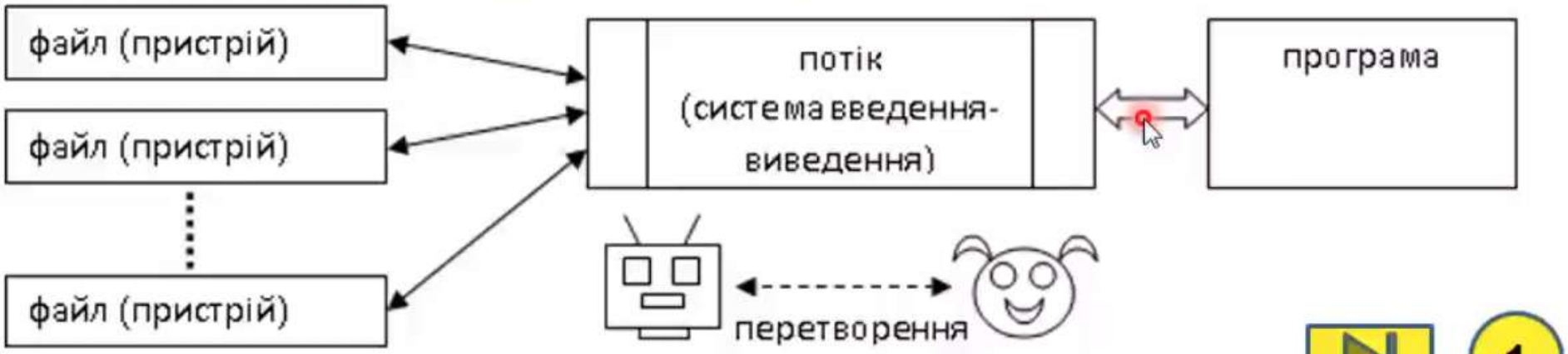
ТЕМА: Файлова система C++.

Потоки

Для більшості мов програмування засоби читання/друкування вбудовані в саму мову. Це є оператори Print, Read, Writeln тощо. Для мови C++ функції читання і друкування не є частиною мови. Читання і друкування здійснюється з допомогою класів, визначених в заголовних файлах `iostream` і `fstream`.

Програма на C++ подає введення та виведення як потік байтів. При введенні програма читає байти з потоку введення, а при виведенні вставляє байти в потік виведення. Байти потоку введення можуть поступати з клавіатури, диску, іншої програми тощо. Аналогічно байти потоку виведення можуть бути виведені на екран, друкуючий пристрій, диск, передані в іншу програму тощо. Головне завдання потоку – перетворити дані з внутрішньої форми величин пам'яті в текстову форму або навпаки. Загальна ідея потоку показана на рисунку.



Отже, керування введенням передбачає два кроки: приєднання потоку введення до програми; звязування потоку з файлом. Аналогічно - для виведення. Зазвичай введення та виведення можна реалізувати більш ефективно, використовуючи буфер - блок пам'яті як проміжну ланку зберігання даних.

Для роботи з потоками програмі потрібний заголовок (файл)

```
#include <iostream>
```

Включення в програму такого файла надає 4 потокові об'єкти:
`cin` – стандартне введення з клавіатури;
`cout` – стандартне виведення на екран;
`cerr` – стандартне виведення повідомлень про помилки;
`clog` – другий варіант повідомлень про помилки (буферизований).

За замовчуванням стандартні потоки взаємодіють з консоллю, проте їх можна перенаправити на інші пристрої чи файли.

