Tutorial

Jak mają wyglądać Zawody?

Zawody wzorowane są na ogólnoświatowym konkursie International Collegiate Programming Contest - organizowanych przez ACM. Zadaniem wszystkich drużyn będzie napisanie programów, rozwiązujących problemy natury algorytmicznej opisane w treściach zadań, dostarczonych w kopertach tuż przed rozpoczęciem zawodów. Po oficjalnym rozpoczęciu odliczania czasu można przystąpić do rozwiązywania zadań. Zawody trwają 5 godzin i tylko w tym czasie można wysyłać rozwiązania problemów. Rozwiązania będą sprawdzane i oceniane na bieżąco. Pliki z kodem waszych programów można wysyłać poprzez specjalny system zwany „Sprawdzarką". Jest to system automatyczny - Wasze programy nie będą, więc testowane przez ludzi. Bardzo ważnym elementem zawodów jest fakt, że zawodnicy startują w trzyosobowych drużynach i mają do dyspozycji tylko jeden komputer. To jak zawodnicy w ramach drużyny podzielą się obowiązkami zależy wyłącznie od nich samych. Choć dostęp do Internetu na waszym stanowisku będzie zablokowany, pamiętajcie o tym, że możecie mieć ze sobą wszelkiego rodzaju materiały w formie papierowej (książki, kody programów, dokumentacje, zeszyty itd.) – jedynie materiały w formie elektronicznej są zabronione.

„Sprawdzarka" będzie testować nadsyłane programy za pomocą zestawu testującego zawierającego informacje wejściowe i wyjściowe jakie powinien wygenerować oceniany program. Jeżeli kod waszego programu zostanie prawidłowo skompilowany, po czym nie nastąpi w trakcie jego realizacji błąd wykonania, nie będzie działał zbyt długo lub nie użyje zbyt dużo pamięci i na końcu da poprawną odpowiedź zostanie zaakceptowany i „Sprawdzarka" wyświetli komunikat **„TAK"**. Jeżeli wasze rozwiązanie zostanie odrzucone „Sprawdzarka" wyświetli jeden z następujących komunikatów:

„NIE - przekroczenie czasu"
„NIE - przekroczenie pamięci"
„NIE - zła odpowiedź"
„NIE - błąd kompilacji"

Wszystkie te komunikaty dają drużynie tak zwaną „bombę" - czyli karę 20 minut do czasu rozwiązania danego zadania. O tym jednak (jak i bardziej szczegółowo o tych komunikatach) później. Każda drużyna może próbować poprawić błędne rozwiązania i ponownie wysyłać plik z kodem do sprawdzenia. Nie ma ograniczenia, które zakazywało by wysyłać zawodnikom dane rozwiązanie więcej niż pewną ilość razy w przypadku niepowodzeń.

„Sprawdzarka" – czy tylko sprawdza?

„Sprawdzarka" mimo swojej jednoznacznej nazwy nie tylko umożliwia wysyłanie zadań w celu ich sprawdzenia. Każda drużyna może sprawdzić, które z ich zadań zostały zaakceptowane. Ponad to „Sprawdzarka" umożliwia zadawanie pytań (ogólnych i dotyczących konkretnych zadań) sędziom. Opcja ta ma służyć wyjaśnieniu niejasności z treści zadań (lub samych zawodów), bądź zgłaszaniu ewentualnych błędów. Odpowiedź zostanie udzielona tak szybko jak będzie to możliwe. Jeżeli sędziowie uznają pytanie za ważne to odpowiedź na nie będzie widoczna dla wszystkich drużyn. Pytania dotyczące stanowiska na którym pracujecie możecie zadawać również personelowi pomocniczemu, który będzie obecny w każdej sali (to jest jedyna możliwość kontaktu z osobą nie będącą w Waszej drużynie).

Jest możliwe wysłanie za pomocą „Sprawdzarki" pliku z kodem w celu jego wydrukowania. Wydruk będzie wam dostarczony tak szybko jak będzie to możliwe. Dzięki temu możecie np. szukać błędów w programie, podczas gdy wasz kolega będzie na komputerze tworzył rozwiązanie do innego zadania.

Z do tej pory omawianymi aspektami zapoznacie się bliżej podczas godzinnej sesji próbnej przed zawodami właściwymi.

