Zajęcia V

Cele:

- omówić obsługę plików
- wprowadzić struktury

Problem

Jak wprowadzić do programu duże ilości danych?

Odczyt plików

```cpp
#include <fstream>

int main() {
    std::ifstream file("pliczek.txt");
    std::string line;

    while( std::getline(file, line) ) {
        // process line
    }

    return 0;
}
```

```python
with open("pliczek.txt") as file:
    for line in file:
        # process line
```

Należy pamiętać sprawdzić, czy plik poprawnie został otworzony!

```
...
std::ifstream file("pliczek.txt");

if( !file ) {
    // process error
}
```

W C jest trochę inaczej:

```c
#include <stdio.h>

int main() {
    FILE *file = fopen("pliczek.txt", "r");
    if( !file ) return 1;   // error occur, exiting

    char line[1024];

    while( fgets(line, 1024, file) ) {
        // process line
    }

    fclose(file);
    return 0;
}
```

Zapis do pliku

```cpp
#include <fstream>

int main() {
    std::ofstream file("pliczek.txt");

    std::string entries[2] = {
        "Drodzy studenci,",
        "Uczcie się :)"
    };

    for( size_t i = 0; i < 2; i++ )
        file << entries[i] << std::endl;

    return 0;
}
```

W tym przypadku, zawartość pliku zostanie nadpisana, aby dopisać dane do istniejącego pliku, należy:

```
std::ofstream file(filepath, std::ios::app);
```

(std -> standard, ios -> input/output stream, app -> append)

main

```
int main();
```

```
int main(int argc, char* argv[]);
```

```
int main(int argc, char* argv[]);
```

```
./main Hello plik.txt

argc -> 3
argv[0] -> "./main"
argv[1] -> "Hello"
argv[2] -> "plik.txt"
```

```
./program 10 20 30 ATCG

argc -> 5
argv[0] -> "./program"
argv[1] -> "10"
argv[2] -> "20"
argv[3] -> "30"
argv[4] -> "ATCG"
```

Struktury

```
struct dna {
    int a;
    int t;
    int c;
    int g;

    float gc;
};  // note semicolon
```

```cpp
#include <iostream>

struct dna_t {
    int a;
    int t;
    int c;
    int g;

    float gc;
};

int main(int argc, char* argv[]) {
    dna_t dna;
    dna.a = 10;
    dna.c = 5;

    std::cout << dna.a << std::endl;
    std::cout << dna.t << std::endl;
}
```

```c
#include <iostream>

struct dna_t
{
    int a,t,c,g;
    float gc;
};

void process(dna_t *dna)
{
    dna->t = 3;
    std::cout << dna->a << std::endl;
    std::cout << dna->t << std::endl;
}

int main(int argc, char *argv[])
{
    dna_t dna;
    dna.a = 1;
    process(&dna);
}
```

Zadanie I

Napisz program, który:

- policzy zawartość poszczególnych nukleotydów w ciągu
- musi korzystać ze struktur/y i instrukcji `switch`
- musi wczytać pojedynczy łańcuch dna przez wejście standardowe
- musi zawierać funkcję o podanej sygnaturze:
  `dna_t count_nucleotides( char * dna )`

- dla ułatwienia, ciągi dna będą zapisane dużymi literami oraz będą zawsze poprawne.

Przykład

Wejście

```
ATTCG
```

Wyjście

```
A: 1 T: 2 C: 1 G: 1
```

Zadanie II

Napisz program, który:

- wczyta z pliku ciągi DNA, oraz dla każdego ciągu w pliku wydrukuje zawartość `GC` (procentową, minimum 2 miejsca po przecinku, np. 72.24%)
- przed obliczeniem zawartości `GC` sprawdzi, czy ciąg jest prawidłowy, jeśli nie, to wydrukuje odpowiednią wiadomość
- obsłuży małe i duże litery

- ścieżka do pliku wejściowego musi być zdefiniowana jako zmienna globalna
- każdy ciąg wejściowy musi mieć dokładnie jedną linię wyjściową
- musi zawierać co najmniej 3 funkcje: `main`, `validate_dna` oraz `calculate_gc`
- funkcja `calculate_gc` oraz `validate_dna` nie może wewnątrz swojego ciała zawierać `std::cout`

Struktura pliku wejściowego:

```
1: ATGGC
2: ATGCGCGatcgGTTT
...
12: gc
...
n: GCTCGA
```

Przykład:

Wejście:

```
1: atCG
2: TTAT
```

Wyjście:

```
1: 50.00
2: 0.00
```

Zadanie III

Napisz program, który:

- wczyta z pliku pary ciągów dna i zapisze dla nich do osobnego pliku pozycje (po przecinku), dla których znaki w ciągach różniły się pomiędzy sobą
- obsłuży małe i duże litery
- sprawdzi czy ciąg jest prawidłowy, jeśli nie, to zapisze do pliku o tym informację (która musi zawierać słowo `invalid`)

- każda para dna musi mieć dokładnie jedną linię w pliku wyjściowym
- ścieżka do plików musi być zdefiniowana jako zmienna globalna
- cały kod nie może mieścić się tylko w `main`
- dla ułatwienia, plik wejściowy gwarantuje równą długość ciągów w parze oraz gwarantuje istnienie poprawnych par

Przykład:

Wejście w pliku:

```
TTAT
ttCt
ilovecpp
justkidd
TTAT
ttCG
```

Wyjście w pliku:

```
2
invalid dna
2,3
```