

Dokumentacja Techniczna Do Programu Porównującego Oceny Wartości Hodowlanej

Program porównuje ocenę dwie oceny

Dokumentacja Techniczna Do Programu Porównującego Oceny Wartości Hodowlanej

Program porównuje ocenę dwie oceny

Sporządził Karol Brzozowski

Sporządził Karol Brzozowski

1. Opis programu

Celem programu, jest porównanie dwóch ocen wartości hodowlanej. Program porównuje wszystkie dane od danych ID, po dane produkcyjne, oraz indeksy, z powodów wydajnościowych program uruchamiany jest na serwerze linux. Program po uruchomieniu prosi użytkownika o podanie trzycyfrowej wartości oznaczającej numer oceny, następnie prosi o przedział wiekowy dla której ma dokonać sprawdzenia(opcja dodatkowa), po czym przystępuje do działania:

- program tworzy listę osobników występujących zarówno w jednej jak i drugiej ocenie ponieważ w poszczególnych ocenach przybywa lub ubywa osobników
- utworzenie nowych plików, z takimi samymi danymi, w takiej samej kolejności, numery z poprzedniej oceny = z nr z bieżącej + sprawdzenie roczników
- sprawdzenie poprawności danych dla danych ID i cech produkcyjnych
- zapisanie różnic w plikach txt.

Program ma za zadanie przyspieszyć i usprawnić konwencjonalną ocenę wartości hodowlanej bydła mlecznego.

Program korzysta z następujących plików wejściowych:

- Pliki z Oceną poprzednią i bieżącą, nazwa w formie `pisz_iz_XXX.txt` gdzie XXX- oznacza numer oceny np. 193(trzecia ocena z roku 2019) lub 201(pierwsza ocena z 2020 roku)
- `Pozycje.txt`
- `Nazwy.txt`

Plik `pozycje.txt` jest jednokolumnowym ciągiem numerycznym zawierającym informacje o ilości znaków dla danej cechy, ile dana cecha zawiera znaków np.:

```
16    // numer_oryg 16 znaków
12    // numer krajowy 12 znaków...
14
8
2
16
12
14
2
16
14
2
16
14
```

Nazwy.txt – zawiera nazwy wszystkich kolumn oddzielonych średnikiem.

Pliki wyjściowe :

-Raport840USA – Informuje o numerach osobników występujących w jednym pliku(z oceną poprzednią) z numerem rozpoczynającym się od 840, a w drugim pliku(z oceną bieżącą) rozpoczynającym się od USA, i na przemian. Pierwsza kolumna ocena poprzednia, druga ocena bieżąca.

-RaportBiezace840USA – Informuję o nieprawidłowościach związanych z wystąpieniem tego samego numeru osobnika(w pliku z oceną bieżącą) pod dwoma postaciami, w pierwszym przypadku z przedrostkiem 840 w drugim z USA.

-RaportPoprzednie840USA - Informuję o nieprawidłowościach związanych z wystąpieniem tego samego numeru osobnika(w pliku z oceną poprzednią) pod dwoma postaciami, w pierwszym przypadku z przedrostkiem 840 w drugim z USA.

Nazwa - **RaportNazwa**

Numer w Polsce - **RaportNumerPolski**

Data urodzenia - **RaportDataUr**

Rasa/odmiana(HO,RW,PC,JE) - **RaportRasaOdmiana**

Numer międzynarodowy ojca - **RaportNumerMiedzyNarOjca**

Nazwa Ojca – **RaportNazwaOjca**

Numer ojca w Polsce – **RaportNumerOjcaWPolsce**

Rasa/Odmiana(HO,RW,...) Ojca- **RaportRasaOdmianaOjca**

Status Buhaja w Polsce(AI/PK/NN/NU) – **RaportStatusBuhWPolsce**

Raport- Informuję o nieprawidłowościach , struktura pliku kolumny:

1. – nr w pliku z oceną poprzednią.
2. – nr w pliku z oceną bieżącą.
- 3.– dane w pliku z oceną poprzednią.
- 4.- separator '<===>'.
- 5.- dane w pliku z oceną bieżącą.
6. – informacja o nieprawidłowości.

RaportNoweOsobniki – Przechowuję numery osobników które występują w pliku z oceną bieżącą, natomiast nie występują w pliku z oceną poprzednią.

RaportStareOsobniki – Przechowuje numery osobników które występują w pliku z oceną poprzednią, natomiast nie występują w pliku z oceną bieżącą.

pisz_iz_172.tstN – Przechowuję tylko te dane osobników które występują zarówno w pliku z oceną poprzednią jak i bieżącą

pisz_iz_173.tstN - Przechowuję tylko te dane osobników które występują zarówno w pliku z oceną poprzednią jak i bieżącą

Raport.txt - Statystyki

Oficjalna ocena mleczna - **RaportMleko.csv**

Oficjalna ocena pokroju - **RaportPokroj.csv**

Ocena płodności - **RaportPlodnosc.csv**

Dystocja córek buhaja - **RaportDystocjaCB.csv**

Dystocja bezpośrednia - **RaportDystocjaBezp.csv**

Przeżywalność - **RaportPrzezywalnosc.csv**

Indeksy **RaportIndeksy.csv**

Kolumny:

1 – Nr ocena poprzednia

2 – Nr ocena bieżąca

3 – Oceniony w kraju (1- tak, 0 – nie)

4- typ oceny ocena poprzednia

5- typ oceny ocena bieżąca

6 – wartość ocena poprzednia

7 – wartość ocena bieżąca

8 – Cecha produkcyjna

9- Pod cecha (np. typ oceny, liczba obór, liczba córek...)

10 – informacja o tendencji względem oceny poprzedniej (wzrost/spadek)

11 – Wartość zmiany w procentach

12 – Informacja.

Średnie dla poszczególnych roczników dwie wersje plików, różny sposób wyświetlania.

RaportSrednieDlaRocznikow.csv

RaportySrednieDlaRocznikowII.csv

2. Mechanizm działania programu

-Pierwszym etapem programu zaraz po uruchomieniu jest ustawienie zakresów, od do... pobieranych znaków z wiersza wczytanego z pliku.

- Następnie program prosi o podanie numeru oceny poprzedniej i bieżącej w formacie, trzy cyfrowym np. 193 201. 193 oznacza rok 2019 cyfra 3 oznacza trzecią ocenę. Po wpisaniu program poprosi o podanie przedziału rocznikowego.

- Program przystępuje do obliczenia długości wiersza, jak i ilości wierszy w pliku i generuje raport z tą informacją

- Wczytanie do tabeli osobników z poprzedniej oceny
- Wczytanie do tabeli osobników z poprzedniej oceny.
- Sprawdzenie występowania osobników z oceny bieżącej w poprzedniej.
- Sprawdzenie występowania osobnika z oceny poprzedniej w bieżącej.
- Sprawdzenie występowania tych samych osobników 840/USA w ocenie poprzedniej
- Sprawdzenie występowania tych samych osobników 840/USA w ocenie bieżącej
- Utworzenie nowych plików, z takimi samymi danymi, w takiej samej kolejności, numery z poprzedniej oceny = z nr z bieżącej + sprawdzenie roczników
- Sprawdzanie poprawności danych.

2.1. Szczegółowe omówienie konstrukcji programu

Wszystkie wytworzone i użyte funkcje i klasy w programie znajdują się w pliku z funkcją główną oraz w pliku z rozszerzeniem *.h

Omówienie funkcji głównej wraz z funkcjami.

```
int main()
{
    int k;
    //cin>>k;
```

Opis dla cechy produkcyjnej Mleko, Pozostałe Pokrój, Płodność indeksy bazują na tych samych funkcjach i obiektach.

```
MlekoBiez.UstawZakres(213);           // Dla oceny bieżącej
MlekoBiez.UstawDane(ddane);
//MlekoBiez.wyswietl();
MlekoPop.UstawZakres(213);           //Dla oceny poprzedniej
MlekoPop.UstawDane(ddanePoP);
//MlekoPop.wyswietl();
```

Opis bloku funkcyjnego :

Funkcja **UstawZakres** jest funkcją ustawiającą zakresy od którego znaku do którego znaku w pliku z danymi jest dana cecha.

Funkcja używa trzech podfunkcji:

Dla Cech:

```
void UstawZakres(int d1)
{
    this->Cechy[0].UstawWartoscMleko(213);           //liczba krajów w których buhaj ma ocenę mleczną
    this->Cechy[0].Podindeksy[6].DoZnaku =213+32;     // mleko kg
    this->Cechy[1].UstawWartoscMleko(246);           // tłuszcz kg
    this->Cechy[2].UstawWartoscMleko(278);           // tłuszcz %
    this->Cechy[3].UstawWartoscMleko(310);           // białko kg
    this->Cechy[4].UstawWartoscMleko(342);           // białko %
    this->Cechy[5].UstawWartoscMleko(374);           // komórki somatyczne
};
```

Dla Podcech:

```
inline void UstawWartoscMleko(int Od)
{
    this->OdZnaku =Od;
    Podindeksy[0].UstawKomorka(OdZnaku,OdZnaku);   // typ oceny
    Podindeksy[1].UstawKomorka(OdZnaku+1,OdZnaku+5); //liczba obór
    Podindeksy[2].UstawKomorka(OdZnaku+6,OdZnaku+11); //liczba córek
    Podindeksy[3].UstawKomorka(OdZnaku+12,OdZnaku+17); // liczba córek efektywna
    Podindeksy[4].UstawKomorka(OdZnaku+18,OdZnaku+23); //EDC
    Podindeksy[5].UstawKomorka(OdZnaku+24,OdZnaku+25); //Powtarzalność *100
    Podindeksy[6].UstawKomorka(OdZnaku+26,OdZnaku+31); //wartość hodowlana *100
};
```

Funkcja **UstawKomorka** ustawia zakres od znaku do znaku dla danej pod cechy

```
void UstawKomorka(int PZ,int DZ)
{
    this->PlusZnakow =PZ;           //od znaku
    this->DoZnaku =DZ;             //do znaku
};
```


Funkcja *UstawDane* jest funkcją ustawiającą dane wycina dane od znaku do znaku z pliku txt .

```
inline void UstawDane(string &dane)           // Dane z pliku
{
    for(int i=0; i<6; i++)
    {
        this->Cechy[i].wycinanie(dane);        // Funkcja wycinająca dane z pliku txt
    }
}
```

Dla pozostałych cech opis j.w

```
//cout<<"=====Pokroj"<<endl;
//cout<<"=====Plodnosc"<<endl;

//cout<<"=====DystocjaCorekBuhaja"<<endl;
//cout<<"=====DystocjaBezposrednia"<<endl;

//cout<<"=====Przezywalnosc"<<endl;

//cout<<"=====Indeksy"<<endl;
```

```
int z;
//cin>>z;
```