Функції форматування виведення

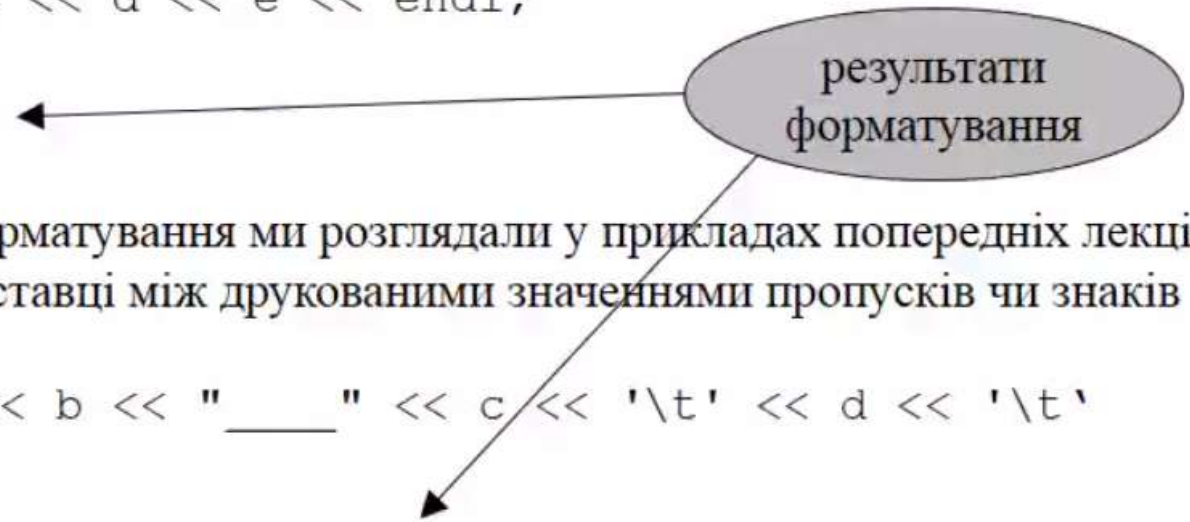
Мова C++ дозволяє виконати операції форматування введення-виведення, тобто, змінити форму зображення даних. Для форматування застосовують два подібні за змістом способи: 1) ознаки форматування і функції форматування; 2) маніпулятори – спеціальний різновид функцій. Ми розглянемо вибірково лише найчастіше вживані способи форматування.

У загальному випадку існують правила замовчування для форматування значень типів char, цілих, дійсних, рядків. Розмір поля для кожного значення відповідає цьому значенню:

```
int a=100, b=-25; float c=3.6502; char d='#'; char e[20]="test";
cout << a << b << c << d << e << endl;
```

Буде надруковано:

```
100-253.6502#test
```



Способи простого форматування ми розглядали у прикладах попередніх лекцій. Він полягає у «ручній» вставці між друкованими значеннями пропусків чи знаків табуляції:

```
cout << a << "__" << b << "____" << c << '\t' << d << '\t'
     << e << endl;
```

Буде надруковано:

```
100__-25____3.6502      #       test
```

Мова C++ має три функції форматування виведення (методи потоку `cout`).

`width()` - визначити ширину поля виведення даних:

```
int x=100, y=200, z=300;
cout.width(8);  cout << x;
cout.width(8);  cout << y;
cout.width(1);  cout << z;
cout << endl;
```

Буде надруковано:

```
_ _ _ _ _100_ _ _ _ _200300
```

Якщо поле більше від потреби, воно доповнюється пропусками, якщо ж менше – значення все одно друкується повністю, відкидання знаків немає.

Функція `width()` має дію лише на одне наступне значення, тому її треба виконати перед кожним окремим друкованим значенням.

`precision()` – для виведення дійсних чисел визначає кількість значущих цифр, тобто, точність зображення:

```
float fa=12.345666,  fb = 8.0 / 9.0;
cout << fa << "   " << fb << endl;
cout.precision(3);  cout << fa << "   " << fb << endl;
```

Буде надруковано:

```
12.3457   0.888889
12.3   0.889
```

Функція `precision()` має дію на всі наступні виведені значення (на відміну від `width`). За замовчуванням точність має 6 цифр.

