Politechnika Śląska w Gliwicach
Wydział Automatyki, Elektroniki i Informatyki

Podstawy Programowania Komputerów

DARWIN

autor	Adam Trznadel
prowadzący	mgr inż. Maciej Długosz
rok akademicki	2018\2019
kierunek	informatyka
rodzaj studiów	SSI
semestr	1
termin laboratorium / ćwiczeń	wtorek, 12:00 – 13:30
grupa	2
sekcja	5
termin oddania sprawozdania	2019-01-19
data oddania sprawozdania	2019-01-08

1 Treść zadania

Napisać program symulujący ewolucję populacji osobników. Populacja może liczyć dowolną liczbę osobników. Każdy osobnik zawiera chromosom, który jest ciągiem liczb naturalnych. Chromosomy mogą być różnej długości. W każdym pokoleniu wylosowywanych jest k par osobników, które się następnie krzyżują. Krzyżowanie polega na tym, że u każdego osobnika dochodzi do pęknięcia chromosomu w dowolnym miejscu. Część początkowa chromosomu jednego osobnika łączy się z częścią końcową drugiego. Inaczej mówiąc: osobniki wymieniają się fragmentami swoich chromosomów. Jeden osobnik może być wylosowany do kilku krzyżowań. Po dokonaniu wszystkich krzyżowań w pokoleniu sprawdzane jest przystosowanie osobników do warunków środowiska. W tym celu dla każdego osobnika wyznaczana jest wartość $f \in [0, 1]$ funkcji dopasowania. Osobniki, dla których wartość f < w (gdzie w jest progiem wymierania), są usuwane z populacji. Osobniki, dla których f > r (gdzie r jest progiem rozmnażania) są klonowane. A osobniki, dla których w <f<r pozostają w populacji, ale się nie rozmnażają. Program uruchamiany jest z linii poleceń z wykorzystaniem następujących przełączników (kolejność przełączników jest dowolna): -i plik wejściowy z populacją -o plik wyjściowy z populacją -w współczynnik wymierania $w \in [0, 1]$ -r współczynnik rozmnażania $r \in [0, 1]$ -p liczba pokoleń p -k liczba k par osobników losowanych do krzyżowania

Program uruchamiany jest z linii poleceń z wykorzystaniem następujących przełączników:

- `-p` liczba pokoleń p
- `-k` liczba k par osobników losowanych do krzyżowania
- `-w` współczynnik wymierania $w \in [0, 1]$
- `-r` współczynnik rozmnażania $r \in [0, 1]$
- `-i` plik wejściowy z populacją
- `-o` plik wyjściowy z populacją

2 Analiza zadania

Zagadnienie przedstawia jeden z problemów ewolucji populacji osobników.
Osobniki są reprezentowane za pomocą ciągów znaków.

2.1 Struktury danych

W programie użyto dynamicznych struktur danych, tj. lista, lista list.

2.2 Algorytmy

Program modyfikuje populację poprzez *krzyżowanie/ klonowanie/* usuwanie istniejących osobników,. Wyżej wymienione operacje są wykonywane p razy co reprezentuje ilość nowych pokoleń. Po zakończenia tzw. cyklu cala populacja zostaje zapisana do pliku.

3 Specyfikacja zewnętrzna

Program jest uruchamiany z linii poleceń. Należy przekazać do programu nazwy plików: wejściowego i wyjściowego po odpowiednich przełącznikach (odpowiednio: `-i` dla pliku wejściowego i `-o` dla pliku wyjściowego `-p` liczba pokoleń p `-k` liczba k par osobników losowanych do krzyżowania `-w` współczynnik wymierania $w \in [0, 1]$ `-r` współczynnik rozmnażania $r \in [0, 1]$), np.

```
program -i populacja.txt -o wynik -p 1 -k 3 -w 0.6 -r 0.8
program -o wynik.txt -p 2 -k 3 -r 0.7 -w 0.3 -i populacja.txt
```

Pliki są plikami tekstowymi, ale mogą mieć dowolne rozszerzenie (lub go nie mieć). Przełączniki mogą być podane w dowolnej kolejności.