Deklaracja zmiennych plikowych typu *fstream*:

```
fstream plk1, plk2, plkN1, plkN2, plkNowe, plkStar, plkRaportSzcz, plkRaport, plkNazwa, plkNumerPolski, plkDataUr,
plkRasaOdmiana, plkNumerMOjca, plkNazwaOjca, plkNumerOjcaWP, plkRasaOjca, plkStatusBuhWPL, plk840USA,
plkStar840USA, plkNowe840USA, plkRaportMleko, plkRaportPokroj, plkRaportPlodnosc, plkRaportDystocjaCB,
plkRaportDystocjaBez, plkRaportPrzezywalnosc, plkRaportIndeksy, plkRaportSrednie, plkRaportSrednieRocznikami,
plkStrukturaPliku, plkNazwyKolumn;
```

Deklaracja zmiennych typu string:

```
string zm1, zm2, zmw1, zmw2;
```

Deklaracja zmiennych typu bool:

```
bool z1=1, z2=1;
```

Deklaracja zmiennych typu int:

```
int licznik=0, n=0, liczStar=0, liczNowe=0, StaraNazwa=0, NowaNazwa=0, RzoneNazwa=0, StaryNumer=0,
NowyNumer=0, RzoneNumer=0, NowaDataUr=0, StaraDataUr=0, RzoneDatyUr=0,
NowaRasaOdmiana=0, StaraRasaOdmiana=0, RzonaRasaOdmiana=0, NowyNumerMOjca=0, StaryNumerMOjca=0,
RoznyNumerMojca=0, NowaNazwaOjca=0, StaraNazwaOjca=0, RznaNazwaOjca=0, NowyNumerOjcaWP=0,
StaryNumerOjcaWP=0, RoznyNumerOjcaWP=0, NowaRasaOjca=0, StaraRasaOjca=0, RznaRasaOjca=0,
NowyStatusBuhWPL=0, StaryStatusBuhWPL=0, RoznyStatusBuhWPL=0, Liczba840USA=0,
liczStarUSA=0, liczStar840=0, liczNoweUSA=0, liczNowe840=0, liczStar840USA=0, liczNowe840USA=0;
```

Deklaracja zmiennych typu year dla roczników:

```
year RocznikiP[100], RocznikiB[100];
```

Deklaracja zmiennych dynamicznych typu char:

```

char *buf1;
char *buf2;
char **tabStar = new char*[20000];
char **tabNowe = new char*[20000];
char **tabStar840 = new char*[30000];
char **tabStarUSA = new char*[50000];
char **tabNowe840 = new char*[30000];
char **tabNoweUSA = new char*[50000];

```

Wstawka testowa programu klasy StructPliku. !!!!!!!!!!!!!

```
StructPliku StrPliku;
```

```

plkStrukturaPliku.open("Pozycje.txt", ios::in);
plkNazwyKolumn.open("Nazwy.txt", ios::in);

```

```

StrPliku.Ustaw(plkNazwyKolumn, plkStrukturaPliku);
StrPliku.wyswietl();
int NaszaKlasa;
cin >> NaszaKlasa;

```

Koniec programu testowego !!!!!!!!!!!!!!!!!!!!!!!!!!!!!

Pobieranie danych z klawiatury wraz z komunikatem wyświetlanym na ekranie:

```

string NrStary = "163", NrNowy = "171";
cout << "Podaj numer oceny poprzedniej i biezacej w postaci np. 163,171... w kolejnosci 1.ocena poprzednia, 2. ocena biezaca. UWAGA !!! zbiory musza byc posortowane !!!" << endl;
cin >> NrStary >> NrNowy;

```

```

int lata1, lata2;
cout << "Podaj przedzial roczników od do " << endl;
cin >> lata1 >> lata2;

```

Tworzenie zmiennych string wraz z automatycznym dodawaniem nazw dla nowo utworzonych plików:

```

string OcenaPoprzednia = "pisz_iz_" + NrStary + ".tst";
string OcenaBizaca = "pisz_iz_" + NrNowy + ".tst";
string OcenaPoprzedniaN = OcenaPoprzednia + "N";
string OcenaBizacaN = OcenaBizaca + "N";
string Raprot = "Raport.txt", RaportSzc = "RaportSzc.txt";
int ilosc840USA1 = 0, ilosc840USA2 = 0;

```

Otwarcie zmiennych plikowych dla istniejących plików :

```

plk1.open(OcenaPoprzednia.c_str(), ios::in);
plk2.open(OcenaBizaca.c_str(), ios::in);

```

Otwarcie zmiennych plikowych dla plików raportowych, jeżeli plik nie istnieje jest tworzony:

```

plkN1.open(OcenaPoprzedniaN.c_str(), ios::out);
plkN2.open(OcenaBizacaN.c_str(), ios::out);
plkNowe.open("RaportNoweOsobniki", ios::out);

```

```

plkStar.open("RaportStareOsobniki", ios::out);
plkRaportSzcz.open(RaportSzcz.c_str(), ios::out);
plkRaport.open(Raport.c_str(), ios::out);
plkNazwa.open("RaportNazwa", ios::out);
plkNumerPolski.open("RaportNumerPolski", ios::out);
plkDataUr.open("RaportDataUr", ios::out);
plkRasaOdmiana.open("RaportRasaOdmiana", ios::out);
plkNumerMOjca.open("RaportNumerMiedzyNarOjca", ios::out);
plkNazwaOjca.open("RaportNazwaOjca", ios::out);
plkNumerOjcaWP.open("RaportNumerOjcaWPolsce", ios::out);
plkRasaOjca.open("RaportRasaOdmianaOjca", ios::out);
plkStatusBuhWPL.open("RaportStatusBuhWPolsce", ios::out);
plk840USA.open("Raport840USA", ios::out);
plkStar840USA.open("RaportPoprzednie840USA", ios::out);
plkNowe840USA.open("RaportBiezace840USA", ios::out);

plkRaportMleko.open("RaportMleko.csv", ios::out);
plkRaportPokroj.open("RaportPokroj.csv", ios::out);
plkRaportPlodnosc.open("RaportPlodnosc.csv", ios::out);
plkRaportDystocjaCB.open("RaportDystocjaCB.csv", ios::out);
plkRaportDystocjaBezp.open("RaportDystocjaBezp.csv", ios::out);
plkRaportPrzezywalnosc.open("RaportPrzezywalnosc.csv", ios::out);
plkRaportIndeksy.open("RaportIndeksy.csv", ios::out);
plkRaportSrednie.open("RaportSrednieDlaRocznikow.csv", ios::out);
plkRaportSrednieRocznikami.open("RaportSrednieDlaRocznikowII.csv", ios::out);

```

Sprawdzenie czy pliki wejściowe istnieją jeśli tak to program przystępuje do dalszego etapu, jeśli nie zostanie zakończone działanie programu:

```

if(plk1.good()==true && plk2.good()==true)
{
    string zzm2;
    plk1.seekg(0, ios::end);
    int wiersze1=plk1.tellg();           // Sprawdzenie wielkości pliku dla pliku z oceną poprzednią
    plk1.seekg(0, ios::beg);

    plk2.seekg(0, ios::end);
    int wiersze2=plk2.tellg();           // Sprawdzenie wielkości pliku dla pliku z oceną bieżącą
    plk2.seekg(0, ios::beg);

    getline(plk1, zm1);
    getline(plk2, zzm2);                 // Pobranie wiersza dla plików w celu określanie liczby znaków
    plk1.close();
    plk2.close();

    plk1.open(OcenaPoprzednia.c_str(), ios::in);
    plk2.open(OcenaBizaca.c_str(), ios::in);

    int siz11 = zm1.length();
    int siz22 = zzm2.length();
    int Lwierszy1 = wiersze1/(siz11+1);   // Obliczenie liczby wierszy jak i liczby znaków w wierszu.
    int Lwierszy2 = wiersze2/(siz22+1);
}

```

Wygenerowanie komunikatu i raportu odnośnie liczby wierszy liczby znaków w wierszu dla plików z oceną poprzednią i bieżącą:

```

cout<<"Liczba wierszy w "<<OcenaPoprzednia.c_str()<<" = "<<Lwierszy1<<" znakow = "<<siz11<<endl;
cout<<"Liczba wierszy w "<<OcenaBizaca.c_str()<<" = "<<Lwierszy2<<endl;
plkRaport<<"Liczba wierszy w pliku "<<OcenaPoprzednia<<" = "<<Lwierszy1<<" ,liczba znakow w wierszu =
"<<siz11<<endl;
plkRaport<<"Liczba wierszy w pliku "<<OcenaBizaca<<" = "<<Lwierszy2<<" ,liczba znakow w wierszu =
"<<siz22<<endl;

```

```

char **tab1 = new char*[Lwierszy1];

```

```
char **tab2 = new char*[Lwierszy2];
```

```
vector<char*> myvector;
```

```
int eee;
```

Wczytywanie do tabeli osobników z poprzedniej oceny

```
WczytanieDoTabeli(tab1,Lwierszy1,plk1,tabStar840, tabStarUSA, liczStar840, liczStarUSA,siz11+1 );
cin>>eee;
cout<<"etap 1.2 wczytywanie do tabeli osobników z bieżącej oceny"<<endl;
WczytanieDoTabeli(tab2,Lwierszy2,plk2, tabNowe840, tabNoweUSA, liczNowe840, liczNoweUSA,siz22+1 );
cin>> eee;
```

Omówienie Funkcji **WczytanieDoTabeli()**: Funkcja tworzy tabelę z numerami których prefiks jest 840 i USA, jest to potrzebne do sprawdzenia czy nie nastąpiła pomyłka, czyli sprawdzenie czy dany osobnik nie występuje dwa razy z tym samym numerem a różnym oznaczeniem kraju:

```
inline void WczytanieDoTabeli(char **tab, int Lwierszy, fstream &plk, char **tab840, char **tabUSA, int &licz840, int &liczUSA, int LiczbaZnakow)
{
    int n=0;
    for(int i=0; i<Lwierszy; i++)
    {
        tab[i] = new char[17];
        plk.seekg(n,ios::beg);
        plk.read(tab[i],17);
        tab[i][16] = '\0';
        cout<<tab[i]<<endl;
    }
}
```

Warunek jeżeli prefiks numeru = USA to wczytaj do tabeli tabUSA ten numer

```
if((tab[i][0]=='U')&&(tab[i][1]=='S')&&(tab[i][2]=='A'))
{
    tabUSA[liczUSA] = tab[i];
    liczUSA++;
}
}
```

