Tablice

- Tablice w C++ mogą być jedno- lub więcej wymiarowe, i ich elementami mogą być dowolne obiekty. Deklaracja tablicy wygląda następująco

```
int tab[10];
```

- Zadeklarowaliśmy w ten sposób tablicę składającą się z 10 zmiennych typu `int`. Do tych zmiennych odnosimy się następująco: `tab[0]`, ..., `tab[9]`. Pamiętajmy, że elementy indeksowane są od 0.

Tablice

- Rozmiar tablicy musi być znany w momencie kompilacji. Jeżeli jest zmienną, to musi być typu `const`. To nie jest żadne ograniczenie, po prostu w ten sposób definiujemy tablice *statyczne*. Mają one różne zalety, na przykład dostęp do elementów jest szybki. Ale rozmiar musi być określony w momencie kompilacji. Jeżeli potrzebujemy tablicy, której rozmiar nie jest znany w momencie kompilacji, korzystamy z tablic *dynamicznych*, o których też będziemy niedługo mówić.
- Niektóre kompilatory mogą pozwolić nam zdefiniować tablicę o nieznanym w momencie kompilacji rozmiarze, ale to będzie po cichu zrobiona tablica dynamiczna. Nie jest to zgodne ze standardem, i lepiej tego unikać.

Tablice

- Przykładowy kod z użyciem tablicy:

```
int tab[10];

for (int i = 0; i < 10; i++)
  tab[ i ] = i*i;
```

- Przypomnijmy, że indeksy zaczynają się od 0 a ostatnim indeksem jest `rozmiar -1`. Pilnowanie rozmiaru tablicy należy do nas. Jeżeli w powyższym kodzie pętla biegłaby do `i = 10`, to program zapisałby obszar liczbę 100 (`i*i`) w kolejnych komórkach pamięci bezpośrednio po naszej tablicy. Jest bardzo prawdopodobne, że te komórki byłyby wykorzystywane przez inne obiekty programu, i zostałyby one zniszczone. Również odczyt, na przykład `tab[14]` zwróciłby nam jakąś przypadkową liczbę, odczytaną z danych w kolejnych komórkach pamięci. jest będzie po cichu zrobiona tablica dynamiczna. Nie jest to zgodne ze standardem, i lepiej tego unikać.

Tablice

- Pamiętajmy, pilnowanie zakresu indeksów należy do nas. To nie jest wada C++. Jeżeli chcielibyśmy używać tablicy która sama pilnuje zakresu indeksów, to takie obiekty też istnieją w odpowiednich bibliotekach. Ale za taką dodatkową wygodę płacimy czasem dostępu od elementów. Jeżeli program ma sprawdzić indeks przed dostępem do pamięci, to zawsze trochę czasu to kosztuje.
- Indeksy tablicy mogą być dowolnymi zmiennymi typu całkowitego. Czasem wygodnie jest używać zmiennych, których nazwy coś znaczą. Na przykład

```
int konsultacje[5];
enum dni
{
    pon,
    wt,
    sr,
    czw,
    pt,
};
konsultacje[czw] = 11;
```

Tablice

- Pamiętamy, że C++ kolejne pozycje w zmiennej typu `enum` przypisuje kolejnym liczbom całkowitym począwszy od 0 (tak robi domyślnie), i można je traktować jako zmienne typu `int`.
- Dopóki do elementów tablicy nie zapiszemy swoich wartości, będą tam 0 (lub cokolwiek odpowiada samym 0 w typie danych tablicy. Program w ten sposób sam inicjalizuje wszystkie zmienne globalne. Konkretne wartości możemy do tablicy zapisać w momencie jej definicji, na przykład

```
int konsultacje[5] = {8,8,7,11,7};
```