Jeżeli plik wejściowy nie będzie znajdował się w odpowiedniej lokalizacji, lub będzie błędny, program wyświetli komunikat:

```
Blad pliku odczytu
```

i zakończy działanie. Jeżeli nie uda się utworzyć pliku wyjściowego, program wyświetli komunikat:

```
Blad pliku zapisu
```

i zakończy działanie.

Jeżeli nastąpi błędne wpisanie parametrów wejścia lub/i wyjścia np.

```
program -i -w 0.3 -r 2
```

Program wyświetli komunikat:

```
Bledne parametry
```

i zakończy działanie.

Jeżeli plik z osobnikami nie będzie posiadał poprawnej populacji, tj. składającej się z ciągów liczb całkowitych dodatnich oddzielonych spacjami/znakami nowej linii jeżeli chcemy rozróżnić osobniki, program wyświetli komunikat:

```
Nieprawdilowe dane
```

następnie zakończy działanie.

4 Specyfikacja wewnętrzna

4.1 Typy zdefiniowane w programie

Lista przechowująca informacje o poszczególnych chromosomach.

4.1.1 Geny

```
1 struct Geny
2 {
3     ///wartosc chromosomu
4     int     chromosom;
5
6     ///wskaznik na kolejny element listy genow
7     Geny *Gnext;
8 };
```

4.1.2 Osobniki

Lista przechowująca informacje o poszczególnych osobnikach.

```
1 struct Osobniki
2 {   ///dlugosc listy z chromosomami
3     int Odlugosc;
4
5     ///wartosc funckji przystowsowania dla osobnika
6     double przystosowanie;
7
8     ///wskaznik na poczatek listy z chromosomami
9     Geny *Gglowa;
10
11    ///wskaznik na kolejny element listy osobnikow
12    Osobniki *Onext;
13 };
```

4.1.3 Populacja

Struktura przechowująca listę osobników i jej długość.

```
1 struct Populacja
```

```
{
    ///dlugosc listy z osobnikami
    int Ldlugosc;

    ///wskaznik na pierwszy element listy Osobnikow
    Osobniki *Oglowa;
};
```

4.1.4 dwie_liczby

Struktura przechowująca dwie liczby calkowite.

```
struct dwie_liczby
{
    int a;
    int b;
};
```

4.1.5 dwa_wsk

Struktura przechowująca dwa wskazniki na elementy listy osobnikow.

```
struct dwa_wsk
{
    Osobniki *Oa;
    Osobniki *Ob;
};
```

4.1.6 Parametry

Struktura przechowująca parametry.

```
struct Parametry
{
    ///ilosc par do losowania
    int k;

    ///ilosc pokolen
    int p;

    ///prog wymierania
```

```
10      double w;
11
12      ///prog rozmnazania
13      double r;
14
15      ///nazwa pliku do odczytu
16      std::string plikin;
17
18      ///nazwa pliku do zapisu
19      std::string plikout;
20  };
```

4.2 Ogólna struktura programu

Pliki używane w programie:

- main.cpp główny plik źródłowy
- funkcje.cpp plik z ciałami funkcji
- funkcje.h plik z nagłówkami funkcji
- struktury.h plik z definicjami struktur

Plik main.cpp odpowiada za uruchamianie funkcji i wypisywania odpowiednich do nich komunikatów o błędach.