Przykładowe zadania

Charakter zadań z jakimi się spotkacie będzie podobny do zadań z takich konkursów jak: Central European Programming Contest, Akademickie Mistrzostwa Polski w Programowaniu Zespołowym, Potyczki Algorytmiczne, Internetowe Mistrzostwa Polski w Programowaniu. Treść tych zadań (wraz z opisem wejścia/wyjścia i przykładem) będzie zajmowała około 1-2 strony A4. Zadania będą problemami natury algorytmicznej i matematycznej. Na stronach internetowych wymienionych konkursów (również na naszej) są podane przykładowe zadania np. z poprzednich edycji zawodów. Materiału dla treningu więc nikomu nie powinno zabraknąć.

Które zadanie rozwiązywać wcześniej, a które później i jak je rozwiązywać?

Oczywiście kolejność rozwiązywania i wysyłania zadań jest całkowicie zależna od was. Warto rozwiązywać najpierw zadania łatwiejsze, a dopiero później te trudniejsze. Choćby dlatego, że jeżeli jakaś inna drużyna zrobi tą samą ilość zadań co Wy, to o miejscu w rankingu będzie decydował zsumowany czas wszystkich Waszych rozwiązanych zadań (powiększone ewentualnie o kary za błędne rozwiązania). Jak się oblicza taki czas? Jeżeli drużyna rozwiąże jakieś zadanie poprawnie to bierze się czas nadesłania danego rozwiązania (liczony od początku zawodów) i dodaje się do ogólnego czasu tej drużyny. Jeżeli wcześniej drużyna wysyłała błędne rozwiązania będzie on powiększony o 20 minut za każde błędne rozwiązanie. Czasy karne za zadanie, którego nie udało się wam rozwiązać nie będą liczone. Np.: Drużyna rozwiązała jedno zadanie po 46 minutach. Po czym dwa razy wysłała złe rozwiązanie zadania drugiego, aż w końcu po 90 minutach (licząc od początku zawodów) udało się zaakceptować zadanie drugie. Czas = 46min + (90 + 40)min = 176 minut Czyli 2h i 56 min.

Żeby wiedzieć które zadania są łatwiejsze należałoby przeczytać treści wszystkich z nich albo jeżeli ranking jest dostępny zobaczyć, które zadanie rozwiązała duża ilość drużyn.

Jeżeli już decydujecie się na zabranie się za jakieś zadanie, raczej złym pomysłem jest od razu pisanie programu, który ma być jego rozwiązaniem. Na początek upewnij się, że zadanie jest dla Ciebie jasne (dobre zrozumienie treści zadania jest bardzo ważne). Następnie zastanów się jak można je najlepiej rozwiązać. Dopiero kiedy będziesz miał algorytm gotowy do implementacji, zacznij pisanie kodu programu.

Oczywiście strategia jaką finalnie obierzecie na zawodach, będzie zależeć wyłącznie od was.

Ważne by przy rozwiązywaniu zadań opierać się wyłącznie na danych podanych w treści zadania. Nawet jeżeli „historyjka" zadania dotyczy tematyki, którą bardzo dobrze znasz z życia codziennego (i Twoje doświadczenie życiowe nakłada z góry na tą sytuację jakieś założenia) to pamiętaj że służy ona tylko ubarwieniu zadania i nie ma nic wspólnego z rzeczywistością. Jeżeli liczba x mająca symbolizować w treści zadania ilość sprawdzianów/kolokwiów w jednym dniu jest opisana następująco: x($0 \leq x \leq 20\,000$) to należy przyjąć że na prawdę może być ich tak straszna ilość. Nie należy również zakładać czegokolwiek tylko na podstawie przykładu. Jeżeli w treści zadania nie jest napisane że na wejściu jest ciąg posortowany, a w przykładzie takowy akurat się znajduje to nie wolno wam przyjąć, że tak będzie w każdym teście. Za pewnik natomiast możecie wziąć specyfikacje wejścia w zadaniu. Jeżeli w treści zadania jest informacja że na wejściu będą dwie liczby całkowite oddzielone spacją i większe od zera to tak dokładnie będzie. Nie musicie, a nawet jest to bardzo niewskazane pisać zabezpieczeń sprawdzających poprawność danych wejściowych.

