Lekcja : Tablice + pętle

Wprowadzenie

Oczywiście wiesz już jak dużo można osiągnąć za pomocą tablic oraz jak dużo można osiągnąć za pomocą pętli, jednak tak naprawdę prawdziwe możliwości daje połączenie tablic i pętli i ich wspólne stosowanie w programach, bez dobrej znajomości pętli i tablic, dalsza nauka C++ będzie prawie niemożliwa.

Czas zacząć kojarzyć

Jeśli nie wiesz jeszcze, w jaki sposób połączyć wykorzystanie pętli w tablicach, postaraj się skojarzyć dwa proste fakty o których wspomniano.

Przede wszystkim jak już wiesz pętle umożliwiają powtarzanie pewnej operacji tak długo jak warunek końcowy pętli będzie spełniony. Wszystkie pętle posiadają zmienne (jedną lub kilka) sterujące. W bardzo łatwy sposób można zmieniać wartość zmiennej sterującej w kolejnych krokach pętli i bardzo łatwo stworzyć pętlę, która wykona określoną operację dla zmiennej sterującej z określonego przedziału.

Z drugiej strony, tablice umożliwiają przechowywanie zmiennych danego typu i dla zmiennej o nazwie **auto** typu tablicowego, poszczególne elementy tablicy to auto[0], auto[1], auto[2] itd.

Schematycznie zatem można zapisać postać dowolnego elementu tablicy jako **auto[indeks]**, gdzie indeks zmienia się od 0 do pewnej liczby określonej rozmiarem tablicy.

Jak zatem połączyć pętle z tablicami? Nie jest to takie trudne - zmienną indeksującą tablicy (czyli u nas nazwaną **indeks**) należy potraktować jako zmienną sterującą pętlą i zmieniać ją od **0** aż do **rozmiar tablicy -1** i dzięki temu, uda nam się dostać za pomocą jednej pętli do każdego elementu tablicy i dokonać na nim dowolnych operacji.

Przykłady

Na szczęście użycie tablic i pętli jest tak naprawdę bardzo łatwe. Teraz spróbujmy przedstawić różne przykłady, ilustrujące jak wiele można osiągnąć dzięki umiejętnemu posługiwaniu się poznanymi mechanizmami.

```
#include<iostream>
using namespace std;

int main ()
{

 int calkowite[3]={6,7,10};

 // wersja bez petli
 cout <<calkowite[0]<<' '<<calkowite[1]<<' '<<calkowite[2]<<'\n';

// wersja z uzyciem petli
 for (unsigned int i=0;i<3;++i)
   cout <<calkowite[i]<<' ';

 cout <<"\nNacisnij ENTER aby zakonczyc"<<'\n';
 getchar();
       return 0;
       }
```

Jak widzisz w powyższym przykładzie , w jaki sposób można wypisać wszystkie elementy tablicy. Co prawda w tym wypadku zysk jest niewielki - bowiem tablica ma tylko 3 elementy i w przybliżeniu napisanie kodu wypisującego elementy tej tablicy zajmuje tyle samo czasu, jednak zastanów się, która metoda byłaby wygodniejsza dla tablicy 100-elementowej.

Zastosowaliśmy tutaj przedstawione wcześniej podejście. Zmienna decydująca o tym, którym elementem tablicy się obecnie "zajmujemy" (tutaj oznaczona jako **i**, bo mamy zapis **calkowite[i]**), została wykorzystana jako zmienna sterująca pętlą. Wartość zmiennej sterującej zmienia się od 0 do 2, bowiem nasza tablica jest 3-elementowa.

Zapis przedstawiony w pętli nie jest tak naprawdę najszczęśliwszy, bo dlaczego w pętli warunkiem końcowym jest **<3**? Ty zapewne odpowiesz, że to przecież liczba elementów tablicy.

Co jednak jeśli za moment zmienisz liczbę elementów tablicy? Oczywiście tutaj należałoby również zmienić wartość.