4.3 Szczegółowy opis implementacji funkcji

4.3.1 main

```
1  int main(int ile, char * params[])
2  {
3      atexit(onExit); // wycieki pamieci
4      Populacja populacja{ 0,nullptr };
5      Parametry parametry;
6
7      if (pob_param(parametry, ile, params))
8      {
9          cout << "Bledne_parametry";
```

```
10        return 1;
11    }
12
13
14    switch (pob_pop(populacja, parametry.plikin))
15    {
16    case 1: cout << "Nieprawdilowe dane";
17        break;
18
19    case 2: cout << "Blad pliku odczytu";
20        break;
21
22    default: if (cykl(populacja, parametry.k, parametry.
          p, parametry.w, parametry.r))
23        cout << "Za malo osobnikow w populacji";
24            else
25        if (zapisz(populacja, parametry.plikout))
26            cout << "Blad pliku zapisu";
27        break;
28    }
29
30    usun(populacja.Oglowa);
31
32    return 0;
33 }
```

Funkcja main odpowiada za główne działanie programu. Na początku deklarowane(i "zerowane") są zmienne odpowiadające za populacje i parametry. Następnie poprzez funkcje pob_param(*str*.15) pobierane są parametry i sprawdzana jest ich poprawność. Jeżeli funkcja przejdzie dalej, natknie się na instrukcje switch, która będzie uruchamiała odpowiednie funkcje i dzięki wartościom zwracanym będzie wyświetlała komunikaty lub nie, jeżeli wszystko będzie dobrze.
Funkcje wykorzystywane wewnątrz ww. switch'a:
-cykl (*str*.8)
-zapisz (*str*.23)
Następnie zwalniane jest miejsce w pamięci za pośrednictwem funkcji usun(*str*.22).

4.3.2 cykl

Parametry:
-populacja populacja na ktorej beda wykonwyane modyfikacje,
-k ilosc par do losowania
-p ilosc pokolen
-w próg wymierania
-r próg rozmnazania
Funkcja zwraca kody bledow, tj. 0-cykl zakonczony pomyslnie, 1-za malo osobnikow.

```
1  bool cykl(Populacja &populacja,const double &k,const
       double &p, const double &w, const double &r)
2   {
3
4       srand(time(NULL));
5
6       dwa_wsk para_osobnikow;
7
8       if (populacja.Ldlugosc < 2)
9          return true;
10
11      for (int i = 0; i < p; i++)
12      {
13         if (populacja.Ldlugosc > 1)
14         {
15            for (int j = 0; j < k; j++)
16            {
17               para_osobnikow = przesuwanie(populacja,
                     losowanie_par(populacja.Ldlugosc));
18
19               krzyzowanie(para_osobnikow.Oa,
                     para_osobnikow.Ob);
20            }
21
22            spr_przystosowania(populacja, w, r);
23         }
24         else
25            return false;
26      }
27
28      return false;
```

```
29 }
```

Funkcja przeprowadza modyfikacje na populacji, tj. krzyżowanie(*str*.14), usuwanie osobników i klonowanie poprzez funkcje sprawdzanie przystosowania(*str*.20).
Funkcje pomocnicze:
-losowanie par(*str*.15) jest wykorzystywana przez funkcje przesuwanie
-przesuwanie(*str*.20)

4.3.3 czy_calkowita1

Parametry:
-Tekst badany ciąg znaków
Funkcja zwraca kody błędów, tj. 0-ciąg spełnia warunki 1-ciąg nie spełnia warunków

```
1 bool czy_calkowita(const char* Tekst)
2 {
3     string znaki = "0123456789";
4     string tekst = Tekst;
5
6     return (tekst.find_first_not_of(znaki) != string::
          npos);
7 }
```

Funkcja sprawdza czy ciąg znaków spełnia warunki liczby całkowitej.

4.3.4 czy_calkowita2

Parametry:
-tekst badany string
Funkcja zwraca kody błędów, tj. 0-ciąg spełnia warunki 1-ciąg nie spełnia warunków.

```
1 bool czy_calkowita(const string tekst)
2 {
3     string znaki = "0123456789 ";   //spacja jest
          wymagana(czytanie z linii)
4
5     return ((tekst.find_first_not_of(znaki) != string::
          npos));
6 }
```

Funkcja sprawdza czy tekst spełnia warunki liczby całkowitej, zezwalając przy tym na spacje.