Wczytywanie danych i wypisywanie wyników przez program

Programy tworzone przez was na zawodach powinny czytać dane ze standardowego wejścia i wypisywać rozwiązania na standardowe wyjście (tzn. powinniście pisać program tak by jako program konsolowy czytał dane z klawiatury i wypisywał wyniki na ekran). „Sprawdzarka" przy testowaniu waszych rozwiązań sama przekieruje operacje odczytywania i wypisywania.

Pamiętajcie by nie wypisywać nic ponad to, czego oczekuje od was treść zadania. Nie wypisujcie żadnych dodatkowych komunikatów takich jak *„Podaj n:"*, *„Wynikiem jest:"* itp. Takie komunikaty zostaną potraktowane jako odpowiedzi (oczywiście błędne) waszego programu, co spowoduje zwrócenie przez „Sprawdzarkę" **„NIE- Zła odpowiedź"** i nie zaliczenie zadania. W kodzie wysyłanym do testu nie powinny również znajdować się żadne dodatkowe komendy czekające na interakcję użytkownika. Jeżeli będzie takich używać na swoich stanowiskach, np. by podejrzeć wyniki działania programu, pamiętajcie by je usunąć lub zakomentować przed wysłaniem kodu do sprawdzenia.

Przy wypisywaniu należy pamiętać by wypisując dwie różne liczby oddzielić je spacją (np. jeżeli chcemy wypisać liczby 4 i 6 to zrobimy to tak *„4 6"*, a nie tak *„46"* - w drugim przypadku „Sprawdzarka" potraktuje to jak jedną liczbę. Należy stosować się do specyfikacji wyjścia wstawiając np. znaki nowej linii w odpowiednim miejscu.

Przykładowe kody będą w dalszej części tutorialu wraz z omówieniem.

Przykładowe kody i... ich efekty

Wszystkie kody programów będą odwoływać się do przykładowego zadania powstałego specjalnie na potrzeby tego tutorialu. Treść zadania można znaleźć w dziale FAQ na stronie konkursu.

Kiedy mamy już gotowy algorytm rozwiązania problemu należy go zaimplementować. Otóż to na zadany problem napisaliśmy następujący kod (pamiętając jedno z najpopularniejszych metod sortowania – sortowanie bąbelkowe):

```
#include <stdio.h>
int main ()
{
    int tab[100],n,i,j;

    scanf("%d\n",&n)    // <--------- brak średnika
    for(i=0;i<n;i++)
        scanf("%d ",&tab[i]);

    for (i=0;i<n-1;i++)
    {
        for (j=0;j<n-1-i;j++)
        {
            if(tab[j]>tab[j+1])
            {
               tab[j]+=tab[j+1];                // zamiana
               tab[j+1]=tab[j]-tab[j+1];  // zmiennych
               tab[j]-=tab[j+1];                 // miejscami
            }
        }
    }
    for(i=0;i<n;i++)
        printf("%d ",tab[i]);

    return 0;
}
```

Kiedy program jest już napisany i jesteście przekonani, że działa poprawnie, powinniście go jak najszybciej wysłać. Pamiętajcie, że czas wysłania rozwiązania liczy się w klasyfikacji końcowej. Wysłanie tego programu spowoduje jednak, że „Sprawdzarka" zwróci ocenę:

„NIE - błąd kompilacji"

Oznacza to, że nasz program nie skompilował się prawidłowo. Przyczyna błędu kompilacji jest wskazane przez komentarz w kodzie programu (brak średnika). W przypadku otrzymania takiego komunikatu, należy zabrać się za poszukanie błędu i poprawienie go. Na swoich stanowiskach będziecie mieli do dyspozycji kompilatory, które powinny pomóc wam odszukać błąd. Pamiętajcie jednak by trzymać się standardu języka w którym piszecie – jeżeli będziecie używać komend charakterystycznych tylko dla danego środowiska najprawdopodobniej będzie zwrócony komunikat **„NIE - błąd kompilacji"** mimo że na waszych stanowiskach kompilacja będzie przebiegała bez problemu.

Wysyłamy kolejną wersję.

