ЛАБОРАТОРНА РОБОТА №2

Основні поняття про класи, методи класів, оператори new, delete в мові програмування С++

Мета: вивчити структуру класу мови С++, навчитися оголошувати класи, їх дані та методи, ознайомитися з поняттям об'єкту, навчитися вирішувати задачі шляхом застосування класів.

Теоретичні відомості

Поняття класу в С++

Клас являє собою головний інструментальний засіб С++ для об'єктно-орієтнтованого програмування. Клас дуже схожий на структуру, у якій згруповані елементи, що відповідають даним про деякий об'єкт, і даними функції, що оперують ними (методи). Об'єкт являє собою деяку сутність, наприклад телефон. Клас С++ дозволяє програмам визначати всі атрибути об'єкта. У випадку, коли об'єктом є телефон, клас може містити такі елементи даних, як номер і тип телефону (кнопкової або дисковий) і функції, що працюють із телефоном, наприклад dial, answer, і hang_up. Групуючи дані про об'єкт і кодуючи їх в одній змінній, спрощується процес програмування й збільшується можливість повторного використання свого коду. Основні концепції оголошення класу:

- для визначення класу програма повинна вказати ім'я класу, елементи даних класу й функції класу (методи).
- визначення класу забезпечує шаблон, за допомогою якого програми можуть створити об'єкти типу цього класу, подібно тому, як програми створюють змінні типу `int`, `char` і т.д.
- програма присвоює значення елементам даних класу, використовуючи оператор крапку.
- програма викликає функцію-елемент класу, використовуючи оператор крапку.

Поняття об'єкту та об'єктно-орієнтоване програмування

У загальному розумінні об'єкт являє собою сутність. Програма звичайно використовує змінні для зберігання інформації про різні реально існуючі сутності, наприклад службовців, книги і навіть файли. При об'єктно-орієнтованому програмуванні необхідно фокусуватися на предметах, що утворять систему, і операціях, які необхідно виконувати над цими предметами. Наприклад, для об'єкта-файлу можна мати операції, які друкують, відображають або змінюють файл. В С++

використовується клас для визначення своїх об'єктів. Мета полягає в тому щоб включити в клас стільки інформації про об'єкт, скільки потрібно. Виходячи із цього, можна підібрати клас, створений для однієї програми, і використати його в декількох різних програмах.

Клас дозволяє програмам групувати дані й функції, які виконують операції над цими даними. Більшість книг і статей про об'єктно-орієнтоване програмування називають функції класу методами. Подібно структурі, клас C++ повинен мати унікальне ім'я, за ним відкриваюча фігурна дужка, один або кілька елементів і закриваюча фігурна дужка:

```
class class_name
{
      int data_member; // Елемент даних
      void show_member(int); // Функція-елемент
};
```

Після визначення класу можна повідомляти змінні типу цього класу (які називаються об'єктами), як показано нижче:

```
class_name object_one, object_two, object_three;
```

Наступне визначення створює клас `employee`, що містить визначення даних і методів:

```
class employee
      {
         public:
         char name[64] ;
         long employee_id;
         float salary;
         void show_employee(void)
         {
            cout << "Ім'я: " << name << endl;
            cout   <<   "Номер   службовця:   "   <<
employee_id << endl;
            cout << "Оклад: " << salary << endl;
         };
      };
```

У цьому випадку клас містить три змінні і одну функцію-елементи. Елементи класу можуть бути приватними (`private`) або загальними (`public`), від чого залежить, як програми звертаються до елементів класу. У цьому випадку всі елементи є загальними, це означає, що програма може звертатися до будь-якого елемента, використовуючи оператор крапку. Після визначення класу всередині програми можна оголосити об'єкти (змінні) типу цього класу, як показано нижче:

```
      Ім'я класу
      employee worker,
```

```
boss, secretary; // Змінні класу (об'єкти)
```

Наступний приклад створює два об'єкти `employee`. Використовуючи оператор крапку, програма присвоює значення елементам даних. Потім програма використовує елемент `show_employee` для виводу інформації про службовця:

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

class employee
{
public:
   char name [64];
   long employee_id;
   float salary;
   void show_employee(void)
   {
      cout << "Ім'я: " << name << endl;
      cout <<"Номер службовця: "<<employee_id<<endl;
      cout << "Оклад: " << salary << endl;
   };
};
void main(void)
{
   setlocale(LC_ALL, "Ukrainian");
   employee worker, boss;
   strcpy(worker.name, "Інокентій Михтодійович");
   worker.employee_id = 12345;
   worker.salary = 25000;
   strcpy(boss.name, "Гаврило Тарасович");
   boss.employee_id = 101;
   boss.salary = 101101.00;
   worker.show_employee();
   boss.show_employee();
cin.get();
}
```

Програма оголошує два об'єкти типу `employee: worker` та `boss`, а потім використовує оператор крапку для присвоювання значень елементам і виклику функції `show_employee`.

Поняття об'єкта

Більшість програм на C++ описують реальні існуючі об'єкти. Об'єкти являють собою сутності, наприклад автомобіль, собаку, годинники й т.д. Звичайно об'єкт має кілька атрибутів й операцій, які програма повинна виконувати над цими атрибутами. Наприклад, у випадку годинника властивості можуть включати поточний час і час будильника. Операції такого об'єкта могли б містити встановлення часу, встановлення будильника або вимикання будильника.

Визначення методів класу поза класом

У попередньому класі `employee` функція була визначена всередині самого класу. При збільшенні функцій визначення вбудованих функцій всередині класу може внести безлад в опис класу. Як альтернатива можна помістити прототип функції всередині класу, а потім визначити функцію поза класом. Опис класу із прототипом стає наступним:

```
class employee
{
public:
  char name[64];
  long employee_id;
  float salary;
  void show_employee(void); // Прототип функції
};
```

Оскільки різні класи можуть використовувати функції з однаковими іменами, необхідно попередньо вказувати імена визначених поза класом функцій ім'ям класу і оператором глобального дозволу (::). У цьому випадку визначення функції стає наступним:

```
void employee:: show_employee (void) // Ім'я класу
      {
      cout <<"Ім'я: "<<name<<endl; //Ім'я елемента
      cout<<"Номер службовця: "<<employee_id<< endl;
      cout << "Оклад: " << salary << endl;
      };
```

Наведений код описується попередньо визначенням функції з ім'ям класу (`employee`) і оператором глобального дозволу (`::`). Наступний приклад демонструє визначення функції `show_employee` поза класом, використовуючи оператор глобального дозволу для вказання імені класу:

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

```
class employee
{
public:
   char name [64];
   long employee_id;
   float salary;
   void show_employee(void);
};
void employee::show_employee(void)
{
   cout << "Ім'я: " << name << endl;
   cout << "Номер службовця: " << employee_id <<
endl;
   cout << "Оклад: " << salary << endl;
};
void main(void)
{
   setlocale(LC_ALL, "Ukrainian");
   employee worker, boss;
   strcpy(worker.name, " Інокентій Михтодійович");
   worker.employee_id = 12345;
   worker.salary = 25000;
   strcpy(boss.name, " Гаврило Тарасович");
   boss.employee_id = 101;
   boss.salary = 101101.00;
   worker.show_employee();
   boss.show_employee();
cin.get();
}
```

Методи класу

Класи C++ дозволяють групувати дані об'єкта й функції об'єкта (методи), які оперують із цими даними, в одній змінній. Таким чином є дві можливості визначення методів об'єкта. Перший полягає в тому, що можна включити весь код функції всередині визначення класу. Незважаючи на те що включення коду методу у визначення класу може бути зручним, проте, коли класи стають складними і включають кілька методів, то оператори функцій можуть вносити безлад у визначення класів. Таким чином, багато програм визначають оператори функції поза класом. У визначення класу програма повинна включати прототип функції, що вказує ім'я функції, тип значення, що повертає, і типи параметрів.

Для визначення функції поза визначенням класу програма повинна випереджати визначення функції ім'ям класу й оператором глобального дозволу, як показано нижче:

```
return_type class_name::function_name(parameters)
{
// Оператори
}
```

Наступний приклад створює клас `dog`, що містить кілька полів даних і функцію `show_breed`. Програма визначає функцію класу поза визначенням самого класу. Потім програма створює два об'єкти типу `dog` і виводить інформацію про кожного собаку:

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

class dogs
{
public:
   char breed[64];
   int average_weight;
   int average_height;
   void show_dog(void) ;
 };

void dogs::show_dog(void)
{
   cout<< "Порода: " << breed << endl;
   cout<< "Середня вага: " << average_weight << endl;
   cout<< "Середня висота: "<<average_height << endl;
 }

void main(void)
{
setlocale(LC_ALL, "Ukrainian");
   dogs happy, matt;
   strcpy(happy.breed, "Долматин") ;
   happy.average_weight = 58;
   happy.average_height = 24;
   strcpy(matt.breed, "Колі");
   matt.average_weight =22;
   matt.average_height = 15;
   happy.show_dog() ;
```

```
   matt.show_dog();
   cin.get();
 }
```

Приватні та загальні дані

Мітка `public` при визначенні забезпечує програмі доступ до кожного елемента класу. Але для керування тим, як програми звертаються до елементів класу, C++ дозволяє визначати елементи як приватні (`privat`) або загальні (`public`).

Приватні елементи дають можливість класу сховати інформацію, яку програмі не потрібно знати. Клас, що використовує приватні елементи, забезпечує інтерфейсні функції, які звертаються до приватних елементів класу.