Dodatkowo, gdyby taka pętla znajdowała się w znacznie odleglejszej części programu od miejsca, gdzie tablica zostaje utworzona, to wówczas w końcu sami byśmy zapomnieli, dlaczego warunek końcowy jest taki a nie inny.

Dlatego też w ogólnym wypadku lub w bardziej skomplikowanych przykładach, najczęściej wprowadzamy zmienną pomocniczą (najlepiej stałą, czyli z modyfikatorem **const**), którą następnie wykorzystujemy w warunku końcowym pętli. Dzięki temu zrozumienie warunku końcowego będzie łatwiejsze i nawet, gdy okaże się, że potrzebujemy mniejszej lub większej tablicy, wystarczy, że zmienimy tylko wartość zmiennej określającej rozmiar tablicy.

Wspomniany przed momentem sposób ilustruje poniższy przykład:

```
#include<iostream>
using namespace std;

int main ()
{

  const int ile=3; // rozmiar tablicy
  int calkowite[3]={6,7,10};

  for (unsigned int i=0;i<ile;++i)
     cout <<calkowite[i]<<' ';

  cout <<"\nNacisnij ENTER aby zakonczyc...\n";
  getchar();
return 0;
}
```

Dla formalności dodam, że tutaj zmienną określającą rozmiar tablicy, jest zmienna **ile** - jak widzisz, zastosowana nazwa zmiennej mówi dość dużo o jej przeznaczeniu.

Oczywiście możemy nie tylko wypisywać elementy tablicy. Możemy również je pobierać w ten łatwiejszy sposób oraz dokonywać pewnych modyfikacji na danych. Jak to zrobić dowiesz się przyglądając się poniższemu przykładowi:

```cpp
#include<iostream>
using namespace std;

int main ()
{

   const int ile=6; // rozmiar tablicy
   int calkowite[ile];
   unsigned int i; // zmienna sterujaca petlami
  // pobieranie elementow tablicy
  for (i=0;i<ile;++i)
  {
     cout <<"Podaj "<<i+1<<". element tablicy: ";
     cin >>calkowite[i];
     cin.ignore();
  }
  // wypisanie elementow tablicy
  cout <<"\nOto tablica: ";
  for (i=0;i<ile;++i)
     cout <<calkowite[i]<<' ';
  // dodanie liczby 2 do kazdego elementu tablicy
  for (i=0;i<ile;++i)
     calkowite[i]+=2;
  // ponowne wypisanie elementow tablicy
  cout <<"\nOto zmodyfikowana tablica: ";
  for (i=0;i<ile;++i)
     cout <<calkowite[i]<<' ';

  cout <<"\n\nNacisnij ENTER aby zakonczyc...\n";
  getchar();
return 0;
}
```

Jak widzisz, w powyższym przykładzie realizujemy wszystkie operacje, które bez użycia pętli nie byłyby już takie łatwe i szybkie. Dodatkowo zmienna sterująca pętlami została utworzona przed wszystkimi pętlami, dzięki czemu, program wykona się w rzeczywistości nieco szybciej.

Innym przykładowym programem może być obliczenie sumy wszystkich wartości występujących w tablicy. Oczywiście taką operację, jak i wszystkie inne można przeprowadzić za pomocą dowolnej pętli (tak jak wszystkie inne operacje), jednak teraz, żeby pokazać, że do przeglądania tablic można używać nie tylko pętli for, użyjemy dwóch pozostałych typów pętli.

```
#include<iostream>
using namespace std;

int main ()
{

  const int ile=6; // rozmiar tablicy
  float calkowite[ile]={3.45, 5, -10, 2.78, 4, 2.22};
  unsigned int i; //zmienna sterujaca petlami
  float suma_pocz, suma_kon;
  // suma liczona od "przodu" petla while
  i=0;
 suma_pocz=0;
  while (i<ile)
  {
    suma_pocz+=calkowite[i];
    ++i;
  }
  // suma liczona "od konca" petla do while
  i=ile;
  suma_kon=0;
  do
  {
    suma_kon+=calkowite[i];
    --i;
  }
```

```
 while (i>0);
 //tutaj i ma wartosc 0 i trzeba dodac do sumy calkowite[i]
 suma_kon+=calkowite[i];

 cout <<"Suma elementow liczona od poczatku petla while wynosi "<<suma_pocz;
 cout <<"\nSuma elementow liczona od konca petla do while wynosi "<<suma_kon;

 cout <<"\n\nNacisnij ENTER aby zakonczyc...\n";
 getchar();
return 0;
}
```