- Jeżeli lista jest dłuższa niż rozmiar tablicy, kompilator zgłosi błąd. Jeżeli jest krótsza, nie ma problemu, pozostałe elementy tablicy pozostaną zerami. Jeżeli tablicę inicjalizujemy w ten sposób, możemy pominąc rozmiar, kompilator dopasuje rozmiar do ilości danych. Poniższa definicja jest równoważna powyższej:

```
int konsultacje[] = {8,8,7,11,7};
```

Tablice

- Tablicę można przesłać do funkcji jako argument. Jeżeli argumentem funkcji ma być tablica, której elementami są zmienne typu `int` to deklaracja funkcji wygląda następująco

```
void jakasFunkcja(int tab[]);
```

- Czasem w kwadratowym nawiasie podaje się rozmiar tablicy, to jest kwestia gustu, nie ma to znaczenia. Funkcja i tak nie sprawdza rozmiaru. Jeżeli wewnątrz funkcji potrzebny jest rozmiar przekazywanej tablicy, trzeba go przekazać oddzielnie. Na przykład:

```
void jakasFunkcja(int tab[], int rozmiar);
```

Tablice

- Wywołanie tak zadeklarowanej funkcji może wyglądać następująco:

```
jakasFunkcja(tab, rozmiar);
```

- Zauważmy, że przesyłamy tylko nazwę tablicy, bez kwadratowego nawiasu i bez indeksu. Tablice przekazywane są zawsze przez referencję. Przy przekazywaniu tablic nie trzeba ani nie można stosować operatora adresu &. Przekazanie jest zawsze przez referencję.
- Powodem tego jest przypuszczalnie fakt, że tablice często są duże, a dostęp potrzebny jest z reguły tylko do części elementów, więc szkoda czasu i stosu na kopiowanie. Jeżeli chcemy zapobiec przypadkowemu nadpisaniu elementów tablicy przez funkcję, stosujemy modyfikator `const`:

```
void jakasFunkcja(const int tab[]);
```

- Jeżeli funkcja będzie chciała nadpisać któryś element tak przekazanej tablicy, kompilator zgłosi błąd.

Tablice

- Oczywiście, poszczególne elementy tablicy możemy przekazywać do funkcji tak jak każdy inny argument, także przez wartość. Na przykład:

```
void jakasFunkcja(int element);
...
jakasFunkcja(tab[3]);
```

- Sama nazwa tablicy, bez indeksu, jest zmienną, która zawiera adres pierwszego elementu. Jeżeli więc zadeklarowaliśmy gdzieś tablicę tab, to wyrażenia

```
tab      oraz      &tab[0]
```

to jest to samo i zwracają adres pierwszego elementu.

Tablice

- Takie zmienne, zawierające adresy nazywamy wskaźnikami, i na następnym wykładzie przyjrzymy się im szczegółowo. Jednak nazwa `tab` nie jest zwykłym wskaźnikiem, bo nie można go zmienić. Jakiekolwiek podstawienie `tab = & x`, nawet jeżeli jest zgodne typem, to wywoła błąd kompilacji. Oczywiście tak powinno być.
- Wiedząc, że nazwa tablicy jest adresem jej pierwszego elementu możemy się domyślić, że poprawny jest rachunek adresów. Kompilator, wiedząc że zmienna pod danym adresem jest określonego typu, może policzyć, pod jakim adresem jest następna zmienna. `tab + 1` jest więc adresem drugiego elementu tablicy, i jest równy `&tab[1]` i tak dalej.

Tablice

- Możemy używać także tablic wielowymiarowych. Ich deklaracja jest zupełnie intuicyjna:

```
double tab[5][10];
```

  deklaruje tablicę dwuwymiarową tablicę zmiennych typu `double`, która ma 5 wierszy, i 10 kolumn. Do elementów odnosimy się tak jak intuicja każe: `tab[2][4]` to element tablicy w 3 wierszu i 5 kolumnie.
- Dwuwymiarowe tablice „układane" są w pamięci leksykograficznie, pierwszy wiersz, drugi wiersz, i tak dalej. Jeżeli wiemy, jaka jest długość wierszy (jaki jest pierwszy rozmiar tablicy) bez trudu odczytamy odpowiednie wartości.
- Na przykład, dla naszej tablicy `tab[roz1][roz2]` jeżeli napiszemy `tab[k][l]`, to jest to k*roz2+l -ty kolejny element tablicy zapisany linearnie w pamięci.