`fill()` – символ-заповнювач невикористаних позицій. Якщо поле для виведення більше від потреби, воно автоматично доповнюється пропусками за замовчуванням. Ця функція дозволяє змінити символ-заповнювач:

```
int m=123,  p=456;  char f='W';  char u[15] = "abcdef";
cout.fill('*');
cout.width(6);  cout << m;  cout.width(8);  cout << p;
cout.width(4);  cout << f;  cout.width(10);  cout << u;
cout << endl;
cout.fill(' ');  // відновити стандартний заповнювач
```

Буде надруковано:

```
***123*****456***W****abcdef
```

Маніпулятори формату

Маніпулятори – це спеціальні функції, які включають в оператори введення-виведення. Маніпулятори виконують операції форматуваня даних. Приклади маніпуляторів (повний список і точне означення – за довідкою):

`boolalpha` – логічні значення зображати словами;
`noboolalpha` – логічні значення зображати числами 1 або 0;

```
bool tt=true, ff=false;
cout << boolalpha << tt << '\t' << ff << endl;
```

```
true    false
```

`dec`, `oct`, `hex` – зображення чисел в системі числення за основою 10, 8, 16 (лише для цілих);
`endl` – вивести символ переходу до нового рядка (це – маніпулятор);

```
int x=350,  y=-100;
cout << hex << x << '\t' << y << endl;
cout << dec << x << '\t' << y << endl;
cout << oct << x << '\t' << y << endl;
cout << dec;  // не забути відновити десяткову систему !
```

```
15e     ffffff9c
350     -100
536     37777777634
```

`scientific, fixed` – друкувати число у форматі науковому чи звичайному;

```
int temp=990033;  float sum=1234.5678;
 cout << fixed << temp << '\t' << sum << endl;
 cout << scientific << temp << '\t' << sum << endl;
```

```
990033  1234.567749
990033  1.234568e+003
```

`setw(int w), setprecision(int w), setfill(int ch)` – аналоги функцій форматування;

```
  float fa = 50.68 ;
  float fb = 9.0 / 13.0 ;
  cout << setprecision(3) << setw(10) << fa << setw(10)
       << fb << setw(10) << setfill('*') << fb << endl;
```

```
_ _ _ _ _ _50.7_ _ _ _ _0.692*****0.692
```

Зауваження щодо маніпуляторів:

1) для використання маніпуляторів з параметрами потрібно включити в програму директиву `#include <iomanip>`;

2) частина маніпуляторів має вплив лише на виведення, а частина – і на введення, і на виведення;

3) вплив маніпуляторів може бути постійним до явної заміни, а може – лише на одне наступне значення; інформація про це є в довідковій системі.

Робота з файлами: відкривання і закривання файлів

Загальна схема роботи з файлами є однаковою в алгоритмічних мовах.
1) Створити потік – файлову змінну. 2) Зв'язати потік з файлом – відкрити файл.
3) Виконати операції читання/запису. 4) Закрити файл – від'єднати від потоку.

Для реалізації файлового вводу-виводу в програму треба включити директиву `#include <fstream>`.

1) Створити потік. Існує три види потоків: введення, виведення, введення-виведення. Для створення відповідного потоку оголошують такі типи (класи):

```
ifstream namein;  // потік введення
ofstream nameout;  // потік виведення
fstream nameinout;  // потік введення-виведення
```

2) Відкрити файл. Для відкривання (зв'язування) є функція `open()`. В переважній більшості випадків параметром функції достатньо записати лише ім'я файла:

```
nameout.open("data.txt");
nameout.open("D:\\temp\\data.txt");
            // звернути увагу на подвійні дроби
```

Правила пошуку і розташування файлів у папках є однакові в межах операційної системи. Ми їх розглядали раніше при вивченні Python. Якщо шлях не записати, то файл має бути в поточній папці – там, де сама програма. Якщо записати повний шлях, тоді файл буде розташований за вказаною адресою у файловій системі.

Файл автоматично відкривається в потрібному режимі – введення, виведення, введення-виведення – відповідно до типу потоку.

В окремих випадках є потреба самому налаштувати спосіб роботи з файлом. В цьому разі функція `open()` має другий параметр – спосіб відкривання, наприклад:

```
nameout.open("data.txt", ios::out | ios::app);
```

Другий параметр може набувати таких значень:

```
ios::app        результати дописати в кінець файла виведення
ios::trunc      стерти попередній зміст файла, якщо він існує
ios::binary     відкрити файл в двійковому режимі
ios::in         відкрити файл для введення
ios::out        відкрити файл для виведення
```

Значення можна комбінувати логічною операцією «або».