Jak widzisz, w powyższym programie obliczyliśmy sumę elementów tablicy dwoma metodami.

Wszystko powinno być dość łatwe do zrozumienia poza jedną kwestią, a mianowicie, dlaczego po pętli **do while** dodajemy do sumy jeszcze wartość początkowego elementu.

Zrobiliśmy tak dlatego, bowiem w pętli ograniczyliśmy ostatnią poprawną wartość pętli do 1 (warunek >0), co spowoduje, że ostatnią wartością zmiennej sterującej będzie 0. Gdybyśmy natomiast zmienili warunek, pisząc przykładowo **>=0**, wówczas otrzymalibyśmy opisywany w poprzednich lekcjach błąd zapętlenia w przypadku zmiennych z modyfikatorem **unsigned** - bo taki modyfikator ma właśnie zmienna **i**.

Oczywiście wystarczyło zastosować zmienną bez modyfikatora unsigned i byłoby po kłopocie.

Stosując jednak taką "sztuczkę", teoretycznie udało nam się zwiększyć zakres dopuszczalnych dodatnich wartości zmiennej sterującej (chociaż tutaj i tak z tego nie skorzystaliśmy).

Na koniec przedstawię przykład realizacji takiego zadania: zmień obecną tablicę tak, aby każdy nowy element był sumą dwóch dotychczasowych elementów: samego siebie i elementu go poprzedzającego. Inaczej mówiąc chcemy, żeby w elemencie o indeksie jeden, znalazła się suma elementu z indeksem jeden i z indeksem zero, a w ostatnim elemencie tablicy, suma ostatniego i przedostatniego elementu. Pierwszy element tablicy (element o indeksie zero) powinien pozostać taki sam, bowiem nie ma przed nim żadnego innego elementu.

Wykonując modyfikacje na tablicy, trzeba być bardzo ostrożnym. Tutaj właśnie dokonując operacji od początku tablicy do końca, nie udałoby się nam osiągnąć właściwego wyniku (bez użycia dodatkowej zmiennej), bowiem "niechcący" zamazywalibyśmy wcześniejsze wartości. Dlatego też w tym wypadku, aby realizacja postawionego problemu była łatwa, należy dokonywać operacje na tablicy od jej końca. Oto prosty przykład realizacji:

```
#include<iostream>
using namespace std;

int main ()
{

  const unsigned int ile=6; // rozmiar tablicy
  float tab[ile]={3.45, 5, -10, 2.78, 4, 2.22};

  cout <<"Poczatkowa tablica: ";
  for (unsigned int i=0;i<ile;++i)
     cout <<tab[i]<<' ';

  for (unsigned int i=ile-1;i>0;--i)
   tab[i]=tab[i]+tab[i-1];

 cout <<"\n\nZmodyfikowana tablica: ";
  for (unsigned int i=0;i<ile;++i)
   cout <<tab[i]<<' ';

  cout <<"\n\nNacisnij ENTER aby zakonczyc...\n";
  getchar();
return 0;
}
```

Podsumowanie

Tak naprawdę tablice wraz z pętlami są podstawową konstrukcją językową, która jest używana w wielu poważnych programach, dlatego też zrozumienie tego zagadnienia jest kluczowe dla dalszej Twojej nauki języka C++.