Tablice

- Zauważmy, że żeby odnaleźć ten element w pamięci, potrzebujemy jego oba indeksy k,l ale także drugi rozmiar `roz2`. Pierwszy rozmiar już nie jest potrzebny.
- Podobnie to działa dla tablic wyższych wymiarów. Żeby odnaleźć odnaleźć w pamięci element, potrzebujemy jego indeksów, oraz wszystkich wymiarów tablicy, z wyjątkiem pierwszego.
- Ma to wpływ na sposób deklaracji funkcji, jeżeli chcemy do niej przekazać tablicę jako argument. W przypadku tablicy 1 wymiarowej mówiliśmy, że jej rozmiar nie jest funkcji do niczego potrzebny, i nie musimy umieszczać go w deklaracji.

Tablice

- Jeżeli tab jest tablicą 1 wymiarową, to deklaracja funkcji, która przyjmuje tab jako argument wygląda następująco

```
void jakasFunkcja(int tab[]);
```

- Możemy w kwadratowym nawiasie wpisać rozmiar, ale nie musimy, i nie jest to w ogóle brane pod uwagę. Jeżeli natomiast tab jest tablicą, powiedzmy 3 wymiarową, to deklaracja funkcji musi wyglądać następująco

```
void jakasFunkcja(int tab[][roz2][roz3]);
```

- Pierwszy rozmiar nie jest funkcji potrzebny, ale pozostałe jak najbardziej. Jest to zupełnie jasne.

Tablice

- Jeżeli chcemy zainicjalizować tablicę wielowymiarową wraz z definicją, to kolejne wartości wpisujemy liniowo: najpierw pierwszy cały wiersz, potem drugi wiersz itp. Podobnie dla tablic więcej niż 2 wymiarowych. Kolejne elementy wpisujemy w takiej kolejności, że najszybciej rośnie indeks najbardziej na prawo.
- Tablice 1 wymiarowe są bardzo często stosowane, natomiast więcej niż 2 wymiarowe raczej rzadko.

Stringi

- Być może najważniejszym zastosowaniem tablic są tablice znakowe, czyli takie, których elementami są zmienne typu `char`. Czyli różnego rodzaju napisy. Będziemy na nie mówili „stringi”, jakkolwiek zabawnie może to brzmieć. Inne spotykane określenia to „łańcuchy znakowe”, „napisy” itp.
- Deklaracja stringu jest taka sama, jak każdej innej tablicy:

```
char napis[50];
```

- String możemy od razu zainicjalicować:

```
char napis[50] = "Witamy␣w␣Instytucie␣Matematycznym";
```

- Tablica `napis` będzie zawierała wyszczególnione znaki, i zera na pozostałych pozycjach. W przypadku tablic znakowych zawsze ostatnim elementem musi być 0. To nie jest liczba 0 (która ma kod ASCII 48), tylko wartość 0. W C++ ta wartość ma swoją nazwę, to jest `\0` lub `NULL`. String to jest więc ciąg znaków, kończący się znakiem `NULL`.

Stringi

- Stringi przechowywane są w tablicach znakowych, których rozmiar musi być co najmniej o 1 większy od przechowywanego stringu (musi być też zapisany kończący znak `NULL`). String może być krótszy, i wtedy program wie, gdzie się kończy dzięki kończącemu znakowi `NULL`.
- Wszystkie funkcje biblioteczne w C++ zakładają, że string kończy się znakiem `NULL`. Trzeba o tym pamiętać. Jeżeli do inicjalizacji stringu używamy tekstu w cudzysłowiu "...", to znak `NULL` zostanie na końcu dodany automatycznie.
- Pamiętajmy, długość stringu to jest ilość jego znaków, a rozmiar stringu (jako tablicy) jest zawsze o 1 większy.