4.3.5 czy_plik

Parametry:
-Tekst badany ciąg znaków Funkcja zwraca kody błędów, tj. 0-ciąg spełnia warunki 1-ciąg nie spełnia warunków

```
1 bool czy_plik(const char* Tekst)
2 {
3     string znaki = "\\/*?:|";//znaki niedozowolone przez
          system windows
4     string tekst = Tekst;
5
6     return (tekst.find_first_of(znaki) != string::npos);
7 }
```

Funkcja sprawdza czy string Tekst spełnia warunki dotyczące nazwy pliku.

4.3.6 czy_puste

Parametry:
-tekst który będzie sprawdzany.
Funkcja zwraca kody błędów, tj. 0-ciąg spełnia warunki 1-ciąg nie spełnia warunków

```
1 bool czy_puste(const string tekst)
2 {
3     string znaki = "␣";
4     if (tekst == "") return 1;
5
6     return (tekst.find_first_not_of(znaki) == string::
          npos);
7 }
```

Funkcja sprawdza czy tekst sklada sie ze spacji lub jest pusta linia.

4.3.7 czy_zmienno

Parametry:
-Tekst badany ciąg znaków

Funkcja zwraca kody błędów, tj. 0-ciąg spełnia warunki 1-ciąg nie spełnia warunków

```
1 bool czy_zmienno(const char* Tekst) //rzeczywiste
    nieujemne
2 {
3     string znaki = "0123456789.";
4     string tekst = Tekst;
5
6     return (tekst.find_first_not_of(znaki) != string::
        npos);
7 }
```

Funkcja sprawdza czy Tekst spełnia warunki liczby zmienno-przecinkowej.

4.3.8 dodaj_nakon1

Parametry:
-phead wskaźnik na początek listy z chromosomami
-chromosom wartosc chromosomu

```
1 void dodaj_nakon(Geny *&phead, const int &chromosom)
2  {
3
4      if (phead)
5      {
6          Geny *pheadtmp = phead;
7
8          while (phead->Gnext)
9          {
10             phead = phead->Gnext;
11         }
12         phead->Gnext = new Geny{ chromosom, nullptr };
13         phead = pheadtmp;
14     }
15     else
16         phead = new Geny{ chromosom, nullptr };
17 }
```

Funkcja dodaje chromosomy na koniec listy z chromosomami osobnika.

4.3.9 dodaj_nakon2

Parametry:
-osobniki wskaźnik na osobnika
-chromosomy wskaźnik na pierwszy element listy z chromosomami (genom
-dlugosc-genomu inaczej mówiąc długość listy na która wskazuje phead

```
1  void dodaj_nakon(Osobniki *&osobniki, Geny *chromosomy
       , int dlugosc_genomu)
2  {
3      if (osobniki)
4      {
5          Osobniki *listatmp = osobniki;
6          while (osobniki->Onext)
7          {
8              osobniki = osobniki->Onext;
9          }
10
11         osobniki->Onext = new Osobniki{ dlugosc_genomu,
               f(chromosomy), chromosomy, nullptr };
12         osobniki = listatmp;
13     }
14     else
15         osobniki = new Osobniki{ dlugosc_genomu, f(
               chromosomy), chromosomy, nullptr };
16 }
```

Funkcja dodaje genom(listę chromosomów) osobnika na koniec listy osobników i aktualizuje wartość przystosowania.

