子・電子情報工学科）その2　9日目

競プロ初心者が書く「標準入出力からはじめる競プロ入門」

C++　競技プログラミング

⚠ この記事は最終更新日から1年以上が経過しています。

この記事は、eeic Advent Calendar 2016 その2 の9日目の記事として書かれたものです。

競プロをまだやったことがない人や、競プロを始めて間もない人で、C/C++の入出力がよく分からないという人を対象とした記事です。執筆者が入出力で躓いたことがあるので、克服も兼ねて注意点を集積しました。したがって経験者の方(レッドコーダーの方、C++の言語仕様書を読破している方、あるいは「私はHaskellerだ」「私はJavaのプロだ」という方など)は得られるものはありません。
タイトルの通り、執筆者は競プロおよびC++の初心者ですので、不正確な情報を書いている場合があります。見つけた場合はコメント欄で適当につついてくださるとありがたいです。

競プロ(競技プログラミング)とは

実際の問題を見てもらうのが手っ取り早いので、これ(はじめてのあっとこーだー)を見てください。

各問題には、**制限時間**、**メモリ制限**、**問題文**、**入力の形式と条件**、**出力の形式**が示されています。端的には、問題文に従い、与えられた入力に対して制限時間･メモリ制限の中で何らかの計算を行った結果を出力するプログラムを提出することが目的です。今回の例では与えられた3つの数を足すという計算を行う必要があります。

提出したプログラムの正当性は、**オンラインジャッジ**と呼ばれる採点システムにより、複数のテストケースで自動的に評価されます。全てのテストケースをパスすると**AC**(Accepted)となります。制限時間をオーバーしたり、間違った答えを出力すると不正解となります。詳細はAtCoderの説明ページやAOJの説明ページを見てください。

競プロの始め方

まず、TopCoder部をブックマークします。コンテストの情報がまとまっています。
次に、主なコンテストのウェブサイトをブックマークします。とりあえず日本語のAtCoder, AOJ, yukicoderあたりで十分だと思います。この3つでは、オンラインジャッジが常に稼働していて練習に最適です。ありがたく使いましょう。他にはPOJ, Codeforces, HackerRank, CodeFightsなどがあります。TopCoderは始めるのが大変であることで有名です。
AtCoderをやっていくなら、AtCoder Problemsも押さえておきましょう。これはAtCoderの非公式過去問集です。ユーザーIDを入力すると、ユーザーページにてAtCoderでの統計情報を見られます。

競プロではC++を使うことが多いです。実行速度が速く、標準ライブラリも豊富だからです。また、AtCoderなど多言語に対応しているサイトでも、全ての言語でACできることは保証していませんが、C++でACできることは保証していることが多いです。なのでC++のリファレンスをブックマークしておきましょう。ここや、日本語が好きな方はここがいいでしょう。

次に実行環境を整えます。
Macをお使いの方は、デフォルトで `clang` というコンパイラが入っているので大丈夫です。
Windowsをお使いの方は、Cygwinをインストールするのが手っ取り早いと思います。セ

ットアップ中のSelect Packagesにて、 `Devel` カテゴリの `gcc-core` と `gcc-g++` を選択すればOKです。
(Windowsよく分からないので詳しい方ぜひ教えてください)
どちらの場合でも、最低限のターミナルの操作は適宜ググってください。
もしくは、ソースを投げるとコンパイルしてくれるウェブサービスを使うという手もあります。
あとは、お気に入りのエディタを用意するだけです。お気に入りがない場合、Atomとか万人向けでいいんじゃないでしょうか。

計算量

制限時間、メモリ制限に関係するお話です。

制限時間は数秒であることがほとんどです。プログラムはこの制限時間内で出力を返す必要があります。2016年現在、一般的なCPUでは、目安としてループ回数が10^7程度までででないと、この制限時間を超えてしまう可能性が高くなります。実行時間の大きさを時間計算量と呼びます。計算量については、こちらの記事が詳しいです。

メモリ制限について。メモリ消費量は変数を多く使うほど増えます。メモリ消費量の大きさを空間計算量と呼びます。時間計算量より多く変数を宣言することはあまりないので、時間計算量ほど気にする必要はありませんが、大量の変数を宣言したりコピーするときには少し気をつけましょう。

標準入出力

メイントピックです。
競プロではTopCoderのような特殊な形式を除いて、ほぼ必ず標準入力から入力を受け取り、結果を標準出力に出力します。int型変数の入出力は、C言語では次のように書きます。

sample_stdio.c

```
#include <stdio.h>
int main() {
  int a;
  scanf("%d", &a);
  printf("%d\n", a);
  return 0;
}
```

C++はC言語を包含しているので、当然この書き方が可能ですが、別の書き方も可能です。

sample_iostream.cc

```
#include <iostream>
using namespace std;
int main() {
  int a;
  cin >> a;
  cout << a << endl;
  return 0;
}
```

見た感じ、C言語の書き方はやや冗長で、C++の書き方のほうが簡単です。今更C言語の書き方は必要なのでしょうか？
答えはYESです。 `iostream` を使った入出力はとにかく遅いのです。ただし、この遅さを軽減する手段があるようです。
"iostream 遅い"などと検索すると、 `cin.tie(0);` `ios::sync_with_stdio(false);` の2つの記述を加えると速くなる、という情報が見つかります。これはそれぞれ、 `cin` と `cout` の同期を切るもの、 `iostream` と `stdio` の同期を切るものです。
以上の情報が正しいかどうかを確認するため、簡単な実験を行ってみました。

