Marcel Szelwiga
Bartosz Chomiński
Zajęcia z informatyki w XIV LO
Jak testować programy w CP

Niezbędne oprogramowanie

Linux

Oczywiście najlepszym rozwiązaniem jest system Linux z prawdziwego zdarzenia, w szczególności w połączeniu z umiejętnością korzystania z terminala, jednak poradzimy sobie również bez tego.

Windows Subsystem for Linux

Alternatywnym rozwiązaniem jest instalacja maszyny wirtualnej, ten sposób pozwala na uzyskanie większości benefitów posiadania Linuxa – najważniejszym z nich jest oczywiście terminal z prawdziwego zdarzenia. Osobiście polecam Ubuntu 20.04.

Po udanej instalacji, by dostać się do plików windowsowych, należy wejść do katalogu `/mnt/c/`.

Poruszanie się w terminalu

Przydatne polecenia

- `pwd` – wypisanie aktualnej ścieżki,
- `cd ..` – przejście do folderu nadrzędnego (przykładowo z `/mnt/c/` do `/mnt/`),
- `cd <nazwa>` – przejście do folderu niżej (przykładowo z użyciem `cd c` przejdziemy z `/mnt/` do `/mnt/c/`),
- `sudo apt install g++` – instalacja kompilatora,
- `g++ <nazwa1>.cpp -o <nazwa2>` – skompilowanie pliku o nazwie `<nazwa1>.cpp` do pliku wykonywalnego o nazwie `<nazwa2>` (przykładowo `g++ brut.cpp -o brut` skompiluje plik `brut.cpp` do pliku o nazwie `brut`),
- `./<nazwa>` – uruchomienie programu o nazwie `<nazwa>` (przykładowo `./brut`),

- `./<program> < <nazwa>` – uruchomienie programu `<program>` z danymi pobieranymi z pliku `<nazwa>` zamiast z konsoli (przykładowo `./brut < test.in`),
- `./<program> > <nazwa>` – uruchomienie programu `<program>` z danymi zapisywanymi do pliku `<nazwa>` zamiast do konsoli (przykładowo `./brut > test.out`),
- `./<program> < <nazwa1> > <nazwa2>` – uruchomienie programu `<program>` z danymi pobieranymi z pliku `<nazwa1>` i zapisywanymi do pliku `<nazwa2>` zamiast z i do konsoli (przykładowo `./brut < test.in > test.out`).

Pisanie skryptów

Edytor tekstu

Warto nauczyć się podstaw wybranego popularnego terminalowego edytora tekstu. Osobiście bardzo polecam `nano`, ponieważ jest naprawdę proste w obsłudze (`Ctrl + S` by zapisać plik i `Ctrl + X` by wyjść z pliku).

Ambitniejszym i bardziej odpornym na wysoki próg wejściowy użytkownikom poleca się edytor `vim`. Popularne są również `nvim`, `gedit` i `emacs`.

Pierwszy skrypt

Możemy utworzyć plik `sc.sh` o zawartości

```
for i in {1..10}
do
    echo $i
done
```

Aby go uruchomić najpierw należy nadać plikowi uprawnienia do bycia wykonywanym za pomocą komendy `chmod +x sc.sh`, a następnie standardowo uruchomić program komendą `./sc.sh` – skutkuje to wypisaniem w konsoli kolejnych liczb całkowitych od 1 do 10.

Testowanie samodzielne

Gdy napiszemy rozwiązanie wolne, ale poprawne (tzw. *bruta*) i program generujący testy (tzw. *generatorkę*), o nazwach (skompilowanych) odpowiednio `brut` i `gen` oraz domniemane rozwiązanie wzorcowe (tzw. *wzorcówkę*) o nazwie `wz` (po skompilowaniu), możemy napisać następujący skrypt:

```
for i in {1..10}
do
    ./gen > in
    ./wz < in > wzout
    ./brut < in > brout
    diff -b wzout brout || break
    echo $i
done
```

Skrypt ten dziesięć razy wygeneruje test przy użyciu generatorki, policzy odpowiedź dla niego według wzorcówki i bruta, a następnie porówna obie odpowiedzi; jeśli będą się one różnić, to przerwie działanie skryptu.

Uwaga techniczna: zaprezentowany skrypt przy porównywaniu plików ignoruje zmiany w liczbie odstępów między słowami – `a␣b` nie zostanie odróżnione od `a␣␣b`. O pozostałych opcjach skryptu `diff` można przeczytać tutaj.

„Losowość”

Pisząc własną generatorkę, chcemy, aby z każdym jej uruchomieniem losowane były inne liczby, a więc potrzebujemy odpowiednio losowego ziarna. Przykładowym źródłem losowości może być zegar bardzo wysokiej precyzji `chrono::steady_clock::now().time_since_epoch().count()` z biblioteki `chrono` lub funkcja `getpid()` z biblioteki `unistd.h`. Program

```
#include <iostream>
#include <unistd.h>
using namespace std;
int main(){
    srand(getpid());
    cout << rand() << "\n";
    return 0;
}
```

będzie z każdym uruchomieniem wypisywał rozsądnie różne i nieprzewidywalne liczby.

Nie warto używać `time(NULL)`, ponieważ ziarno będzie zależało tylko od sekundy, w której wykona się program. Wykonanie takiego programu 1 000 razy między godzinami 23:07:20,43 i 23:07:21,37 wygeneruje tylko dwa różne testy o ziarnach 1668550040 i 1668550041 (ziarna prawdziwe dzisiaj, 15.11.2022).

Testowanie z paczkami testów z forum OI

Kiedy mamy już zestaw testów, na przykład w postaci:

- Dane wejściowe: `tests/in/aaa<numer>.in`,
- Dane wyjściowe: `tests/out/aaa<numer>.out`,

to skrypt testujący powinien wyglądać następująco:

```
for i in {1..10}
do
    ./wz < in > tests/in/aaa$i.in > out
    diff -b in tests/out/aaa$i.out || break
    echo $i
done
```

Bajery

- `timeout 2s /usr/bin/time -f "Wz Time: %E " ./wz < in > out` – ograniczenie czasu wykonania programu do dwóch sekund i wypisanie czasu wykonania programu.
- `g++ a.cpp -O3` – kompilacja pliku stosując optymalizację jak na OI (program zwykle będzie działał szybciej).