Warunek jeżeli prefiks numeru = 840 to wczytaj do tabeli tab840 ten numer

```
if((tab[i][0]=='8')&&(tab[i][1]=='4')&&(tab[i][2]=='0'))
{
    tab840[licz840] = tab[i];
    licz840++;
}
n = n+LiczbaZnakow;
}
}
```

porównanie danych poprzednich z bieżącymi

```
SprWystepowaniaOsobnika(tabStar, tab1,tab2,Lwierszy1,Lwierszy2,plkStar,liczStar);

cout<<"etap 2.2 porownanie danych biezacych z poprzednimi"<<endl;
SprWystepowaniaOsobnika(tabNowe, tab2,tab1,Lwierszy2,Lwierszy1,plkNowe,liczNowe);

cout<<"etap 2.3 sprawdzenie wystepowania tych samych osobnikow 840/USA w ocenie poprzedniej"<<
OcenaPoprzednia<<endl;
SprWystepowaniaOsobnika830USA(tabStar840,tabStarUSA,liczStar840,liczStarUSA,plkStar840USA,
liczStar840USA);

cout<<"etap 2.4 sprawdzenie wystepowania tych samych osobnikow 840/USA w ocenie biezacej"<<
```

```
OcenaBizaca<<endl;
```

```
SprWystepowaniaOsobnika830USA(tabNowe840,tabNoweUSA,liczNowe840,liczNowe840USA,plkNowe840USA,  
liczNowe840USA);
```

Omówienie funkcji ***SprWystepowaniaOsobnika()***, Funkcja sprawdza czy dany osobnik np. z oceny bieżącej występuje w ocenie poprzedniej, jeśli nie to zostanie zapisany do pliku stare.txt

```
inline void SprWystepowaniaOsobnika(char **tabSN, char **tab1, char **tab2, int LwierszyI, int LwierszyJ, fstream  
&plk, int &licz)  
{  
    fstream ddd;  
    ddd.open("NowaG", ios::out);  
    int param=0;  
    for(int i=0; i<LwierszyI; i++)  
    {  
        for(int j=param; j<LwierszyJ; j++)  
        {  
            //cout<<tab1[i]<<"====="<<tab2[j]<<endl;  
            if(sprawdz(tab1[i], tab2[j])==1)           // Sprawdzenie czy nr. osobika z poprzedniej oceny występuje w Bieżacej  
            {  
                param = j;  
                break;  
            }  
  
            if(j==LwierszyJ-1)                        // Jeśli nie ma osobnika z poprzedniej oceny w bieżacej  
            {  
                plk<<tab1[i]<<endl;                    // zapis do pliku osobników tzw "starych" których nie ma w bieżacej ocenie.  
                tabSN[licz]=tab1[i];  
                licz++;  
            }  
        }  
    }  
    tabSN[licz] = tab1[0];                            // Uwaga wyzerowanie tablicy  
}
```

Omówienie funkcji ***SprWystepowaniaOsobnika830USA()*** Funkcja sprawdza czy osobnik z różnym prefiksem 840 lub USA występuje dwa razy raz z prefiksem 840 lub USA ale z tym samym numerem, funkcja sprawdza duble, wynikające z błędu kodu kraju.

```
inline void SprWystepowaniaOsobnika830USA(char **tab830, char **tabUSA, int Lwierszy830, int LwierszyUSA,  
fstream &plk, int &licz)  
{  
    for(int i=0; i<Lwierszy830; i++)  
    {  
        for(int j=0; j<LwierszyUSA; j++)  
        {  
            if(sprawdz(&tab830[i][3], &tabUSA[j][3])==1)           // Jeżeli numer z tabeli z numerami 840 = numer z tabUSA  
            {  
                plk<<tab830[i]<<"====="<<tabUSA[j]<<endl;  
                licz++;  
            }  
        }  
    }  
}
```

Omówienie funkcji ***sprawdz()*** Funkcja sprawdzająca, jeżeli wyrazy są takie same zwraca 1 jeżeli różne 0 funkcja powstała w dwóch typach dla zmiennych typu ***char*** oraz ***string***
Dla zmiennych string:

```
inline bool sprawdz(string p1, string p2)
```

Dla zmiennych typu *char*:

```
inline bool sprawdz(char *p1, char *p2)
```

```

{
    int i=0;
    for(i=0;; i++)
    {
        if(p1[i]=='\0' || p2[i]=='\0') break;
        if(p1[i] != p2[i]) break;
    }
    if(p1[i] == p2[i]) return 1;
    else      return 0;
}

```

Wygenerowanie raportów do plików txt:

plikRaport<<"Liczba osobników z oceny poprzedniej "<<OcenaPoprzednia<<" których numer powtarza się 840/USA = "<<liczStar840USA<<endl;

```
plkRaport<<"Liczba osobników z oceny bieżącej "<<OcenaBieżąca<<" których numer powtarza się 840/USA =  
"<<liczNowe840USA<<endl;
```

```
plkRaport<<"Liczba osobnikow z pliku " <<OcenaPoprzednia<<" nie występujące w pliku " <<OcenaBizaca<<" =
"<<liczStar<<endl;
```

```
plkRaport<<"Liczba osobnikow z pliku "<<OcenaBizaca<<" nie występujące w pliku "<<OcenaPoprzednia<<" =
"<<liczNowe<<endl;
```

Wygenerowanie raportu do pliku txt osobników ze zmienionym symbolem(prefiksem) kraju:

cout<<" Osobniki ze zmienionym symbolem kraju 840 USA w ocenie poprzedniej i bieżącej"<<endl;

```
for(int i=0; i<liczStar; i++)
```

```

{
    for(int j=0; j<liczNowe; j++)
    {

```

$$if((Sprawdz840USA(tabStar[i], tabNowe[j]) == 1))$$

```

    plk840USA<<tabStar[i]<<"<<<=====>>>"<<tabNowe[j]<<endl;
    cout<<tabStar[i]<<"<<<=====>>>"<<tabNowe[j]<<endl;
    Liczba840USA++;
    break;

```

plkRaport<<"Liczba Osobników ze zmienionym oznaczeniem kraju 840 USA w obydwu ocenach =

```
"<<Liczba840USA<<endl;
```

Utworzenie plików do sprawdzenia ocen, pliki mają takie same dane, takie same osobniki, osobniki które nie znalazły dubla w ocenie poprzedniej, lub bieżącej są nieuwzględniane.

```
cout<<"etap 3 utworzenie nowych plikow, z takimi samymi danymi, w takiej samej kolejnosci, numery z poprzedniej  
oceny == z nr z bieżącej + sprawdzenie roczników"<<endl;
```

```
Zerowanie(RocznikiP);  
Zerowanie(RocznikiB);
```

```
n=0;  
buf1 = new char[siz11+1];           //Utworzenie zmiennej dynamicznej o wielkości siz11+1  
buf2 = new char[siz22+1];  
liczNowe =0;  
liczStar =0;
```

Utworzenie pliku dla osobników z oceny poprzedniej

```
for(int i=0; i<Lwierszy1; i++)  
{  
    plk1.seekg(n,ios::beg);           // Ustawienie wskaźnika odczytu na początek pliku  
    plk1.read(buf1,siz11);           // Odczyt z pliku i zapis do zmiennej buf1 ilości znaków siz11  
    buf1[siz11] = '\0';  
  
    if(sprawdz(tab1[i],tabStar[liczStar])==1)  
    {  
        liczStar++;                   // Oblicza ilość starych osobników  
    }  
    else  
    {  
        plkN1<<buf1<<endl;           // Zapis do pliku  
        string rok = wytnij(buf1,42,45); // Funkcja wycinająca dane opisana niżej  
        int RokInt = NaInt(rok);       // Funkcja zmieniająca string na int opisana niżej  
        ZapiszRok(RocznikiP,RokInt);  // Oblicz i zapisuje do tabeli ilość buhajów dla danego roku  
    }  
    n = n+siz11+1;                   // następna linia ponieważ pobieranie danych z pliku jest blokowe  
}  
  
n=0;
```

Utworzenie plików dla osobników z oceny bieżącej

```
for(int i=0; i<Lwierszy2; i++)  
{  
    plk2.seekg(n,ios::beg);           //j.w  
    plk2.read(buf2,siz22);           //j.w  
    buf2[siz22] = '\0';  
    if(sprawdz(tab2[i],tabNowe[liczNowe])==1) //j.w  
    {  
        liczNowe++;                   //Oblicze ilość starych osobników  
    }  
    else  
    {  
        plkN2<<buf2<<endl;           //j.w  
        string rok = wytnij(buf2,42,45); //j.w  
        int RokInt = NaInt(rok);       //j.w  
        ZapiszRok(RocznikiB,RokInt);  //j.w  
    }  
}
```

```

    }

    n = n+sz22+1;                                //j.w
}

cout<<"Sprawdzenie roczników" <<endl;
plkRaport<<"Wykaz rocznikow"<<endl;

plkRaport<<endl;

```

Funkcje użyte w powyższym bloku funkcyjnym:

Funkcja odpowiada za zerowanie tabeli strukturalnej

```

inline void Zerowanie(year *tab)
{
    for(int i=0; i<100; i++)    // Tabela dla 100 osobników
    {
        tab[i].ile =1;          //liczba osobników w danym roku
        tab[i].rok =0;          // Rok urodzenia danych osobników
    }
}

```

Funkcja odpowiada za wycinanie danych z tablicy typu char:

```

inline string wytnij(char *dane, int p1, int p2) //p1, p2 są zmiennymi od znaku do znaku
{
    string zm;
    for(int i=p1; i<=p2; i++)
    {
        zm += dane[i];
        if(dane[i] == '\0')break;
    }
    return zm;
}

```

Funkcja odpowiada za wycięcia danych z tablicy typu string:

```

inline string wytnij(string dane,int p1, int p2) //j.w
{
    string zm;
    int dl = dane.length();
    for(int i=p1; i<=p2; i++)
    {
        zm += dane[i];
        if(i == dl-1) break;
    }
    return zm;
}

```

Funkcja zmienia ciąg znaków w zmiennej typu string na integer :

```

inline int NaInt(string lb) //parametr funkcji zmienna typu string lb
{
    int cyfra=0, wynik=0;

```



```

char znak;
int zz=1; // zz dla pierwszego przebiegu pętli.
for(int i=lb.length()-1; i>=0; i--) // czytanie ciągu znaków od końca
{
    znak = lb[i]; // przypisanie zmiennej char znak wartości z poszczególnej komórki[i] zmiennej string
    if(znak <48 || znak >57) return 0; // jeśli znak jest z poza przedziału cyfrowego z tablicy ascii return 0
    cyfra = znak - 48; // - 48 ponieważ tyle pozycji w tablicy ascii dzieli wartości znakowe do liczbowych
    wynik = wynik + (cyfra *zz); // * zz gdzie zz= kolejną potęgę liczby 10, w pierwszym przebiegu pętli zz=1
    zz=zz*10; // w kolejnych przebiegach zz * 10, dla jedności, dziesiątek, setek tysięcy....
}
return wynik;
}