Зауваження. Якщо є другий параметр, тоді треба перелічити всі ознаки відкривання операцією «або».

Найчастіше функцію `open()` не записують, а використовують конструктори, які автоматично відкривають файл:

```
ofstream nameout("data.txt");  // потік виведення
```

3) Операції читання/запису. Форма запису операцій залежить від виду файла: *текстовий* чи *двійковий*. Розглянемо далі.

4) Закрити файл. В кінці роботи програми кожний відкритий файл потрібно закрити функцією `close()`:

```
nameout.close();
```

Читання і запис текстових файлів

Читання і запис текстових файлів виконати дуже просто. Для цього використовують звичайні операції “>>” і “<<” видобування і вставки в потік, як це було для випадку консольного вводу-виводу. Замість потоків `cin` і `cout` тепер записуємо ім’я потоку, тобто, файлову змінну. Залишаються такими самими всі правила перетворення даних між внутрішньою машинною формою і текстовою формою, а також правила форматування.

Приклад роботи з файлами. У файлі `mtdata.txt` записана матриця розміру 4×5, наприклад:

```
2   5   -3   9   0
-4   12   10   0   6
4   7   8   3   -1
7   5   5   16   1
```

Прочитати матрицю, обчислити найменше і найбільше значення, записати результати у файл `minmaxdata.txt`. Вважати, що обидва файли є в поточній папці.

Файл з даними `mtdata.txt` можна надрукувати програмою `Блокнот` і записати в поточну папку проекту. Але краще залишатись в середовищі C++ і виконати команду

`Project -> Add New Item… -> Text File (.txt) -> Name: mtdata (без розширення)`

```
ifstream indata("mtdata.txt");  // потік введення
ofstream outres("minmaxdata.txt");  // потік виведення
float matrix[4][5];  // пам'ять для матриці
float min, max;
int i,j;
for(int i=0; i<4; i++)
  for(j=0; j<5; j++)
    indata >> matrix[i][j];

min = max = matrix[0][0];
for(int i=0; i<4; i++)
  for(j=0; j<5; j++)
  {
    if(matrix[i][j] < min)  min = matrix[i][j];
    if(matrix[i][j] > max)  max = matrix[i][j];
  }
// записати результати у вихідний файл
outres << min << "  " << max << endl;  // пробіли між числами !
indata.close();  outres.close();
cout << "Program execute" << endl;
```

Переглянути результати, записані у файл, можна за командою меню `Open File`:

```
-4  16
```

Двійковий ввід-вивід (безформатний)

Двійковий ввід-вивід є більш ефективним порівняно з текстовими файлами, бо немає перетворення даних між внутрішньою машинною формою і текстовою формою. Двійкові файли – це копії ділянок машинної пам'яті. Щоб виконати з файлом двійкові операції, файл треба відкрити в режимі `ios::binary`.

Найчастіше з двійковим файлом працюють на рівні блоків даних потрібного розміру. За одну операцію можна читати або писати зразу цілий блок:

```
read(адреса пам'яті, розмір блоку);
write(адреса пам'яті, розмір блоку);
```

У випадку послідовного доступу до файла всі операції вводу-виводу автоматично пересувають поточну позицію до наступного блоку.

Задача 1. Записати у двійковий файл параметри трьох академічних дисциплін.