Szybko odnaleźliśmy błąd i postanawiamy wysłać taki oto kod:

```
#include <stdio.h>
int main ()
{
    int tab[100],n,i,j;

    scanf("%d\n",&n);
    for(i=0;i<n;i++)
        scanf("%d ",&tab[i]);

    for (i=0;i<n-1;i++)
    {
        for (j=0;j<n-1-i;j++)
        {
            if(tab[j]>tab[j+1])
            {
               tab[j]+=tab[j+1];                // zamiana
               tab[j+1]=tab[j]-tab[j+1];  // zmiennych
               tab[j]-=tab[j+1];                 // miejscami
            }
        }
    }
    for(i=0;i<n;i++)
        printf("%d ",tab[i]);

    return 0;
}
```

Wysłanie takiego programu spowoduje, że „Sprawdzarka” zwróci ocenę:

„NIE - zła odpowiedź”

Cóż... Nasz kod skompilował się poprawnie, ale ma błąd wykonania. Ten komunikat oznacza, że najprawdopodobniej nasz program chciał wykonać operację niedozwoloną np. przepełnił stos (może się tak zdarzyć przy wywołaniu funkcji rekurencyjnej bez warunku końcowego), dzieli w którymś miejscu przez zero, albo próbuje się odwołać do niezarezerwowanego miejsca w pamięci. W naszym przypadku jest to ta ostatnia opcja. Dlaczego? W specyfikacji wejścia podane jest, że ciąg może mieć 10 000 elementów, podczas gdy nasza tablica, która ma przechowywać jego elementy ma 100 elementów. Za to przewinienie otrzymamy 20 minut karnych.

Chcą dużo? Dam im jeszcze więcej!

By mieć pewność, że nasz program będzie odwoływał do wystarczająco pojemnej tablicy, postanowiliśmy zadeklarować sporo większą niż jest wymagane:

```
#include <stdio.h>
int main ()
{
    int tab[10000000],n,i,j;
    scanf("%d\n",&n);
    for(i=0;i<n;i++)
        scanf("%d ",&tab[i]);

    for (i=0;i<n-1;i++)
    {
        for (j=0;j<n-1-i;j++)
        {
            if(tab[j]>tab[j+1])
            {
               tab[j]+=tab[j+1];                // zamiana
               tab[j+1]=tab[j]-tab[j+1];  // zmiennych
               tab[j]-=tab[j+1];                 // miejscami
            }
        }
    }
    for(i=0;i<n;i++)
        printf("%d ",tab[i]);

    return 0;
}
```

Wysłanie takiego programu spowoduje, że „Sprawdzarka” zwróci ocenę:

„NIE - zła odpowiedź”

Co tym razem? Nasz program próbował, więc zaalokować więcej niż może. Dobrym pomysłem jest stworzenie tablicy większej niż jest wymagana, ale należy to czynić z umiarem. 10 elementów rezerwy powinno już w zupełności wystarczyć. Jeżeli przesadzimy z zapasem(użyjemy więcej pamięci niż przewidział autor zadania) możemy spodziewać się komunikatu:

„NIE - przekroczenie pamięci”

No dobra to może jednak nie tak dużo....

```
#include <stdio.h>
int main ()
{
    int tab[10001],n,i,j;
    scanf("%d\n",&n);
    for(i=0;i<n;i++)
    scanf("%d ",&tab[i]);
    for (i=0;i<n-1;i++)
    {
        for (j=0;j<n-1-i;j++)
        {
            if(tab[j]>tab[j+1])
            {
               tab[j]+=tab[j+1];                // zamiana
               tab[j+1]=tab[j]-tab[j+1];   // zmiennych
               tab[j]-=tab[j+1];                 // miejscami
            }
        }
    }
    for(i=0;i<n;i++)
        printf("%d ",tab[i]);

    return 0;
}
```

Wysłanie takiego programu spowoduje, że „Sprawdzarka” zwróci ocenę:

„NIE - przekroczenie czasu”

Taki komunikat oznacza, że w program działał zbyt długo gdyż sortowanie bąbelkowe zastosowane przez nas jest jednym z najbardziej kosztownych czasowo algorytmów sortowania. Prawdopodobnie autor zadania jako wzorcowe rozwiązanie tego problemu przyjął szybszy algorytm co spowodowało, że nasz nie zmieścił się w ustalonym limicie czasowym dla tego zadania. Niestety dostajemy kolejne karne 20 minut dla tego zadania. Jeżeli program działa zbyt długo, nie jest sprawdzana poprawność udzielonych odpowiedzi.