4.3.10 f

Parametry:
-param osobnik osobnik którego przystosowanie będzie oceniane
-return zwraca wartość przystosowania

```
1 double f(const Geny * osobnik)
2 {
3
4     int max = 0, dlugosc = 0;
5     double suma = 0;
6     int tmp = 0;
7     if (osobnik)
```

```
8           max = osobnik->chromosom;
9
10
11      while (osobnik)
12      {
13          dlugosc++;
14          tmp = osobnik->chromosom;
15          if (tmp > max) max = tmp;
16          osobnik = osobnik->Gnext;
17          suma += tmp;
18      }
19      if (max == 0) return 0;
20      return suma / dlugosc / max;
21  }
```

Funkcja obliczająca wartość przystosowania podanego osobnika.

4.3.11 klonuj

Parametry:
-param populacja populacja w której dojedzie do klonowania
-param osobnik osobnik który zostanie sklonowany

```
1  void klonuj(Populacja &populacja,   Osobniki *&osobnik)
2  {
3      Geny *wzor = osobnik->Gglowa;
4      Geny *klon(nullptr);
5      while (wzor)
6      {
7          dodaj_nakon(klon, wzor->chromosom);
8          wzor = wzor->Gnext;
9      }
10
11     populacja.Oglowa = new Osobniki{ osobnik->Odlugosc,
           osobnik->przystosowanie, klon, populacja.Oglowa };
12     populacja.Ldlugosc++;
13 }
```

Funkcja klonująca i dodająca osobnika na początek listy osobników.

4.3.12 krzyzowanie

Parametry:
-osobnikA pierwszy osobnik na którym będzie dokonywane krzyżowanie
-osobnikB drugi osobnik do krzyżowania

```
1  void krzyzowanie( Osobniki *&osobnikA, Osobniki *&
       osobnikB)
2  {
3
4      Geny*Atmp = osobnikA->Gglowa, *Btmp = osobnikB->
           Gglowa, *Zamiana;
5
6      //-----------------------------------losowanie miejsca
           pekniecia-----------------------------------
7          int dlugoscA_pekniete = rand() % (osobnikA->
               Odlugosc -1); //Dlugosc zawsze wieksza od 1
8
9      for (int i =0; i <dlugoscA_pekniete; i++)
10     {
11         Atmp = Atmp->Gnext; //przechodznie po liscie
12     }
13         int dlugoscB_pekniete = rand() % (osobnikB->
               Odlugosc-1);
14
15     for (int i = 0; i < dlugoscB_pekniete; i++)
16     {
17         Btmp = Btmp->Gnext;
18     }
19
20     //-----------------------------------obliczanie dlugosci
           genomu
21     int dlugosctmp = osobnikA->Odlugosc;
22
23     osobnikA->Odlugosc = dlugoscA_pekniete+osobnikB->
           Odlugosc-dlugoscB_pekniete;
24     osobnikB->Odlugosc = dlugoscB_pekniete+dlugosctmp-
           dlugoscA_pekniete;
25
26     //-----------------------------------przepinanie listy
27     Zamiana = Atmp->Gnext;
28     Atmp->Gnext = Btmp->Gnext;
```

```
29      Btmp->Gnext = Zamiana;
30
31      osobnikA->przystosowanie = f(osobnikA->Gglowa);
32      osobnikB->przystosowanie = f(osobnikB->Gglowa);
33  }
```

Funkcja miesza 'chromosomy' z 'genów' OsobnikA i OsobnikaB. Po wymieszaniu chromosomów oblicza na nowo wartość przystosowania za pomocą funkcji f(*str.*12)

4.3.13 losowanie_par

Parametry:
-zakres zakres losowanych liczb
Funkcja zwraca strukturę dwie-liczby.
Zwracane wartości są posortowane rosnąco

```
1  dwie_liczby losowanie_par(const int &zakres)
2  {
3      int a, b;
4      a = rand() % zakres;
5
6      do
7      {
8      b = rand() % zakres;
9      } while (a == b);
10
11     // sortowanie
12     if (a < b)
13        return { a,b };
14     else
15        return { b,a };
16 }
```

Funkcja losuje dwie różne liczby z podanego zakresu (zakres musi być większy od 2)