Stringi

- Jeżeli string inicjalizujemy od razu w momencie definicji, najlepiej jest nie podawać rozmiaru. W takim przypadku, jak wiemy, kompilator sam dobierze rozmiar tablicy. Nie ma problemu z tym, że ewentualnie moglibysmy zapomnieć o dodatkowym znaku `NULL` potrzebnym do zakończenia.

```
char napis[] = "Witamy␣w␣Instytucie␣Matematycznym";
```

- Powstały string ma rozmiar 34 bajtów
- Podstawienie napisu do stringu jest możliwe tylko w momencie jego definicji. Powyżej napisana definicja jest jak najbardziej prawidłowa. Ale jeżeli chcielibyśmy ten string zmienić w dalszej części programu, na przykład gdzieś dalej pojawiłoby się podstawienie

```
...
napis = "Instytut␣Matematyczny␣żegna";
...
```

kompilator zgłosi błąd.

Stringi

- To jest dosyć oczywiste. Pamiętajmy, że nazwa tablicy jest adresem jej pierwszego elementu. Jest zmienną typu wskaźnik (za moment powiemy więcej o wskaźnikach), ale stałą, nie można jej zmienić. Istniejący string można nadpisać, ale dostęp do niego jest poprzez poszczególne znaki, tak jak dostęp do każdej tablicy. Możemy to zrobić następująco:

```
char napis2[] = "Instytut Matematyczny żegna";
int i = 0;
while ( napis2[i] != '\0' )
{
   napis[i] = napis2[i];
   i++;
}
```

- Tak napisana procedura wymaga jeszcze dopracowania, bo stringi mogą być różnej długości. Długość łatwo jest sprawdzić operatorem `sizeof()`. Oczywiście `sizeof()` zwróci rozmiar całej tablicy, która może być większa niż zawarty w niej string.

Stringi

- Uwaga: moglibyśmy próbować włączyć inkrementację `i` do któregoś z indeksów, i na przykład napisać `napis[i++]=napis2[i]`. W ten sposób zaoszczędzilibyśmy linijkę kodu. Taki manewr byłby jednak bardzo ryzykowny. Nie jestem pewien, czy kolejność ewaluacji wyrażeń jest ściśle określona w standardzie C++.
- Jeżeli chcemy wczytać string (na przykład, prosimy użytkownika o podanie nazwiska) robimy to następująco:

```
char napis[255]; // wpis ograniczony będzie do 254 znaków
cin >> napis;
```

- O pewnym szczególe należy pamiętać. Konsola pozwoli użytkownikowi wprowadzić napis ze spacjami. Konsola czeka na wciśnięcie `return`, i cały napis wysyła programowi. Ale do zmiennej `napis` podstawiona będzie tylko część stringu do pierwszej spacji. Reszta będzie czekała w buforze strumienia `cin`.

Stringi

- Kolejne elementy wpisanego stringu, do kolejnych spacji, moglibyśmy wczytywać w pętli. Byłby wtedy problem z wykryciem końca wejściowego stringu.
- Jeżeli chcemy pozwolić użytkownikowi wpisywać stringi ze spacjami, używamy konstrukcji

```
char napis[255]; // wpis ograniczony będzie do 254 znaków
cin.getline( napis, 254 );
```

- `getline` to funkcja związana ze strumieniem `cin` która cały wpisany string, do wciśnięcia `return` wysyła do tablicy `napis`. Drugi parametr to dodatkowe ograniczenie długości wysyłanego stringu. Pozwala to zabezpieczyć docelową tablicę `napis` przed przelewem.
- Strumienie takie jak `cin` są szczegółowo omawiane na Programowaniu II, obecnie wystarczy nam pamiętać o powyższym przykładzie. Wczytywanie stringów ze spacjami będzie potrzebne do zadania domowego.