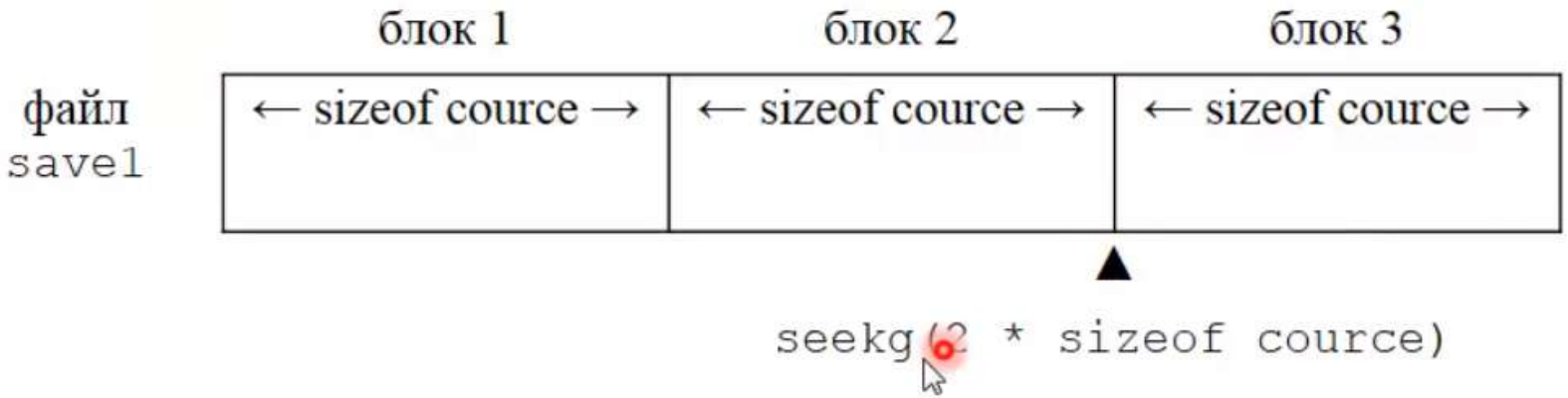
```
struct course  // академічна дисципліна
{
  char name[80];  // назва предмету
  int  less, prakt;  // кількість годин лекцій та практичних
  char annotation[120];  // анотація
  int  finalmark;   // оцінка за екзамен
} ;
// визначаємо деякі академічні дисципліни
course mathematics = { "Математичний аналіз", 32, 32,
                       "Вивчення за літературою", 74 };
course history = { "Історія України", 32, 16,
                   "Методична література", 82 };
course programming = { "Основи програмування", 32, 48,
                       "Індивідуальні завдання", 93 };
// запис блоків у файл
ofstream copydata("save1", ios::out | ios::binary);
                                    // файл двійкових даних
copydata.write( (char *) &mathematics, sizeof course);  // блок 1
copydata.write( (char *) &history, sizeof course);      // блок 2
copydata.write( (char *) &programming, sizeof course);  // блок 3
copydata.close();
cout << "Program execute" << endl;
```

Задача 2. Відшукати в записаному файлі `save1` *інформацію про курс програмування і надрукувати в текстовій формі в файл* `info.txt`.

У цьому випадку застосуємо прямий доступ до файла.

Прямий доступ забезпечує функція `seekg(відстань)`, яка у вказаній формі пересуває поточну позицію курсора файла на вказану відстань від початку файла. Наступна операція вводу або виводу буде виконана з поточної позиції курсора. Відстань вимірюють в байтах.

[Є ще форма `seekg(відстань, від чого)`, для інакшого обчислення відстані]



```
struct course  // академічна дисципліна
{
  char name[80];  // назва предмету
  int  less, prakt;  // кількість годин лекцій та практичних
  char annotation[120];  // анотація
  int  finalmark;   // оцінка за екзамен
} ;

course showdata; // визначаємо пам'ять для читання

ifstream academic("save1", ios::in | ios::binary); // файл дв.даних

// перемістити до потрібного блоку і прочитати
academic.seekg(2 * sizeof course);  // перемістити до блоку 3
academic.read( (char *) &showdata, sizeof showdata); // читати блок
academic.close();
ofstream review("info.txt"); // друкуємо зміст у текстовий файл
review << showdata.name << '\t' << showdata.less << '\t'
       << showdata.prakt << '\t' << showdata.annotation
       << '\t' << showdata.finalmark << endl;
review.close();
cout << "Program execute" << endl;
```

Отриманий текстовий файл `info.txt`:

```
Основи програмування 32  48  Індивідуальні завдання  93
```