4.3.14 pob_param

Parametry:
-ile ilość elementów w tablicy parametrów
- params[] tablica z parametrami

- parametry wszystkie parametry Funkcja zwraca kody błędów, tj. 0-brak błędów, 1-złe parametry parametrów

```
1  bool pob_param(Parametry &parametry,  int ile, char *
       params[])
2  {
3      double suma = 0;//suma kontrolna
4
5      if (ile % 2 == 0||ile<13) //spr czy wszystkie param
6          return true;
7
8      for (int i = 1; i < ile; i += 2)
9      {
10         std::string tmp = params[i];
11         if (tmp == "-k")
12         {
13             if (czy_calkowita(params[i + 1]))
14                 return true;
15             suma += 0.01;
16             parametry.k = atoi(params[i + 1]);
17         }
18         else {
19
20             if (tmp == "-p")
21             {
22                 if (czy_calkowita(params[i + 1]))
23                 return true;
24                 suma += 0.1;
25                 parametry.p = atoi(params[i + 1]);
26             }
27             else {
28
29                 if (tmp == "-w")
30                 {
31                     if (czy_zmienno(params[i + 1])|| strtod(
                           params[i + 1], nullptr) > 1)
32                         return true;
33                     suma += 1;
34
35                         parametry.w = strtod(params[i + 1],
                               nullptr);
```

```
36
37              }
38              else {
39
40                  if (tmp == "-r")
41                  {
42
43                      if (czy_zmienno(params[i + 1]) ||
                            strtod(params[i + 1], nullptr) >
                            1)
44                          return true;
45                      suma += 10;
46                      parametry.r = strtod(params[i + 1],
                            nullptr);
47
48                  }
49                  else {
50
51                      if (tmp == "-i")
52                      {
53                          if (czy_plik(params[i + 1]))
54                              return true;
55                          suma += 100;
56                          parametry.plikin = params[i + 1];
57                      }
58                      else {
59
60                          if (tmp == "-o")
61                          {
62                              suma += 1000;
63                              if (czy_plik(params[i + 1]))
64                                  return true;
65                              parametry.plikout = params[i +
                                    1];
66                          }
67                          else {
68                              return true;
69                          }
70                      }
71                  }
72              }
```

```
73              }
74          }
75      }
76      if (suma == 1111.11 && (parametry.w < parametry.r))
77          return false; //poprawne dane
78
79          return true; //blad
80  }
```

Funkcja pobiera parametry i je weryfikuje, poprzez szereg operacji oraz kilka funkcji pomocniczych np.,
-czy calkowita1(*str*.9)
-czy zmienno (*str*.10)
Na początku i-ty element tablicy params jest porównywany ze wzorcami(przez dużą ilość if/else), jeżeli wyżej wspomniany element pasuje do jednego ze wzorców to nastąpi weryfikowanie i+1-tego elementu który jest jednym z parametrów. W każdej głównej części głównego if'a(2 od początku funkcji) do zmiennej o nazwie suma jest dodawana odpowiednia potęga 10-tki. Wspomniana operacja ma celu weryfikacje tego, czy każdy z przełączników wystąpił tylko raz.

