### Politechnika Śląska

Wydział Informatyki, Elektroniki i Informatyki

Podstawy programowania komputerów

Darwin

#### autor Kuba Domerecki

prowadzący Dariusz Marek

rok akademicki 2020/2021

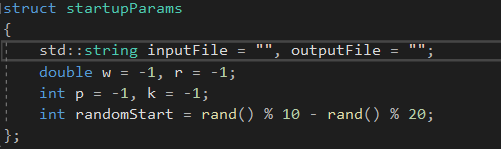
kierunek Informatyka

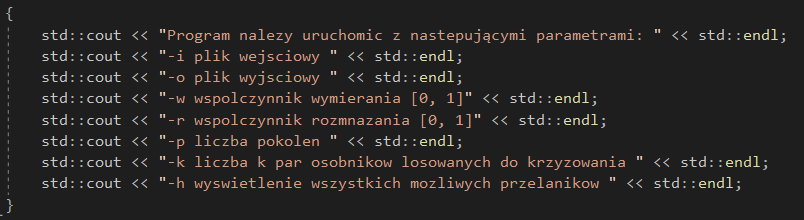
rodzaj studiów stacjonarne (dzienne)

semestr 3

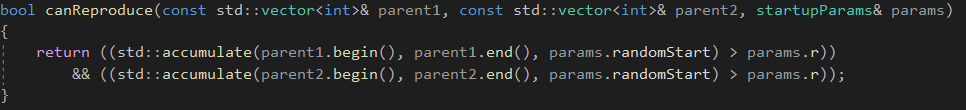
# Opis programu

Program napisany strukturalnie korzystając z Visual Studio 2019. Do przechowywania danych otrzymanych z wiersza poleceń podczas uruchomiania programu użyto struktury startupParams.

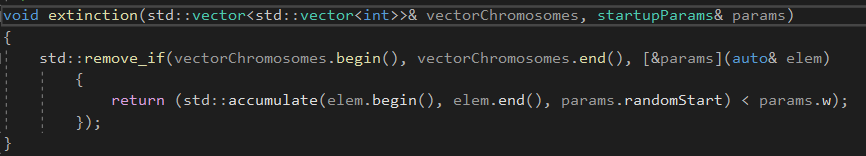


W funkcji run sprawdzamy czy uruchomiono program z poprawną ilością przełączników lub czy z przełącznikiem –h lub bez żadnego przełącznika. Brak przełącznika lub wywołanie programu z przełącznikiem –h spowoduje wypisanie na konsole pomocy z użytkowania programu:

Jeśli uruchomimy program z odpowiednimi argumentami, pobierzemy dane z początkowego pokolenia z pliku wejściowego. Następnie uruchomi się pętla z ilością iteracji równą p(liczba pokoleń podana podczas uruchomienia programu). W każdej iteracji losujemy k par do krzyżowania. Rozmnażanie danej pary dochodzi do skutku tylko jeśli warunek rozmnażania zostanie spełniony:



Po wszystkich k krzyżowań dla danego pokolenia wołana jest funkcja wymierania, która może usunąć niektórych osobników:

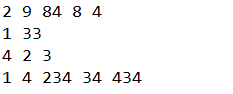


Po zakończeniu wszystkich iteracji dla pokoleń ostateczna populacja zapisywana jest do pliku wyjściowego.

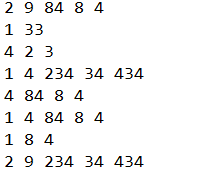
# Opis przykładowego testu

Argumenty programu: -o output.txt -i input.txt -r 0.5 -w 0.3 -p 2 -k 2

Zawartość pliku input.txt:



Zawartość pliku wyjściowego output.txt po zakończeniu działania programu:

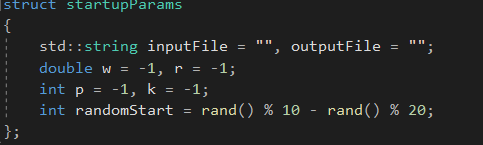


# 3 Struktury danych

Główną strukturą danych był vector z biblioteki standardowej użyty do deklaracji kontenera przechowującego chromosomy wszystkich osobników utworzony lokalnie w funkcji run:



Kolejna struktura jest struktura startowa, służąca tak jak wyżej opisane do przechowywania danych otrzymanych z wiersza poleceń tzn. plik wejściowy, plik wyjściowy, odpowiedni współczynnik wymierania oraz rozmnażania, liczba pokoleń i krzyżowań osobników, a także losową liczbę wykorzystaną przy funkcjach w programie.