```

Funkcja odpowiada za zapisanie roku urodzenia do tablicy typu year.

```

inline int ZapiszRok(year *tab, int rokk)
{
    for(int i=0; i<100; i++)
    {
        if(tab[i].rok == rokk) // jeśli dany rok jest już w tablicy to:
        {
            tab[i].ile++; // do zmiennej liczącej osobniki dodajemy 1, liczenie ile osobników w danym roku.
            break;
        }
        if(tab[i].rok ==0) // jeżeli danego rocznika nie ma w tabeli to:
        {
            tab[i].rok =rokk; // wpisanie danego roku do tablicy
            break;
        }
    }
    return 0;
}

```

Tworzenie zmiennych dynamicznych typu year:

```

cout<<" Sprawdzenie poprawnosci danych"<<endl;
year RokiPop[200],RokiBiez[200];
OdchylenieSt *DSPop = new OdchylenieSt[200];
OdchylenieSt *DSBie = new OdchylenieSt[200];
int LRok=0;

```

Tworzenie zmiennych plikowych fstream:

```

fstream plkDaneID, plkDameMleko, plkDanePokroj, plkDanePlodnosc, plkDaneDystocjaCB, plkDaneDystocjaBez,
plkDanePrzezywalnosc, plkDaneIndeksy, plkTabWynikowa;

```

Otwarcie zmiennych plikowych:

```

plkDaneID.open("RaportDaneID.csv",ios::out);
StrPliku.NazwyKolumn(plkDaneID,0,9);

plkDameMleko.open("RaportDameMleko.csv",ios::out);
StrPliku.NazwyKolumn(plkDameMleko,26,69);

plkDanePokroj.open("RaportDanePokroj.csv",ios::out);
StrPliku.NazwyKolumn(plkDanePokroj,69,202);

```

```

plkDanePlodnosc.open("RaportDanePlodnosc.csv",ios::out);
StrPliku.NazwyKolumn(plkDanePlodnosc,202,226);

plkDaneDystocjaCB.open("RaportDaneDystocjaCB.csv",ios::out);
StrPliku.NazwyKolumn(plkDaneDystocjaCB,226,236);

plkDaneDystocjaBez.open("RaportDaneDystocjaBez.csv",ios::out);
StrPliku.NazwyKolumn(plkDaneDystocjaBez,236,246);

plkDanePrzezywalnosc.open("RaportDanePrzezywalnosc.csv",ios::out);
StrPliku.NazwyKolumn(plkDanePrzezywalnosc,246,249);

plkDaneIndeksy.open("RaportDaneIndeksy.csv",ios::out);
StrPliku.NazwyKolumn(plkDaneIndeksy,249,258);

plkTabWynikowa.open("TablicaWynikowaIS", ios::out);

```

Funkcja nazywa kolumny według ustalonego wzorca zmienna *Nazwa*

inline void NazwyKolumn(fstream &plk, int Od, int Do)

```

{
    if(Od > 0) plk<<Nazwa[0]+"";
    for(int i=Od; i<Do; i++)
    {
        plk<<Nazwa[i]+" Pop. ;"<<Nazwa[i]+" Bie. ;"<<Nazwa[i]+" Spr. ;";
    }
    plk<<endl;
}

```

```

cout<<Lwierszy1-liczStar<<endl; // wyświetlenie liczby wiersz osobników które będą brane do porównań
fstream Dom; // utworzenie zmiennej plikowej
Dom.open("RapDom", ios::out); //otwarcie zmiennej plikowej
//year tab[20];

```

```

for(int i=0; i<200; i++)
{
    RokiPop[i].zerowanie(); // zerowanie zmiennej RokiPop
    RokiBiez[i].zerowanie(); // zerowanie zmiennej RokiBie
}

```

```

plkN1.close(); // Zemkniecie zmiennej
plkN2.close(); // Zemkniecie zmiennej

plkN1.open(OcenaPoprzedniaN.c_str(), ios::in); // Otwarcie zmiennej plikowej
plkN2.open(OcenaBizacaN.c_str(), ios::in); // Otwarcie zmiennej plikowej

```

```

for(int i=0; ; i++)
{
    string dane1, dane2; // utworzenie zmiennych typu string

    getline(plkN1,dane1); // pobranie danych z pliku z oceną Pop
    getline(plkN2,dane2); // pobranie danych z pliku z oceną Bierz

    if(plkN1.eof() == true || plkN2.eof() == true ) break; // jeśli napotkasz na koniec pliku przerwij działanie

    string nr1 = wytnij(dane1,0,15); // wycięcie numeru dla oceny poprzedniej
    string nr2 = wytnij(dane2,0,15); // wycięcie numeru dla oceny bieżącej.
}

```

```

string OcenionyWKraju = wynij(dane2,207,207); // wycięcie danych dla oceniony w kraju
string Rrok = wynij(dane1,42,45); // wycinanie danych dla roku urodzenia
int rok = NaInt(Rrok); // rzutowanie danych typu string na integer
if(rok < 1900 || rok >2025) rok = 1900; // dla roku mniejszego od 1900 i 2025
rok = rok-1900;

```

```

StrPliku.WczytajDane(dane1, dane2);

```

```

StrPliku.Publikuj(plkDaneID,0,9,"ID");
StrPliku.Publikuj(plkDaneMleko,26,69,"Mleko");
StrPliku.Publikuj(plkDanePokroj,69,202,"Pokroj");
StrPliku.Publikuj(plkDanePlodnosc,202,226,"Plodnosc");
StrPliku.Publikuj(plkDaneDystocjaCB,226,236,"DystocjaCB");
StrPliku.Publikuj(plkDaneDystocjaBez,236,246,"DystocjaBB");
StrPliku.Publikuj(plkDanePrzezywalnosc,246,249,"Przezywalnosc");
StrPliku.Publikuj(plkDaneIndeksy,249,258,"Indeksy");

```

```

StrPliku.PublikujTabWynikowa(plkTabWynikowa);
StrPliku.Zerowanie();

```

Metoda użyta w klasie class StructPliku {} Metoda na bieżąco wczytuje dane z pobranych wierszy oceny poprzedniej i bieżącej z pliku, i sprawdza je i zapisuje do TabWynikowaString. parametry i rodzaje zmiennych są określone w pliku SprPlik.h, metoda posiada dwa bloki pierwszy blok z pętlą od 0 do 26 dla cech ID wartości string, natomiast od 26 do 258 dla pozostałych cech które są cechami numerycznymi. Jeżeli różnica w pozostałych cechach wyniesie więcej niż 6% różnica zostanie zapisana do tabeli wynikowej, wartość 6 procent została ustalona.

```

inline void WczytajDane(string OcenaPop, string OcenaBie)
{
    for(int i=0; i<26; i++) // Dla danych ID typu string
    {

        WartoscStringPop[i] = wynij(OcenaPop,OdZnaku[i],DoZnaku[i]);
        ZmienLiteryNaDuze(WartoscStringPop[i]); // Funkcja zmienia litery na duze
        WartoscPopNull[i] = SprawdzPoprawnoscString(WartoscStringPop[i]);

        WartoscStringBie[i] = wynij(OcenaBie,OdZnaku[i],DoZnaku[i]);
        ZmienLiteryNaDuze(WartoscStringBie[i]);
        WartoscBieNull[i] = SprawdzPoprawnoscString(WartoscStringBie[i]);

        if(WartoscPopNull[i] ==0 && WartoscBieNull[i] ==1) // Warunek dla brak danych dla oceny Poprzedniej
        {
            TabWynikowaString[i] = "Brak Pop";
        }
        else if(WartoscPopNull[i] ==1 && WartoscBieNull[i] ==0) // Warunek dla brak danych dla oceny Bieżącej
        {
            TabWynikowaString[i] = "Brak Bie";
        }
        else if(WartoscPopNull[i] ==1 && WartoscBieNull[i] ==1) // Warunek Obie oceny mają dane
        {
            if(WartoscStringPop[i] != WartoscStringBie[i]) // Obie oceny mają różne dane
            {
                TabWynikowaString[i] = "Rozny";
            }
        }
    }
}

```

```

    }
    for(int i=26; i<258; i++) // dla danych typu integer
    {
        WartoscStringPop[i] = wytnij(OcenaPop,OdZnaku[i],DoZnaku[i]); //
        WartoscPopNull[i] = SprawdzPoprawnoscString(WartoscStringPop[i]);

        WartoscStringBie[i] = wytnij(OcenaBie,OdZnaku[i],DoZnaku[i]);
        WartoscBieNull[i] = SprawdzPoprawnoscString(WartoscStringBie[i]);

        WartoscfloatPop[i] = NaInt(WartoscStringPop[i]); // zmiana ze string na float !!!!!
        WartoscfloatBie[i] = NaInt(WartoscStringBie[i]); // j.w !!!!!!!!!!!!!!!!!!!!!

        if(WartoscPopNull[i] ==0 && WartoscBieNull[i] ==1)
        {
            TabWynikowaString[i] = "Brak Pop";
        }
        else if(WartoscPopNull[i] ==1 && WartoscBieNull[i] ==0)
        {
            TabWynikowaString[i] = "Brak Bie";
        }
        else if(WartoscPopNull[i] ==1 && WartoscBieNull[i] ==1) // Kiedy istnieją dane dla oceny Pop i Bie
        {
            if(WartoscfloatPop[i]==0) // Jeśli wartość float = 0 to dodaj do niej jeden, uniknięcie dzielenie przez 0!
            {
                WartoscfloatPop[i]+=1;
                WartoscfloatBie[i]+=1;
            }

            //Obliczanie procenta spadku lub wzrostu w stosunku do oceny poprzedniej
            long procent = ((WartoscfloatBie[i]- WartoscfloatPop[i])/WartoscfloatPop[i])*100; // Obliczanie procenta

            if(WartoscStringPop[i] != WartoscStringBie[i]) // Warunek dla różnych wartości
            {
                TabWynikowaString[i] = "Rozny";
            }
            if(procent > 6 || procent < -6) //Warunek kiedy różnica w stosunku do oceny poprzedniej jest > 6%
            {
                TabWynikowaInt[i] = long(procent); // Wczytanie do tabeli wynikowej różnicy
                cout<<WartoscStringBie[0]<<" ==>"<<WartoscfloatBie[i]<<" ==>"<<WartoscfloatPop[i]<<"
==>"<<procent<<endl;
            }
        }
    }
}

