Podział plików na pliki HPP i CPP.

Najpierw funkcje standardowe, a później:
#include „struktury.h”
#include „funkcje.h”

Modyfikowanie main.cpp -> modyfikowany main.o.
Każda z funkcji źródłowych jest kompilowana osobno. -c oznacza tylko kompilacje. Pliki main.o i funkcje.o łączą się do main.cpp.

Dodać pliki źródłowe do projektu.

W pliku h umieszczamy nagłówki funkcji, np. std:: string odwroc (const std::string & napis);
Komentarz doxygenowy tylko w pliku h. W komentarzu jest informacja co funkcja robi. Przy używaniu czyjegoś programu używamy tylko pliku h.

Aby uniknąć zbędnego dołączania plików dodajemy:

#ifndef FUNKCJE_H
#define FUKCJE_H

… (kod)

#endif

Są to dyrektywy preprocesora.

Nie używać pragma (używane w Visualu, nieprzenośne). Powyższa forma jest standardem.

Napisy

```
std::string napis („tresc napisu”);
```

Podstawowym typem przechowywania napisów w c++ jest char*, nie string -> nie używać auto.

String też jest kontenerem -> można używać size, capacity… .

Do poszczególnych znaków można się dostać poprzez [ ], zupełnie jak dla tablic indeksowane od 0.

Znaki przekazujemy w pojedynczych cudzysłowach.

\ - znak ucieczki, używany np. gdy chcemy \”o\” w cudzysłowie w coucie, zamiast ‚o’.

```
for (auto znak : napis) // dla każdego znaku w napisie…
for (auto & znak : napis) // to samo, ale gdy chcemy zachować zmiany w zmiennych - sytuacja analogiczna, jak w funkcjach
```

Referencja jest konieczna aby modyfikować oryginalny znak, a nie jego kopię.

Do napisów, tak jak w wektorze, można dodać lub usunąć znaki poprzez push_back i pop_back.
Po usunięciu warto usunąć puste miejsca shrink_to_fit.

Dostęp do pierwszego i ostatniego elementu:

```
napis.back() = „ . ” // zmiana ostatniego elementu
napis.front() // dostęp do 1 elementu
```

Funkcja find znajduje indeks np. danej części napisu. Jeśli nic nie znajdzie, wyświetli „npos” (not a position).

size_t przechowuje tylko wartości dodatnie.

```
if (miejsce == std::string::npos)
    std::cout << „nie znaleziono”; // aby nie wyświetlało liczby npos
```

substr (substring) wyświetla fragment do danego znaku.

```
auto przecinek = napis.find(, , ’);
auto litwa = napid.substr(0, przecinek); // wyświetla fragment od początku do przecinka
```

Zwracanie znaków w odwrotnej kolejności:

```
auto dlugosc = napis.legth();

for (std::size_t i=dlugosc; i > 0; i- -);
    do_zwrotu.push_back(napis[i-1]);
```

Struktury

```
struct nazwa
{


}; ŚREDNIK JEST NIEZBĘDNY
```

Struktury są publiczne, a klasy prywatne - jedyna różnica.

Wypisywanie - sposób „drewniany”

```
void wypisz (const student & st)
{

    std::cout << st.imie << „ „ << st.nazwisko << …….

}
```

Wyrównanie słowa maszynowego dla całej struktury -> rozmiar elementów 76
-> rozmiar struktury 80

Inicjowanie struktury studenta - sposób szybki:

```
student EMatianek { „Ewa”, „Matianek”, 4325, 4.67 };
```

Wypisywanie szybkie

```
for (const auto & st : grupa)
    wypisz(st);
```

Sprawdzanie pustości wektora:

```
grupa.size( ) == 0
grupa.empty( )
```

Struktura anonimowa

```
struct
{

        int liczba;
        int znak;

} anonimowa;

anonimowa.liczba = 5;
anonimowa.znak = , g ';
```

Można definiować struktury w funkcji, ale nie funkcję w innej funkcji.

Unie

```
union
{

}
```

Wykorzystywane np. przy budowie kompilatorów.

Można odczytać tylko jedną wartość w danym momencie.

W unii wszystkie wielkości zmiennych są nałożone na siebie.

Do struktury można dodać string, do unii nie.