# 4 Funkcje

void **help**()

***zwraca*** ***na*** ***konsole*** ***informacje*** ***z*** ***jakimi*** ***przełącznikami*** ***można*** ***uruchomić program***

bool **init**(char \*\*argGiven, startupParams& params)

***funkcja*** ***parsuje*** ***argumenty wywołania programu***

* ***argGiven – parametry wywołania programu*** ***podane*** ***wraz*** ***z*** ***jego*** ***wartościami***
* ***params – struktura przechowująca parametry startowe***

bool **load**(std::vector<std::vector<int>> &vectorChromosomes, startupParams& params)

***funkcja*** ***odczytuje*** ***plik*** ***wejściowy***

* ***vectorChromosomes -*** ***kontener*** ***przechowujący*** ***chromosomy*** ***wszystkich*** ***osobników***
* ***params – struktura przechowująca parametry startowe***

void **print**(*const* std::vector<std::vector<int>> &vectorChromosomes)

***funkcja*** ***wyświetla*** ***na*** ***konsole*** ***aktualnie*** ***istniejące*** ***chromosomy***

* ***vectorChromosomes -*** ***kontener*** ***przechowujący*** ***chromosomy*** ***wszystkich*** ***osobników***

void **save**(*const* std::vector<std::vector<int>> &vectorChromosomes, startupParams& params)

***funkcja*** ***zapisuje*** ***do*** ***pliku*** ***wyjściowego*** ***wszystkie*** ***chromosomy*** ***które*** ***przeżyły*** ***do*** ***ostatniego*** ***pokolenia***

* ***vectorChromosomes -*** ***kontener*** ***przechowujący*** ***chromosomy*** ***wszystkich*** ***osobników***
* ***params – struktura przechowująca parametry startowe***

bool **canReproduce**(*const* std::vector<int> &parent1, *const* std::vector<int> &parent2, startupParams& params)

***funkcja*** ***sprawdza*** ***czy*** ***wylosowana*** ***para*** ***osobników*** ***może*** ***się*** ***rozmnażać***

* ***parent1 - chromosom rodzica 1***
* ***parent2 - chromosom rodzica 2***
* ***params – struktura przechowująca parametry startowe***

void **newGeneration**(std::vector<std::vector<int>> &vectorChromosomes, startupParams& params)

***funkcja*** ***dodaje*** ***do*** ***kontenera*** ***przechowujących*** ***chromosomy*** ***nowo*** ***narodzone*** ***osobniki***

* ***vectorChromosomes -*** ***kontener*** ***przechowujący*** ***chromosomy*** ***wszystkich*** ***osobników***
* ***params – struktura przechowująca parametry startow***

void **extinction**(std::vector<std::vector<int>> &vectorChromosomes, startupParams& params)

***funkcja*** ***usuwa*** ***osobniki*** ***z*** ***kontenera,*** ***które*** ***spełniają*** ***warunek*** ***wymarcia***

* ***vectorChromosomes -*** ***kontener*** ***przechowujący*** ***chromosomy*** ***wszystkich*** ***osobników***
* ***params – struktura przechowująca parametry startowe***

void **run** (int argStartup, char \*\* argGiven, startupParams& params)

***funkcja*** ***sprawdzajaca argumenty i wykonujaca odpowiednie instrukcje dla nich***

* ***argStartup – ilość argumentów wywołania***
* ***argGiven – argumenty wywołania***
* ***params – struktura przechowująca parametry startowe***

# 5 Podsumowanie

Program początkowo wydawał się dość ciężki i treść zadania nie była na pierwszy rzut oka zbytnio przyjemna, lecz w trakcie pisania zmieniło się to. Pisanie strukturalne na pewno charakteryzuje się wykonywaniem i odpowiednim wyborem kolejnych instrukcji oraz powtarzaniem ich do momentu spełnienia warunku co widać w moim programie. Myślę, iż w przyszłości z pewnością wykorzystam coś czego się nauczyłem w trakcie wykonywania tego projektu.