```

Funkcja odpowiada na zmianę małych liter na wielkie

```

inline void ZmienLiteryNaDuze(string &Napis)
{
    for(int i=0; i<Napis.size(); i++)
    {
        if(Napis[i] >= 97 && Napis[i] <= 122) Napis[i] = Napis[i] -32;
    }
}

```

```
inline void Publikuj(fstream &plk, int Od, int Do, string rodzaj)
```

SprawdzDane(dane1,dane2,nr1,nr2, StaraNazwa, NowaNazwa,RozneNazwa,plkNazwa,16,27, "'Nazwa'");

SprawdzDane(dane1,dane2,nr1,nr2, StaryNumer, NowyNumer,RozneNumer,plkNumerPolski,28,41, "'Numer w Polsce'");

SprawdzDane(dane1,dane2,nr1,nr2, StaraDataUr, NowaDataUr,RozneDatyUr,plkDataUr,42,49, "'Data Urodzenia'")

SprawdzDane(dane1,dane2,nr1,nr2,StaraRasaOdmiana,NowaRasaOdmiana,RoznaRasaOdmiana,plkRasaOdmiana,50,51, "'Rasa/Odmiana'")

SprawdzDane(dane1,dane2,nr1,nr2, StaryNumerOjca, NowyNumerOjca,RoznyNumerOjca,plkNumerOjca,52,67, "'Numer Miedzynarodowy Ojca'");

SprawdzDane(dane1,dane2,nr1,nr2, StaraNazwaOjca, NowaNazwaOjca,RoznaNazwaOjca,plkNazwaOjca,68,79, "'Nazwa Ojca'");

SprawdzDane(dane1,dane2,nr1,nr2, StaryNumerOjcaWP, NowyNumerOjcaWP,RoznyNumerOjcaWP,plkNumerOjcaWP,80,93, "'Numer Ojca w Polsce'");

SprawdzDane(dane1,dane2,nr1,nr2, StaraRasaOjca, NowaRasaOjca,RoznaRasaOjca,plkRasaOjca,94,95, "'Rasa Ojca'");

SprawdzDane(dane1,dane2,nr1,nr2, StaryStatusBuhWPL, NowyStatusBuhWPL,RoznyStatusBuhWPL,plkStatusBuhWPL,192,193, "'Status Buhaja w Polsce'");

```

MlekoBiez.UstawDane(dane1);
MlekoPop.UstawDane(dane2);
SprawdzDaneMleko(MlekoPop,MlekoBiez,plkRaportMleko,nr1,nr2,RokiPop[rok],RokiBiez[rok],OcenionyWKraju, DSPop[rok], DSBie[rok]);

PokrojBie.UstawDane(dane1);
PokrojPop.UstawDane(dane2);
SprawdzDanePokroj(PokrojPop,PokrojBie,plkRaportPokroj,nr1,nr2,RokiPop[rok],RokiBiez[rok],OcenionyWKraju, DSPop[rok], DSBie[rok]);

PlodnoscBie.UstawDane(dane1);
PlodnoscPop.UstawDane(dane2);
SprawdzDanePlodnosc(PlodnoscPop,PlodnoscBie,plkRaportPlodnosc,nr1,nr2,RokiPop[rok],RokiBiez[rok],OcenionyWKraju, DSPop[rok], DSBie[rok]);

DystocjaCBie.UstawDane(dane1);
DystocjaCBPOP.UstawDane(dane2);
SprawdzDaneDystCurekBuh(DystocjaCBPOP,DystocjaCBie,plkRaportDystocjaCB,nr1,nr2,RokiPop[rok],RokiBiez[rok],OcenionyWKraju,
DSPop[rok],DSBie[rok]);

DystocjaBBie.UstawDane(dane1);
DystocjaBPOP.UstawDane(dane2);
SprawdzDaneDystBezposrednia(DystocjaBBie,DystocjaBPOP,plkRaportDystocjaBezpos,nr1,nr2,RokiPop[rok],RokiBiez[rok],OcenionyWKraju, DSPop[rok],
DSBie[rok]);

PrzeBie.UstawDane(dane1);
PrzePop.UstawDane(dane2);
SprawdzDanePrzezywalnosc(PrzeBie,PrzePop,plkRaportPrzezywalnosc,nr1,nr2,RokiPop[rok],RokiBiez[rok],OcenionyWKraju, DSPop[rok], DSBie[rok]);

IndeksyIBie.UstawDane(dane1);
IndeksyIPop.UstawDane(dane2);
SprawdzDaneIndeksy(IndeksyIBie,IndeksyIPop,plkRaportIndeksy,nr1,nr2,RokiPop[rok],RokiBiez[rok],OcenionyWKraju, DSPop[rok], DSBie[rok]);
}

```

Omówienie funkcji sprawdź dane tj. w pierwszym warunku sprawdza czy są dane dla oceny poprzedniej i bieżącej, jeśli są, do sprawdza czy dane się różnią czy są takie same, jeśli się różnią, funkcja przekazuje tę informację do pliku. W drugim warunku funkcja sprawdza czy są dane dla oceny bieżącej, jeśli ich brak to tę informację wypisuje do pliku. W trzecim warunku funkcja sprawdza czy są dane dla oceny poprzedniej, jeśli ich brak funkcja zapisuje tę informację do pliku.

```

inline int SprawdzDane(const string &dane1, const string &dane2, string nr1, string nr2, int &Stara, int &Nowa, int
&Rozne, fstream &plik,int od,int doo, string rodzaj)
{
    string wycinek1 = wytnij(dane1,od,doo);           // wycięcie danych ze zmiennej dane1
    string wycinek2 = wytnij(dane2,od,doo);           // wycięcie danych ze zmiennej dane2

    int wynik1 = SprawdzPoprawnoscString(wycinek1);   // sprawdzenie poprawności zmiennej 1 typu string
    int wynik2 = SprawdzPoprawnoscString(wycinek2);   // sprawdzenie poprawności zmiennej 2 typu string

    if(wynik1 ==1 && wynik2 ==1)                       // jeżeli dane są poprawne to jeżeli są dane dla oceny pop i bie
    {
        int zmienna =sprawdz(wycinek1,wycinek2);       // sprawdzenie czy dane różnią się
        if(zmienna ==0)                                // jeżeli dane się różnią to zostanie zapisana do pliku poniższa //formuła
        {
            //cout<<nr1<<" "<<"<<nazwa1<<"<"<====> "<<nazwa2<<"<"<====> "Rozna nazwa!!!"<<endl;
            plik<<nr1<<" "<<nr2<<" "<<wycinek1<<"<"<====> "<<wycinek2<<"<"<====> "Rozny "<<rodzaj<<"!!!"<<endl;
            Rozne++;
        }
    }
    return zmienna;
}

if(wynik1==1 && wynik2==0)                             // jeżeli dla oceny poprzedniej =1 a dla bieżącej =0
{
    Stara++;
    //cout<<nr1<<" "<<"<<nazwa1<<"<"<====> "<<nazwa2<<"<"<====> "Brak nazwy dla oceny bieżącej!!!"<<endl;
    plik<<nr1<<" "<<nr2<<" "<<wycinek1<<"<"<====> "<<wycinek2<<"<"<====> "Brak "<<rodzaj<<"<"<====> "dla oceny
    bieżącej!!!"<<endl;
    return 0;
}

```

```

    }
    if(wynik1==0 && wynik2==1)                // jeżeli dla oceny poprzedniej =0, a dla bieżącej = 1
    {
        Nowa++;
        //cout<<nr1<<"  "<<"<<nazwa1<<"<<"<====> "<<nazwa2<<" Brak nazwy dla oceny poprzedniej!!!"<<endl;
        plik<<nr1<<" "<<nr2<<" "<<wycinek1<<"<<"<====> "<<wycinek2<<" Brak "<<rodzaj<<" dla oceny
        poprzedniej!!!"<<endl;
        return 0;
    }
}

```

Funkcja sprawdza poprawność zmiennych typu string w poszczególnych kolumnach pliku.
funkcja sprawdza czy w stringu występują puste znaki, oznaczone w pliku jako * lub ' ' (spacje), jeżeli suma pustych znaków = sumie długości danego string-a wówczas uznaje się że wyraz jest pusty funkcja zwraca 0, w przeciwnym wypadku funkcja zwraca 1.

```

inline int SprawdzPoprawnoscString(string nazwa)
{
    int licz=0;
    for(int i=0; i<nazwa.length(); i++)
    {
        if(nazwa[i] == '*' || nazwa[i] == ' ') licz++; // zliczanie dla warunku występowania pustego znaku
    }
    if(nazwa.length() == licz) return 0;           // jeżeli zliczana wartość = długości Sting-a zwraca 0
    else
        return 1;                                // w przeciwny wypadku zwraca 1
}

```

```

inline void SprawdzDaneMleko(OcenaMleczna &OcenaPop, OcenaMleczna &OcenaBie, fstream &RaportMleko, string
nr1, string nr2, year &TabelaRocznikowPop, year &TabelaRocznikowBiez, const string &OcenionyWKraju, OdchylenieSt
&DSPop, OdchylenieSt &DSBie)
{
    //Deklaracja struktury danych, dla indeksów i podindeksów
    string WartMleko[6]={"Mleko[Kg]", "Tluszcz[Kg]", "Tluszcz[%]", "Bialko[Kg]", "Bialko[%]", "Komorki Somatyczne"};
    string PodindeksWartoscMleko[7] = {"Typ Oceny", "Liczba Obor", "Liczba Corek", "Liczba Corek
Efektywnych", "EDC", "Powtarzalnosc", "Wartosc Hodowlana"};

```

```

// Deklaracja zmiennych typu int
int liczK=0;
int liczG=0;
int liczGK=0;
int maxi=0;
int mini=1000000;

```

```

// Zmienne zliczające roczniki
TabelaRocznikowPop.ile++;
TabelaRocznikowBiez.ile++;
int ile =0;

```

```

for(int i=0; i<6; i++)
{

```

```

    int GenoKonwPop = OcenaPop.Cechy[i].Podindeksy[0].dane;
    int GenoKonwBie = OcenaBie.Cechy[i].Podindeksy[0].dane;

```

```

    // Przypisanie zmiennej int typu oceny
    // Przypisanie zmiennej int typu oceny