環境

- PC : MacBook Pro (Retina, 13-inch, Early 2015)
- OS : Mac OS X El Capitan 10.11.6
- コンパイラ : Apple LLVM version 8.0.0 (clang-800.0.42.1)
- CPU : Core i5-5257U (2.7GHz)
- メモリ : 16GB

まず10^6個のランダムな符号付き32ビット整数のデータを用意します。用意したデータを標準入力から受け取り、そのまま出力するプログラムの実行時間が以下です。
実行時間は何回か計測した最小値を記載しています。
最適化オプションは付けていません。

stdio	iostream	iostream(高速化)
0.420(s)	4.956(s)	4.657(s)

マシンの状態にもよりますが、 `iosteram` では読み込んだだけで2secを超えてしまうようです。私の環境では `iostream` 高速化もあまり効果がありませんでした。
この結果から、データサイズが10^5以下なら `iostream` でもOKですが、問題によっては使ってはいけないということが分かります。また、データサイズがそれほど大きくない場合であっても、入出力にかかる時間が少ないほど有利であるのは言うまでもありません。したがって競プロでは `stdio` の入出力を主に使うべきでしょう。

これらの入出力関数の挙動を理解し制御するためには、どう実装されていて、メモリの中がどうなっているのかを把握するのが一番ですが、それは一筋縄ではいかないと思います。私は分かりません。一筋縄でいってしまう方は、ここでブラウザバックしてください。
以下では競プロの本題であるアルゴリズムに集中するために最低限必要な知識だけをまとめていきます。

入出力には `scanf` と `printf` を使えばいいのは分かりました。競プロをやっていく上で読み込みたくなるのは、**整数(int)**、**小数(double)**、**文字(char)**、**文字列(std::string)**の4つくらいでしょう。
このうち整数、小数についてはスペースで区切られていようが、改行で区切られていようが、

```
#include <cstdio>
for (int i = 0; i < n; ++i) {
  scanf("%d", &a[i]);
}
```

などと適当に書いておけばスペースや改行を無視していい感じに読み込んでくれます。文字、文字列については問題が生じます。

文字(char)を読み込む場合

`scanf("%c", &c);` などと書くと、文字を受け取ることができますが、これだとスペースや改行文字やタブ文字を無視せずに読み込んでしまいます。これを防ぐためには、 `scanf(" %c", &c);` と書くという方法があります。 `%c` の手前にスペースを付けることで、スペース(改行文字とタブ文字を含む)を読み飛ばして別の文字(アルファベットなど)を受け取ることができます。また、1文字を明示的に読み飛ばしたいときは、 `scanf("%c%*c", &c);` と書くことができます。この書き方は、受け取りたい文字の直後に改行文字が存在するので、確実に改行文字を読み飛ばしたい、といった状況で有効です。

stringを使う場合

ではC++の `string` を使う場合はどうでしょう。
`scanf` , `printf` はC言語の関数なので、C++の `string` を扱うことはできません。この場合、入出力は次のように書く必要があります。

```
#include <iostream>
#include <string>
using namespace std;
string s;
cin >> s;
cout << s << endl;
```

`iostream` の入出力は遅いということは知っていますが、 `string` の便利さに負けて仕方なく使うわけです。
実は `endl` は遅いので、以下の(1)ではなく(2)のように書きましょう。(数回の出力ならどっちでもいいですが、習慣づけということで)

```
cout << endl; // (1)
cout << "\n"; // (2)
```

参考までに、(1)と(2)をそれぞれ10^6回繰り返したときの実行時間が以下です。環境は上に書いたのと同じです。

(1) endl	(2) "\n"
1.473(s)	0.083(s)

ここまでで大抵の問題には対応できると思います。でももしかしたら、スペースが含まれる1行を丸ごと `string` に入れたいということもあるかもしれません。そんなときはこちら。

```
string s;
getline(cin, s);
```

`getline` 関数を連続で使うことで、改行を無視しながら各行を `string` に格納することができます。

最後に、 `cin` または `getline` と `scanf` を同時に使った場合に起こる問題について。

```
char c;
string s;

scanf("%c", &c); // 直後の改行は無視される
cin >> s;

cin >> s;        // 直後の改行が無視されない
scanf("%c", &c);

scanf("%c", &c); // 直後の改行が無視されない
getline(cin, s);

getline(cin, s); // 直後の改行は無視される
scanf("%c", &c);
```

上記の通りです。 `cin` は最初に改行が来るとそれを飛ばして読み、スペースや改行以外の文字を1文字以上読み込んだ後、終端に来た改行は未処理のままにしておく、という挙動で、 `getline` は最初に改行が来るとその改行が全てであると解釈し、終端の改行を含めた処理を行っている、とみなすことができます。これはもしかしたら環境によって異なるかもしれませんので、気になった方は自分の環境で試してみてください。

参照

リンクまとめ

- はじめてのあっとこーだー
  - 様々な言語の入出力サンプル
- 競技プログラミングの用語集
  - オンラインジャッジのステータスについては一度見ておくとよいでしょう
- Cygwinのインストールと使い方
- [初心者向け] プログラムの計算量を求める方法
- scanf - Wikipedia
  - scanfの問題点と回避方法
- プログラミングのテクニック/入力の処理(C,C++)

@blackcurrant

フォロー