4.3.15 pob_pop

Parametry:
-populacja
- nazwa_pliku
Funkcja zwraca kody błędów, tj. 0-pobieranie zakończone pomyślnie, 1-nieprawidłowe dane, 2-błąd przy otwieraniu pliku.

```
1  int pob_pop(Populacja &populacja, std::string
       nazwa_pliku)
2  {
3      Osobniki *osobniki = nullptr; //glowa populacji
4
5      string a = nazwa_pliku;
6      ifstream plik(nazwa_pliku);
7      if (plik)
8      {
9
10         int ilosc_osobnikow = 0;
11         int  dlugosc_genow = 0;
12         string liniatmp = "";
```

```
13
14      int b = 0;
15      while (getline(plik, liniatmp))
16      {
17          if (czy_puste(liniatmp)) continue; //
                sprawdzanie czy linia jest pusta
18          istringstream linia(liniatmp);
19          dlugosc_genow = 0;
20          Geny *chromosomy(nullptr);
21
22
23          while (linia >> b)
24          {
25
26              dodaj_nakon(chromosomy, b);
27              dlugosc_genow++;
28          }
29
30          if (czy_calkowita(liniatmp) || dlugosc_genow <
                2)
31          {
32              usun(chromosomy);
33              usun(osobniki);
34              populacja = { 0,nullptr };
35
36              return 1; // nieprawidlowe dane
37
38          }
39          else
40          {
41              dodaj_nakon(osobniki, chromosomy,
                    dlugosc_genow);
42              ilosc_osobnikow++;
43          }
44      }
45
46      plik.close();
47      populacja = { ilosc_osobnikow,osobniki };
48      return 0; // poprawne dane
49  }
50
```

```
51     return 2; // Blad przy otwierania pliku
52 }
```

Funkcja pobiera populacje z podanego pliku. W trakcie pobierania weryfikuje poprawność danych, poprzez różne operacje oraz funkcje czy_puste(*str*.10) oraz czy_calkowita2 (*str*.9) Jeżeli funkcja zakończy się przez błąd danych, wartości zmiennej populacja wyzerują się, a wcześniej zarezerwowane miejsca w pamięci zostaną zwolnione poprzez przeciążone funkcje usun(*str*.21, 22). Dodawanie osobników do populacji odbywa się poprzez dwie przeciążone funkcje dodaj_nakon(*str*.11, 12). Gdy funkcja zakończy działanie pomyślnie, zmienna populacja otrzyma poprzez referencje zaktualizowane dane.

4.3.16 przesuwanie

param populacja populacja po której będziemy się przemieszczać param a_b zmienna przechowująca dwie liczby, które odpowiadają za to o ile miejsc w liście funkcja ma się przesuwać return zwraca strukturę dwa_wsk przechowującą dwa wskaźniki na element z listy osobników

```
1 dwa_wsk przesuwanie(const Populacja &populacja,const
    dwie_liczby &a_b)
2 {
3     Osobniki *osobnikAtmp=populacja.Oglowa, *osobnikBtmp
        ;
4
5     for (int  i = 0; i < a_b.a; i++)
6     {
7         osobnikAtmp = osobnikAtmp->Onext;
8     }
9     osobnikBtmp = osobnikAtmp;
10    for (int i = 0; i < a_b.b-a_b.a; i++)
11    {
12        osobnikBtmp = osobnikBtmp->Onext;
13    }
14
15    return {osobnikAtmp,osobnikBtmp };
16 }
```

Funkcja przemieszczajaca sie po populacji.

4.3.17 spr_przystosowania

Parametry:
-populacja populacja na której operuje funkcja
- wymieranie wartość progu wymierania
- rozmnazanie wartość progu rozmnażania

```
1  void spr_przystosowania(Populacja &populacja,const
       double &wymieranie,const double &rozmnazanie)
2  {
3
4      Osobniki *tmp = populacja.Oglowa,
5               *nastepny = tmp,
6               *poprzedni=nullptr;
7
8          while(tmp)
9          {
10             nastepny = tmp->Onext;
11
12             if (tmp->przystosowanie < wymieranie)
13                 usunosobnik(populacja,tmp,poprzedni);
14
15             else
16             {
17                 if (tmp->przystosowanie > rozmnazanie)
18                     klonuj(populacja, tmp);
19
20                 poprzedni = tmp; // nie zostal usuniety
                       wiec jest brany pod uwage
21             }
22
23             tmp = nastepny;
24         }
25 }
```

Funkcja sprawdza czy osobnik powinien zostać usunięty, sklonowany.
Klonowanie odbywa się przez funkcje klonuj(*str*.13)
Usuwanie odbywa się poprzez funkcje usunosobnik(*str*.22)

4.3.18 usun1

Parametry:
-phead wskaznik na pierwszy element listy

```
1 void usun(Geny *&phead)
2 {
3     while (phead)
4     {
5         Geny * ptmp = phead->Gnext;
6         delete phead;
7         phead = ptmp;
8     }
9 }
```

Funkcja usuwająca listę z chromosomami (genom).