```



```

// Sprawdzenie Typu oceny, pierwszy przypadek dla takich samych typów, LUB drugi dla oceny poprzedniej
konwencjonalnej krajowo, dla bieżącej genomowo krajowej. LUB dla oceny poprzedniej konwencjonalnie
międzynarodowej, dla oceny bieżącej genomowo międzynarodowo..
    if((OcenaPop.Cechy[i].Podindeksy[0] == OcenaBie.Cechy[i].Podindeksy[0])||
        ((OcenaPop.Cechy[i].Podindeksy[0]==5 && OcenaBie.Cechy[i].Podindeksy[0]==2 )|| // ocena konwencjonalna
        (OcenaPop.Cechy[i].Podindeksy[0]==6 && OcenaBie.Cechy[i].Podindeksy[0]==3 ))) // ocena genomowa
    {

//Jeżeli wartości dla poszczególnych cech są większe o zera następuje inkrementacja licZK sumowanie osobników ocena
konwencjonalna
        if((OcenaPop.Cechy[i].Podindeksy[1] > 0 && OcenaBie.Cechy[i].Podindeksy[1] > 0) &&
            (OcenaPop.Cechy[i].Podindeksy[2] > 20 && OcenaBie.Cechy[i].Podindeksy[2] > 20) &&
            (OcenaPop.Cechy[i].Podindeksy[5] > 0 && OcenaBie.Cechy[i].Podindeksy[5] > 0))
        {
            licZK++;
        }

//Jeżeli wszystkie pod cechy produkcyjne są równe zero oznacza to że dany osobnik został oceniony genomowo następuje
inkrementacja licZG czyli sumowanie osobników dla oceny genomowej.
        if((OcenaPop.Cechy[i].Podindeksy[1] == 0 && OcenaBie.Cechy[i].Podindeksy[1] == 0) &&
            (OcenaPop.Cechy[i].Podindeksy[2] == 0 && OcenaBie.Cechy[i].Podindeksy[2] == 0) &&
            (OcenaPop.Cechy[i].Podindeksy[5] > 0 && OcenaBie.Cechy[i].Podindeksy[5] > 0))
        {
            licZG++;
        }

        //RaportMleko<<"Typ Oceny"<<GenoKonwPop<<" " <<GenoKonwBie<<endl;
// Jeżeli jest 5 dla oceny poprzedniej i 2 dla bieżącej, czyli w ocenie poprzedniej oceniony genomowo w kraju, a w ocenie
bieżącej oceniony konwencjonalnie w kraju, LUB 6 i 3, oceniony genomowo międzynarodowo i oceniony konwencjonalnie
międzynarodowo
        if((OcenaPop.Cechy[i].Podindeksy[0]==5 && OcenaBie.Cechy[i].Podindeksy[0]==2 )||
            (OcenaPop.Cechy[i].Podindeksy[0]==6 && OcenaBie.Cechy[i].Podindeksy[0]==3 ))
        {

//drukowany jest raport
RaportMleko<<nr1<<"<<nr2<<"<<OcenionyWKraju<<"<<GenoKonwPop<<"<<GenoKonwBie<<"<<";<<Wart
Mleko[i]<<"<<PodindeksWartoscMleko[0]<<"<<Oceniony gonomowo i konwencjonalnie;"<<endl;
        }

// Obliczenie Procentu spadku/ wzrost w stosunku do oceny bieżącej.
for(int j=1; j<6; j++) // j=1 ponieważ dla j=0 jest pod cecha typ oceny.
{
    float liczbaPop = OcenaPop.Cechy[i].Podindeksy[j].dane; //Przypisanie zmiennej float wartości pod indeks
    float liczbaBie = OcenaBie.Cechy[i].Podindeksy[j].dane; //Przypisanie zmiennej float wartości pod indeks
    float liczbaPopP = liczbaPop;
    float liczbaBieB = liczbaBie;
    if(liczbaPop==0) // próba uniknięcia dzielenia przez zero podczas liczenia procenta
    {
        liczbaPopP+=1;
        liczbaBieB+=1;
    }
    float Procent = ((liczbaBieB-liczbaPopP)/liczbaPopP)*100; // Obliczenie procenta
    int ProcentWl = Procent;
    string SpadekWzrost="Wzrost"; // dla zmiennej SpadekWzrost przypisanie Wzrost

    if(Procent<0) // Jeżeli obliczony procent jest mniejszy od ZERO
    {
        SpadekWzrost = "Spadek"; // przypisaną wartość Wzrost zmieniamy na Spadek
        ProcentWl = ProcentWl*(-1); // mnożymy razy (-1) w celu wyeliminowanie ujemnej wartości
    }

    if(liczbaPop > 0 && liczbaBie >0) // Jeżeli wartości poszczególnych pod indeksów są >0
    {

```


//!!!!!!! Ważne założenie od jakiego progu uznawać błąd, po ustaleniach okazało się że będzie do 5% spadku lub wzrostu w stosunku do oceny bieżącej !!!!!!!!!!!

```
if(ProcentWl > 5)
```

```
{
```

```
// Drukowanie raportu dla wzrost/spadek 5%
```

```
RaportMleko<<nr1<<";"<<nr2<<";"<<OcenionyWKraju<<";"<<GenoKonwPop<<";"<<GenoKonwBie<<";"<<liczba  
Pop<<";"<<liczbaBie<<";"<<WartMleko[ij]<<";"<<PodindeksWartoscMleko[ij]<<";"<<SpadekWzrost<<";"<<Procent  
Wl<<";% w stosunku do oceny Poprzedniej!!!;"<<endl;
```

```
}
```

```
}  
Else // Jeżeli wartości poszczególnych pod indeksów nie są >0  
{
```

```
if(liczbaPop==0 && liczbaBie >0) // jeżeli wartość oceny poprzedniej = 0 a ocena bieżąca > 0 to
```

```
{
```

```
// Drukuje raport z uwagą „Brak danych dla oceny poprzedniej !!!
```

```
RaportMleko<<nr1<<";"<<nr2<<";"<<OcenionyWKraju<<";"<<GenoKonwPop<<";"<<GenoKonwBie<<";"<<liczba  
Pop<<";"<<liczbaBie<<";"<<WartMleko[ij]<<";"<<PodindeksWartoscMleko[ij]<<";";brak danych dla ocena  
poprzednia!!!;"<<endl;
```

```
}
```

```
if(liczbaBie==0 && liczbaPop >0) // jeśli wartość oceny poprzedniej >0 a ocena bieżąca = 0
```

```
{
```

```
// Drukuje raport z uwagą „ Brak danych dla oceny bieżącej !!!!!!!!!”
```

```
RaportMleko<<nr1<<";"<<nr2<<";"<<OcenionyWKraju<<";"<<GenoKonwPop<<";"<<GenoKonwBie<<";"<<liczba  
Pop<<";"<<liczbaBie<<";"<<WartMleko[ij]<<";"<<PodindeksWartoscMleko[ij]<<";";brak danych dla ocena  
bieżąca!!!;"<<endl;
```

```
}
```

```
}
```

```
}
```

//!!!! Powtórzenie czynności dla pod indeks Wartość Hodowlana, dla której 0 jest wartością, dla Wartości hodowlanej zero może być wynikiem tak samo jak wartości ujemne.

```
float liczbaPop = OcenaPop.Cechy[ij].Podindeksy[6].dane; //Przypisanie zmiennej float wartości dla WH
```

```
float liczbaBie = OcenaBie.Cechy[ij].Podindeksy[6].dane; //Przypisanie zmiennej float wartości dla WH
```

```
float liczbaPopP = liczbaPop;
```

```
float liczbaBieB = liczbaBie;
```

//Aby nie operować na ujemnych wartościach, w przypadku wystąpienia takich, zmieniamy ją na dodatnią i proporcjonalnie zmieniamy wartość oceny POP lub BIE jeżeli była dodatnia.

```
if(liczbaPopP <0 && liczbaBieB >0) //Wartość dla oceny POP <0 i dla oceny BIE >0
```

```
{
```

```
    liczbaPopP *=(-1); // zmiana znaku z ujemnej na dodatnią dla oceny POP
```

```
    liczbaBieB +=liczbaPopP+1; // Dodanie wartości z oceny POP do BIE plus 1
```

```
    liczbaPop =1; // Przypisanie 1 dla wartości Oceny POP
```

```
}
```

```
if(liczbaPopP >0 && liczbaBieB <0) // Wartość oceny POP >0 ocena BIE < 0
```

```
{
```

```
    liczbaBieB *=(-1); // Zmiana znaku z ujemnego na dodatni dla oceny BIE
```

```
    liczbaPopP +=liczbaBieB+1; //Dodanie wartości z oceny BIE do POP plus 1
```

```
    liczbaBieB =1; // Przypisanie 1 dla wartości oceny BIE
```

```
}
```

```
if(liczbaPop==0) // Próba uniknięcia dzielenia przez ZERO
```

```
{
```

```
    liczbaPopP+=1; // Dodanie 1 do oceny POP
```

```
    liczbaBieB+=1; // Dodanie 1 do oceny BIE
```

```
}
```

```
float Procent = ((liczbaBieB-liczbaPopP)/liczbaPopP)*100; // Obliczenie Procenta
```

```
int ProcentWl = Procent;
```

```
string SpadekWzrost="Wzrost"; // Przypisanie wartości wzrost
```

```
if(Procent<0) // Dla ujemnej wartości procent
```

```
{
```

```

        SpadekWzrost = "Spadek"; // Zmiana WzrostSpadek na spadek
        ProcentWl = ProcentWl*(-1); // * -1 w celu wyeliminowania ujemnej wartości.
    }

    //!!!!!! OcenaPop.Cechy[i].Podindeksy[6].dlaWH != 1 Oznacza to iż dlaWH jest dodatkowym parametrem
    który mówi czy dla Wartości Hodowlanej wartość jest zerowa, czy nie ma jej wcale, jeżeli wartość jest zerowa
    dodatnia lub ujemna *.dlaWH = 0 , jeżeli nie ma wartości czyli jest pusty znak, biały znak w linii, prze co nie
    można przeprowadzi operacji matematycznych wówczas *.dlaWH = 1.

    if(OcenaPop.Cechy[i].Podindeksy[6].dlaWH != 1 && OcenaBie.Cechy[i].Podindeksy[6].dlaWH != 1)
    {
        if(ProcentWl > 5) //!!!!Określony próg poprawności
        {
            // Generowanie raportu dla WH ze wzrost/spadek powyżej 5%
            RaportMleko<<nr1<<";"<<nr2<<";"<<OcenionyWKraju<<";"<<GenoKonwPop<<";"<<GenoKonwBie<<";"<<liczba
            Pop<<";"<<liczbaBie<<";"<<WartMleko[i]<<";"<<PodindeksWartoscMleko[6]<<";"<<SpadekWzrost<<";"<<Procent
            Wl<<";% w stosunku do oceny Poprzedniej!!!;"<<endl;

        }

        liczG++; // liczenie osobników genomowych
        liczK++; // liczenie osobników konwencjonalnych
        ile++; // liczenie wszystkich osobników

    }
    Else // W przypadku kiedy prak jest danych dla jednej z ocen
    {
        // W przypadku kiedy brak jest danych dla oceny POP
        if(OcenaPop.Cechy[i].Podindeksy[6].dlaWH == 1 && OcenaBie.Cechy[i].Podindeksy[6].dlaWH == 0)
        {
            RaportMleko<<nr1<<";"<<nr2<<";"<<OcenionyWKraju<<";"<<GenoKonwPop<<";"<<GenoKonwBie<<";"<<liczba
            Pop<<";"<<liczbaBie<<";"<<WartMleko[i]<<";"<<PodindeksWartoscMleko[6]<<";";brak danych dla ocena
            poprzednia!!!;"<<endl;
        }
        // W przypadku kiedy brak jest danych dla oceny bieżącej
        if(OcenaBie.Cechy[i].Podindeksy[6].dlaWH == 1 && OcenaPop.Cechy[i].Podindeksy[6].dlaWH == 0)
        {
            RaportMleko<<nr1<<";"<<nr2<<";"<<OcenionyWKraju<<";"<<GenoKonwPop<<";"<<GenoKonwBie<<";"<<liczba
            Pop<<";"<<liczbaBie<<";"<<WartMleko[i]<<";"<<PodindeksWartoscMleko[6]<<";";brak danych dla ocena
            bierzaca!!!;"<<endl;
        }
    }
    Else // W przypadku kiedy typ oceny jest inny niż w wyżej wymienionym warunku
    {
        // Generuje raport o niewłaściwej wartości w typie oceny
        RaportMleko<<nr1<<";"<<nr2<<";"<<OcenionyWKraju<<";"<<GenoKonwPop<<";"<<GenoKonwBie<<";";"<<Wart
        Mleko[i]<<";"<<PodindeksWartoscMleko[0]<<";";Niewlasciew wartosc w Typie oceny;"<<endl;
    }

    }

    //Ocena Konw. Moduł Odchylenie standardowe

    if(((GenoKonwPop==2 && GenoKonwBie==2)||((GenoKonwPop==3 && GenoKonwBie==3)) && liczK==2)
    {
        (TabelaRocznikowPop.ileK[i])++; // Dodanie do tabeli roczników dla oceny Poprzednie
        (TabelaRocznikowBiez.ileK[i])++; // Dodanie do tabeli roczników dla oceny Bieżącej

        int mm =OcenaPop.Cechy[i].Podindeksy[1].dane; // utworzenie zmiennej do obliczenia minimum/maximum
        //int nn =