Pamiętaj, że „Sprawdzarka” testuje program z innymi danymi niż te, które zawiera przykład dostarczony do zadania. Jeżeli w treści zadania możemy wyczytać, że nasz ciąg może mieć 10 000 elementów to takiego należy się podziewać. Często też testy zawierają przypadki szczególne.

Trzeba, więc zmienić algorytm ...

Wprowadźmy, więc znacznie szybszy algorytm sortowana przez wybór:

```
#include <stdio.h>
int main ()
{
    int tab[10001],n,i,j,ind_min;
    scanf("%d\n",&n);
    for(i=0;i<n;i++)
        scanf("%d ",&tab[i]);
    for (i=0;i<n-1;i++)
    {
        ind_min=i;
        for (j=i+1;j<n;j++)
        {
            if(tab[j]<tab[ind_min]) ind_min=j;
        }
        if (ind_min!=i)
        {
            tab[i]+=tab[ind_min];   //zamiana
            tab[ind_min]=tab[i]-tab[ind_min]; //zmiennych
            tab[i]-=tab[ind_min];   //miejscami
        }
    }
    printf("Wynikiem jest:");
    for(i=0;i<n;i++)
        printf("%d ",tab[i]);

    return 0;
}
```

Wysłanie takiego programu spowoduje, że „Sprawdzarka” zwróci ocenę:

„NIE - zła odpowiedź”

Nie jest źle, okazuje się bowiem, że nasz program się skompilował, nie wykonywał się zbyt długo i nawet wykonał się prawidłowo. Szkopuł w tym, że nie zwrócił takiej odpowiedzi jaką powinien. Co jest, więc źle? Przecież algorytm i jego implementacja z pewnością jest dobra. Niestety programista stwierdził, że doda mały ozdobnik poprzedzający wypisanie wyniku. „Sprawdzarka” tymczasem jako odpowiedź traktuje każdy znak wypisany na standardowe wyjście, a że ciąg znaków *"Wynikiem jest:"* z pewnością nie należy do posortowanego ciągu, zwraca odpowiedź o błędnej odpowiedzi.

Taka odpowiedź może mieć też inne przyczyny. Oczywiście może to być wina rozłożenia Waszego algorytmu przez jakiś szczególny przypadek, ale nie tylko. Taka odpowiedź również zostanie zwrócona przez „Sprawdzarkę”, gdy dobierzecie za mało pojemny typ dla waszej zmiennej. Zawsze pamiętacie by typy zmiennych, których używacie, mieściły dane które są na wejściu, bądź wartości, które zostaną obliczone w trakcie działania programu. W naszym przypadku typ Integer jest wystarczający (na wejściu mamy liczbę maksymalnie o wielkości 10 000, a w trakcie programu nie wykonujemy żadnych obliczeń, które by powiększyły tą liczbę). W przypadku przepełnienia typu mamy prawie 100% pewności, że pociągnie to za sobą poprawność obliczeń i końcowy wynik.

Organizatorzy umożliwiają opcję wydrukowania kodu waszego programu, może okazać się dobrym pomysłem, wydrukowanie kodu i zamianą z kolegą z innym rozwiązaniem – Ty będziesz wtedy szukać błędu bez użycia komputera.

Wrr... Czas to skończyć

Poprawiamy szybko błąd nadmiernego tekstu i wysyłamy taki kod:

```
#include <stdio.h>
int main ()
{
    int tab[10001],n,i,j,ind_min;
    scanf("%d\n",&n);
    for(i=0;i<n;i++)
        scanf("%d ",&tab[i]);
    for (i=0;i<n-1;i++)
    {
        ind_min=i;
        for (j=i+1;j<n;j++)
        {
            if(tab[j]<tab[ind_min]) ind_min=j;
        }
        if (ind_min!=i)
        {
            tab[i]+=tab[ind_min];   //zamiana
            tab[ind_min]=tab[i]-tab[ind_min]; //zmiennych
            tab[i]-=tab[ind_min];   //miejscami
        }
    }
    for(i=0;i<n;i++)
        printf("%d ",tab[i]);

    return 0;
}
```

Wysłanie takiego programu spowoduje, że „Sprawdzarka” zwróci ocenę:

„TAK”

Zadanie zostało zaliczone!