4.3.19 usun2

Parametry:
-phead wskaźnik na pierwszy element listy osobników

```
1 void usun(Osobniki *&phead)
2 {
3     while (phead)
4     {
5         Osobniki *ptmp = phead->Onext;
6         usun(phead->Gglowa);
7         delete phead;
8         phead = ptmp;
9     }
10 }
```

Funkcja usuwa listę z osobnikami.

4.3.20 usunosobnik

Parametry:
-populacja zmienna z której będzie usuwany osobnik
- slaby osobnik wskaźnik na osobnika który będzie usuwany
-poprzedni wskaźnik na poprzedniego osobnika listy z osobnikami jeżeli nie istnieje to wynosi nullptr.

```
1 void usunosobnik(Populacja &populacja, Osobniki *&
    slaby_osobnik, Osobniki *&poprzedni)
2 {
3     if (slaby_osobnik)
4     {
5
6         if (populacja.Oglowa == slaby_osobnik)
7             populacja.Oglowa = slaby_osobnik->Onext;
8         else
9             poprzedni->Onext = slaby_osobnik->Onext;
10
11        populacja.Ldlugosc--;
12
13        usun(slaby_osobnik->Gglowa);
14
15        delete slaby_osobnik;
16    }
17 }
```

Funkcja usuwa pojedynczy element listy z osobnikami ze struktury przechowującej całą populacje.

4.3.21 zapisz

Parametry:
-populacja ktora bedzie zapisywana do pliku
-nazwapliku string z nazwa pliku do zapisu
Funkcja zwraca kody błędu 0-pomyślny zapis 1-błąd pliku zapisu

```
1 bool zapisz(const Populacja &populacja, const string &
    nazwapliku)
2 {
3     ofstream plik(nazwapliku);
4     if (plik)
5     {
6         Osobniki *osobniki = populacja.Oglowa;
7
8
9         while (osobniki)
10        {
11            Geny*geny = osobniki->Gglowa;
12            {
```

```
13                while (geny)
14                {
15                    plik << geny->chromosom << ' ';
16
17                    geny = geny->Gnext;
18                }
19            }
20            osobniki = osobniki->Onext;
21            plik << endl;
22        }
23        plik.close();
24        return false;
25    }
26    else
27        return true;
28 }
```

Funkcja zapisująca listę osobników do pliku o podanej nazwie.

5 Testowanie

Program został przetestowany na różnego rodzaju danych. Puste linie w pliku wejściowym są ignorowane. Program przestaje działać dla za dużej ilości pokoleń, ponieważ system przydziela programowi ograniczoną ilość pamięci, a ta może nie wystarczyć. Wyznaczenie maksymalnej wielości populacji względem ilości pokoleń(lub odwrotnie) jest rzeczą niemożliwą do wykonania ponieważ cały program w głównej mierze opiera się na losowości(pseudolosowości).

6 Wnioski

Program „DARWIN" jest programem złożonym i w zależności od przyjętych złożeń mógłby znacząco się różnić. Przy założeniach przyjętych przy realizacji projektu, program nie stanowił wielkiego wyzwania jeżeli chodzi o trudność, jednakże pisanie programu opartego głównie o dynamiczne struktury danych, było dla mnie rzeczą nową, a co za tym idzie momentami bardzo czasochłonną. Popełnianie prze zemnie błędy wydłużały czas realizacji projektu, ale dużo mogłem się z nich wynieść.