```

```

// Obliczenie minimum/maximum
    if(mm > maxi) maxi=mm;
    if(mm < mini) mini =mm;

    //cout<<maxi<<"    "<<mini<<endl;
    //cout<<OcenaBie.Cechy[i].Podindeksy[1].dane<<endl;

//Wczytanie do tabeli roczników danych z indeksów i podindeksów
    for(int j=0; j<7; j++)
    {
        TabelaRocznikowPop.MlekoK[i][j] += OcenaPop.Cechy[i].Podindeksy[j].dane; //
        TabelaRocznikowBiez.MlekoK[i][j] += OcenaBie.Cechy[i].Podindeksy[j].dane;
    }

//Dodanie do Wektora podindeksu z wartości hodowlanych dla siedmiu indeksów
    //DS[LRok].MlekoPopK[i] = OcenaPop.Cechy[i].Podindeksy[6].dane;
    //DS[LRok].MlekoBieK[i] = OcenaBie.Cechy[i].Podindeksy[6].dane;
    (DSPop.VekMlekoK[i]).push_back(OcenaPop.Cechy[i].Podindeksy[6].dane); // Dodanie do wektora Pop WH
    (DSBie.VekMlekoK[i]).push_back(OcenaBie.Cechy[i].Podindeksy[6].dane);
}

// Ocena Genom Moduł Odchylenie standardowe
    if(((GenoKonwPop==5 && GenoKonwBie==5)|| (GenoKonwPop==6 && GenoKonwBie==6)) && liczG==2)
    {
        (TabelaRocznikowPop.ileG[i])++; // Dodanie do tabeli roczników dla oceny Poprzednie
        (TabelaRocznikowBiez.ileG[i])++; // Dodanie do tabeli roczników dla oceny Bieżącej

//Wczytanie do tabeli roczników danych z indeksów i pod indeksów
        for(int j=0; j<7; j++)
        {
            TabelaRocznikowPop.MlekoG[i][j] += OcenaPop.Cechy[i].Podindeksy[j].dane;
            TabelaRocznikowBiez.MlekoG[i][j] += OcenaBie.Cechy[i].Podindeksy[j].dane;
        }

//Dodanie do Wektora podindeksu z wartości hodowlanych dla siedmiu indeksów
        (DSPop.VekMlekoG[i]).push_back(OcenaPop.Cechy[i].Podindeksy[6].dane);
        (DSBie.VekMlekoG[i]).push_back(OcenaBie.Cechy[i].Podindeksy[6].dane);
    }

// Ocena GenoKonw. Ocena kombinowana, są to osobniki dla których była ocena genomowa ale z czasem pojawiła się
konwencjonalna.

    if(((GenoKonwPop==5 && GenoKonwBie==2)|| (GenoKonwPop==6 && GenoKonwBie==3) ||
        (GenoKonwPop==5 && GenoKonwBie==5) || (GenoKonwPop==6 && GenoKonwBie==6)) && liczK ==2)
    {
        (TabelaRocznikowPop.ileGK[i])++; // Dodanie do tabeli roczników dla oceny Poprzednie
        (TabelaRocznikowBiez.ileGK[i])++; // Dodanie do tabeli roczników dla oceny Bieżącej

//Wczytanie do tabeli roczników danych z indeksów i pod indeksów
        for(int j=0; j<7; j++)
        {
            TabelaRocznikowPop.MlekoGK[i][j] += OcenaPop.Cechy[i].Podindeksy[j].dane;
            TabelaRocznikowBiez.MlekoGK[i][j] += OcenaBie.Cechy[i].Podindeksy[j].dane;
        }

//Dodanie do Wektora podindeksu z wartości hodowlanych dla siedmiu indeksów
        (DSPop.VekMlekoGK[i]).push_back(OcenaPop.Cechy[i].Podindeksy[6].dane);
        (DSBie.VekMlekoGK[i]).push_back(OcenaBie.Cechy[i].Podindeksy[6].dane);
    }

    liczK = 0;
    liczG = 0;
}
};

```

Wygenerowanie raportu do pliku raport.txt.

```
plkRaport<<"*****Nazwa*****" <<endl;
plkRaport<<"Liczba osobnikow dla ktorych brak nazwy w pliku "<<OcenaPoprzednia<<" = "<<NowaNazwa<<endl;
plkRaport<<"Liczba osobnikow dla ktorych brak nazwy w pliku "<<OcenaBizaca<<" = "<<StaraNazwa<<endl;
plkRaport<<"Liczba osobnikow dla ktorych nazwy roznia sie "<<" = "<<RozneNazwa<<endl;
plkRaport<<endl;

plkRaport<<"*****Numer w Polsce*****" <<endl;
plkRaport<<"Liczba osob. dla ktorych br. numeru Polskiego w pliku "<<OcenaPoprzednia<<" = "<<NowyNumer<<endl;
plkRaport<<"Liczba osob. dla ktorych brak NumeruPolskieg w pliku "<<OcenaBizaca<<" = "<<StaryNumer<<endl;
plkRaport<<"Liczba osob. dla ktorych numery Poskie roznia sie "<<" = "<<RozneNumer<<endl;
plkRaport<<endl;

plkRaport<<"*****Data Urodzenia*****" <<endl;
plkRaport<<"Liczba os. dla ktorych brak daty urodzenia w pliku "<<OcenaPoprzednia<<" = "<<NowaDataUr<<endl;
plkRaport<<"Liczba osob. dla ktorych brak daty urodzenia w pliku "<<OcenaBizaca<<" = "<<StaraDataUr<<endl;
plkRaport<<"Liczba os. dla ktorych daty urodzenia roznia sie "<<" = "<<RozneDatyUr<<endl;
plkRaport<<endl;

plkRaport<<"*****rasa/odmiana*****" <<endl;
plkRaport<<"Liczba os. dla ktorych brak rasay w pliku "<<OcenaPoprzednia<<" = "<<NowaRasaOdmiana<<endl;
plkRaport<<"Liczba os. dla ktorych brak rasy w pliku "<<OcenaBizaca<<" = "<<StaraRasaOdmiana<<endl;
plkRaport<<"Liczba osobnikow dla ktorych rasa/odmiana roznia sie "<<" = "<<RoznaRasaOdmiana<<endl;
plkRaport<<endl;

plkRaport<<"*****Numer miedzynarodowy ojca*****" <<endl;
plkRaport<<"Lb. os. dla ktorych br. nr miedzyNar ojca w pliku "<<OcenaPop<<" = "<<NowyNumerMOjca<<endl;
plkRaport<<"Lb. Os. dla ktorych br. nr miedzyNar ojca w pliku "<<OcenaBizaca<<" = "<<StaryNumerMOjca<<endl;
plkRaport<<"Lb. Os. dla ktorych nr miedzynarodowego ojca roznia sie "<<" = "<<RoznyNumerMojca<<endl;
plkRaport<<endl;

plkRaport<<"*****Nazwa ojca*****" <<endl;
plkRaport<<"Lb. Os. dla ktorych brak nazwy ojca w pliku "<<OcenaPoprzednia<<" = "<<NowaNazwaOjca<<endl;
plkRaport<<"Lb. Os. dla ktorych brak nazwy ojca w pliku "<<OcenaBizaca<<" = "<<StaraNazwaOjca<<endl;
plkRaport<<"Liczba osobnikow dla ktorych nazwa ojca rozni sie "<<" = "<<RoznaNazwaOjca<<endl;
plkRaport<<endl;

plkRaport<<"*****Numer Ojca Polsce*****" <<endl;
plkRaport<<"Lb. Os. dla ktorych brak nr. ojca w pl "<<OcenaPoprzednia<<" = "<<NowyNumerOjcaWP<<endl;
plkRaport<<"Lb. Os. dla ktorych brak nr. ojca w pl. w pliku "<<OcenaBizaca<<" = "<<StaryNumerOjcaWP<<endl;
plkRaport<<"Liczba osobnikow dla ktorych numer ojca w polsce rozni sie "<<" = "<<RoznyNumerOjcaWP<<endl;
plkRaport<<endl;

plkRaport<<"*****rasa/odmiana ojca*****" <<endl;
plkRaport<<"Liczba os. dla ktorych brak rasy ojca w pl. w pliku "<<OcenaPoprzednia<<" = "<<NowaRasaOjca<<endl;
plkRaport<<"Liczba os. dla ktorych brak rasy ojca w polsce w pliku "<<OcenaBizaca<<" = "<<StaraRasaOjca<<endl;
plkRaport<<"Liczba osobnikow dla ktorych rasa/odmiana ojca w polsce rozni sie "<<" = "<<RoznaRasaOjca<<endl;
plkRaport<<endl;

plkRaport<<"*****Status Buhaja w Polsce*****" <<endl;
plkRaport<<"Lb. Os. dla ktorych brak statusu buhaja w pl. "<<OcenaPoprzednia<<" = "<<NowyStatusBuhWPL<<endl;
plkRaport<<"Lb. Os. dla ktorych brak statusu buhaja w pl"<<OcenaBizaca<<" = "<<StaryStatusBuhWPL<<endl;
plkRaport<<"Lb. Os. dla ktorych statusu buhaja w pl. "<<" = "<<RoznyStatusBuhWPL<<endl;
plkRaport<<"* "<<endl;
```