Stringi

- Opisane powyżej stringi, które są tablicami zawierającymi znaki to tak zwane C - stringi. Były one podstawową strukturą danych do operacji na napisach, stosowaną w języku C.
- W C++ dodana została nowa struktura, tak zwana zmienna typu `string` dodaje wiele funkcjonalności. W szczególności zdejmuje z programisty obowiązki związane z kontrolą długości stringu. Jeżeli tablica znakowa jest za krótka do przechowania nowej wartości, programista posługujący się C - stringiem sam się musi martwić utworzeniem nowej, większej tablicy. Zmienna typu `string` w C++ sama się takimi sprawami zajmuje.
- Nie znaczy to, że C - stringi są przestarzałe. Działają bardzo szybko, i nie zużywają zasobów niepotrzebnie. Jeżeli nie jest to priorytetem w programie - możemy używać C++ - stringów (czyli zmiennych typu `string`).

Stringi

- Do operacji na C - stringach dysponujemy wieloma funkcjami bibliotecznymi. Główna biblioteka wymaga dołączenia pliku nagłówkowego `string.h`. Warto pamiętać o tej bibliotece, i zapoznać się z jej funkcjami.
- Przykładową funkcją z biblioteki `string.h` jest `strcpy()`, która kopiuje jeden string na drugi. Uwaga: ta funkcja nie pilnuje rozmiarów. Może przepełnić docelowy string. Jeżeli mamy taką potrzebę, rozmiary stringów możemy sprawdzać funkcją `strlen()`. Zwraca ona długość stringu (bez kończącego znaku `NULL`. Przypomnijmy, że `sizeof()` zwróci rozmiar tablicy znakowej, a nie długość zawartego w niej stringu, który może być krótszy.

Stringi

- Inną często stosowaną funkcją jest `strcmp()`, która porównuje dwa stringi (alfabetycznie). Zarówno ta funkcja jak i `strcpy()` mają swoje wersje, w których działanie ograniczone jest do określonej liczby znaków.
- Żeby użyć tej biblioteki, powinniśmy dołączyć plik nagłówkowy `string.h` lub `cstring`. Ten drugi jest uwspółcześnioną, dostosowaną do C++ wersją poprzedniego, chociaż oba powinny prawidłowo pracować. Mówiliśmy już o tym wcześniej, główną różnicą jest to, że w `cstring` wszystkie identyfikatory zdefiniowane sa w przestrzeni nazw `std`. Powinniśmy jeszcze pamiętać o pewnej subtelności. W C++ występuje jeszcze jedna biblioteka, której plik nagłówkowy to `string`. Ta biblioteka zawiera definicję zmiennej typu `string`. Wszystkie te nazwy plików nagłówkowych są podobne, i trzeba je rozróżniać.

Stringi

- Dodatkowe zamieszanie spowodowane jest tym, że niektóre kompilatory pozwalały wpisać nazwę pliku nagłówkowego bez rozszerzenia `.h`. W jakimś starszym programie może więc być dołączony plik `string`, gdzie tak naprawdę, potrzebny jest `string.h`. Trzymajmy się więc zasady: jeżeli używamy C - stringów, dołączajmy `cstring`. Jeżeli C++ - stringów, dołączajmy `string`.
- Zmienne typu `string` są szczegółowo omawiane na Programowaniu II.

Zadanie domowe 4 (do oddania 10.05.)

Napisz program który prosi o podanie stringu (nie dłuższego niż 254 znaki), następnie wypisuje go od końca, a także podaje ilość wyrazów w stringu. Zakładamy, że użytkownik nie stosuje polskich znaków, i nie wpisuje wielokrotnych spacji.
Program (podobnie jak projekty) proszę przesyłać prowadzącym ćwiczenia