Obliczanie średnich dla poszczególnych roczników:

```
WypiszSrednie(RokiPop,RokiBiez,6,7,"K","Mleko",plkRaportSrednie);
plkRaportSrednieRocznikami<<"*****Mleko*****"<<endl;
RokiRokiRoki(RokiPop,RokiBiez,6,7,"Mleko","K",plkRaportSrednieRocznikami,DSPop,DSBie);
plkRaportSrednieRocznikami<<"*****Pokroj*****"<<endl;
RokiRokiRoki(RokiPop,RokiBiez,22,6,"Pokroj","K",plkRaportSrednieRocznikami,DSPop,DSBie);
plkRaportSrednieRocznikami<<"*****Plodnosc*****"<<endl;
RokiRokiRoki(RokiPop,RokiBiez,4,6,"Plodnosc","K",plkRaportSrednieRocznikami,DSPop,DSBie);
plkRaportSrednieRocznikami<<"*****Dystocja Corek Buhaja*****"<<endl;
RokiRokiRoki(RokiPop,RokiBiez,10,1,"DystocjaCB","G",plkRaportSrednieRocznikami,DSPop,DSBie);
plkRaportSrednieRocznikami<<"*****Dystocja Brzposrednia*****"<<endl;
RokiRokiRoki(RokiPop,RokiBiez,10,1,"Dystocja Bezposrednia","G",plkRaportSrednieRocznikami,DSPop,DSBie);
plkRaportSrednieRocznikami<<"*****Przezywalnosc*****"<<endl;
RokiRokiRoki(RokiPop,RokiBiez,3,1,"Przezywalnosc","G",plkRaportSrednieRocznikami,DSPop,DSBie);
plkRaportSrednieRocznikami<<"*****Indeksy*****"<<endl;
RokiRokiRoki(RokiPop,RokiBiez,9,1,"Indeksy","G",plkRaportSrednieRocznikami,DSPop,DSBie);
```

Omówienie funkcji z bloku funkcyjnego:

Funkcja WypiszSrednie zajmuje się wyliczeniem średnich dla poszczególnych roczników,

```
inline void WypiszSrednie(year *RokiPop, year *RokiBiez, int ileWierszy, int ileKolumn, string TypOceny, string Rodzaj,
fstream &plkSrednie )
```

```
{
    plkSrednie<<Rodzaj<<endl;
    plkSrednie<<endl;

    for(int i=100; i<200; i++)
    {
        int *wskP, *wskB;

        if(TypOceny=="K")                // Ocena konwencjonalna
        {
            wskP = RokiPop[i].ileK;      // Przypisanie ilość osobników dla oceny Poprzedniej
            wskB = RokiBiez[i].ileK;     // Przypisanie ilość osobników dla oceny Bieżącej
        }
        else
        if(TypOceny=="G")                // Ocena genomowa
        {
            wskP = RokiPop[i].ileG;      // Przypisanie ilość osobników dla oceny Poprzedniej
            wskB = RokiBiez[i].ileG;     // Przypisanie ilość osobników dla oceny Bieżącej
        }
        else
        if(TypOceny=="GK")              // Ocena Genomowo-Konwencjonalna
        {
            wskP = RokiPop[i].ileGK;    // Przypisanie ilość osobników dla oceny Poprzedniej
            wskB = RokiBiez[i].ileGK;    // Przypisanie ilość osobników dla oceny Bieżącej
        }
        }

        cout<<RokiPop[i].ile<<endl;
        if(RokiPop[i].ile >0)
        {
            plkSrednie<<"Dla roku "<<i+1900<<endl;
            plkSrednie<<endl;
            plkSrednie<<endl;
            plkSrednie<<"Ocena PoP"<<endl;
            plkSrednie<<RokiPop[i]<<endl;
            plkSrednie<<"Ocena Biez"<<endl;
            plkSrednie<<RokiBiez[i]<<endl;
        }
    }
}
```

```

for(int i=100; i<200; i++)
{
    cout<<i+1900<<" III "<<endl;
    DSPop[i].LiczDSMleko(RokiPop[i]);
    DSBie[i].LiczDSMleko(RokiBiez[i]);
    cout<<"*****Pokroj*****"<<endl;
    DSPop[i].LiczDSPokroj(RokiPop[i]);
    DSBie[i].LiczDSPokroj(RokiBiez[i]);
    cout<<"*****Plodnosc*****"<<endl;
    DSPop[i].LiczDSPlodnosc(RokiPop[i]);
    DSBie[i].LiczDSPlodnosc(RokiBiez[i]);
    cout<<"*****DystocjaCB*****"<<endl;
    DSPop[i].LiczDSDystocjaCB(RokiPop[i]);
    DSBie[i].LiczDSDystocjaCB(RokiBiez[i]);
    cout<<"*****DystocjaBezp*****"<<endl;
    DSPop[i].LiczDSDystocjaBezp(RokiPop[i]);
    DSBie[i].LiczDSDystocjaBezp(RokiBiez[i]);
    cout<<"*****Przezywalnosc*****"<<endl;
    DSPop[i].LiczDSPrzezywalnosc(RokiPop[i]);
    DSBie[i].LiczDSPrzezywalnosc(RokiBiez[i]);
    cout<<"*****Indeksy*****"<<endl;
    DSPop[i].LiczDSIndeksy(RokiPop[i]);
    DSBie[i].LiczDSIndeksy(RokiBiez[i]);
    //DSPop[i].wyswietl();
    //DSBie[i].wyswietl();
}

```

Omówienie Bloku funkcyjnego:

Funkcja liczy odchylenie standardowe dla poszczególnych roczników :

inline void LiczDSMleko(year &Roki) // rodzaj 1-Mleko, 2-Pokroj, 3-Plodnosc,

```

{
    this->rok = Roki.rok;
    for(int i=0; i<6; i++)
    {
        // zerowanie zmiennych
        DSMlekoK[i] =0;
        DSMlekoG[i]=0;
        DSMlekoGK[i]=0;

        int dlugoscK = (VekMlekoK[i]).size(); // Rozmiar wektora (ilość osobników) dla oceny konwencjonalnej
        int dlugoscG = (VekMlekoG[i]).size(); // Rozmiar wektora (ilość osobników) dla oceny Genomowej
        int dlugoscGK = (VekMlekoGK[i]).size(); // Rozmiar wektora (ilość osobników) dla oceny GenomowejKonwec.

        float sredniaK = Roki.MlekoK[i][6]/Roki.ileK[i]; // Średnie dla poszczególnych roczników ocena konwencjonalna
        float sredniaG = Roki.MlekoG[i][6]/Roki.ileG[i]; // Średnie dla poszczególnych roczników ocena genomowej
        float sredniaGK = Roki.MlekoGK[i][6]/Roki.ileGK[i]; // średnie dla poszczególnych roczników oceny genoKonw

        // Obliczenie Odchylenia standardowego dla oceny konwencjonalnej
        if(dlugoscK >0 && Roki.ileK[i] >0) // Warunek dane muszą być większe od zera
        {
            for(int j=0; j<dlugoscK; j++)
            {

                int liczba = VekMlekoK[i][j];

                DSMlekoK[i] += ((liczba - sredniaK)*(liczba - sredniaK)); // Obliczenie odchylenia st dla oceny konw.
            }

            DSMlekoK[i] = sqrt((1*DSMlekoK[i])/(Roki.ileK[i]-1)); // Wyciągnięcie pierwiastka zgodnie z wzorem na SD
            cout<<" Konwenc " <<j<<" " <<int(DSMlekoK[i])<<" "<<Roki.ileK[i]<<" "<<dlugoscK<<" "<<sredniaK<<endl;
        }
    }
}

```



```

// Obliczenie Odchylenia standardowego dla oceny genomowej
if(dlugoscG >0 && Roki.ileG[ij] >0)
{
    for(int j=0; j<dlugoscG; j++)
    {
        int liczba = VekMlekoG[i][j];
        DSMlekoG[ij] += ( liczba - sredniaG )*( liczba -sredniaG );    // Obliczenie odchylenia standardowego
    }
    DSMlekoG[ij] = sqrt((1*DSMlekoG[ij])/(Roki.ileG[ij]-1));

cout<<" Genom " <<j<<" " <<int(DSMlekoG[ij])<<" " <<Roki.ileG[ij]<<" " <<dlugoscG<<" " <<sredniaG<<endl;
}

// Obliczanie odchylenia standardowego dla oceny kombinowanej Genomowo-Konwencjonalnej
if(dlugoscGK >0 && Roki.ileGK[ij] >0)
{
    for(int j=0; j<dlugoscGK; j++)
    {
        int liczba = VekMlekoGK[i][j];
        DSMlekoGK[ij] += ( liczba - sredniaGK )*( liczba - sredniaGK ); // Obliczenie odchylenia standardowego
    }
    DSMlekoGK[ij] = sqrt((1*DSMlekoGK[ij])/(Roki.ileGK[ij]-1));
    cout<<" Geno-Konwenc " <<j<<" " <<int(DSMlekoGK[ij])<<" " <<Roki.ileGK[ij]<<" "
    <<dlugoscGK<<" " <<sredniaGK<<endl;

}

}

};

//Czyszczenie pamięci dla pliku startowego
for(int i=0; i<Lwierszy1; i++)
{
    delete tab1[i];
}
delete [] tab1;

//Czyszczenie pamięci dla pliku nowego
for(int i=0; i<Lwierszy2; i++)
{
    delete tab2[i];
}
delete [] tab2;

delete [] buf1;
delete [] DSPop;
delete [] DSBie;
delete [] buf2;

}

```