Приховування інформації

Класи містять дані і методи (функції). Для використання класу програми просто повинні знати інформацію, що зберігає клас (його елементи даних) і методи, які маніпулюють даними (функції). Програмам не потрібно знати, як працюють методи. Більше того, програми повинні знати тільки, які завдання виконують методи. Наприклад, нехай є клас `file`. В ідеалі програми повинні знати тільки те, що цей клас забезпечує методи `file.print`, що друкує відформатовану копію поточного файлу, або `file.delete`, що видаляє файл. Програмі не потрібно знати, як ці два методи працюють, тобто програма повинна розглядати клас як "чорний ящик". Програма знає, які методи необхідно викликати і які параметри їм передати, але програма нічого не знає про реальну роботу, що виконується всередині класу.

Приховування інформації являє собою процес, у результаті якого програмі надається тільки мінімальна інформація, необхідна для використання класу. Приватні й загальні елементи класу допомагають одержати інформацію, приховану всередині програми. Класи, що використовують мітку `public` для оголошення всіх елементів класу є загальними, тобто видимими для всієї програми. Таким чином, програма може безпосередньо звернутися до будь-якого елемента класу, використовуючи оператор крапку:

```
class employee
{
public:
   char name [64];
   long employee_id;
   float salary;
  void show_employee(void);
 }
```

При створенні класу можна мати елементи, значення яких використовуються тільки всередині класу, але звертатися до них самій програмі немає необхідності. Такі елементи є приватними (`private`), і їх варто приховувати від програми. Якщо мітка public не використовується, то за замовчуванням C++ має на увазі, що всі елементи класу є приватними. Програми не можуть звертатися до приватних елементів класу, використовуючи оператор крапку. До приватних елементів класу можуть звертатися тільки елементи самого класу. При створенні класу необхідно розділити елементи на приватні й загальні, як показано нижче:

```
class some_class
{
public:
   int some_variable;
   void initialize_private(int, float); //загальні
                                        //елементи
   void show_data(void);
 private:
   int key_value; // приватні елементи
   float key_number;
 }
```

Мітки `public` й `private` дозволяють визначати, які елементи є приватними, а які загальними. У цьому випадку програма може використовувати оператор крапка для звертання до загальних елементів, як показано нижче:

```
some_class object; // створити об'єкт
object.some_variable = 1001;
 object.initialize_private(2002, 1.2345);
 object.show_data()
```

Якщо програма намагається звернутися до приватних елементів `key_value` або `key_number`, використовуючи крапку, компілятор оголошує про синтаксичні помилки.

Захист елементів класу від прямого доступу до них здійснюється шляхом їх перетворенням з загальних (`public`) в приватні (`privat`). При цьому програми не можуть безпосередньо присвоювати значення таким елементам, використовуючи оператор крапку. Замість того щоб присвоїти значення, програма повинна викликати метод класу. Запобігаючи прямий доступ до елементів даних можна гарантувати, що їм завжди будуть присвоюватися припустимі значення. Наприклад, об'єкт `nuclear_reactor` використовує ім'я `melt_down`, яке завжди повинне містити значення в діапазоні від 1 до 5. Якщо елемент `melt_down` є

загальним, то програма може безпосередньо звернутися до елемента, змінюючи його значення довільним чином:

```
nuclear_reactor.melt_down = 101;
```

Якщо замість цього зробити змінну приватною, то можна використати метод класу, наприклад `assign_meltdown`, щоб присвоїти значення цієї змінної. Як показано нижче, функція `assign_meltdown` може перевіряти значення, що присвоюється, щоб переконатися, що воно є припустимим:

```
int nuke::assign_meltdown(int value)
{
   if ((value > 0) && (value <= 5))
   {
      melt_down = value;
      return(0); // успішне присвоювання
   } else return(-1); // неприпустиме значення
}
```

Методи класу, які керують доступом до елементів даних, являють собою інтерфейсні функції. При створенні класів можна використовувати інтерфейсні функції для захисту даних своїх класів.

Загальні й приватні елементи

Класи C++ містять дані й елементи. Щоб вказати, до яких елементів класів програми можуть звертатися прямо, використовуючи оператор крапку, C++ дозволяє визначати елементи класу як загальні (`public`) і привватні (`private`). Таким чином, програми можуть безпосередньо звертатися до будь-яких загальних елементів класу, використовуючи оператор крапка. З іншого боку, до приватних елементів можна звернутися тільки через методи класу. Як правило, необхідно захищати більшість елементів даних класу, оголошуючи їх приватними.

Наступний приклад ілюструє використання загальних і приватних елементів класу. Програма визначає об'єкт типу `employee` як показано нижче:

```
class employee
{
public:
  int assign_values(char *, long, float);
   void show_employee(void);
   int change_salary(float);
   long get_id(void);
 private:
   char name [64] ;
```

```
   long employee_id;
   float salary;
 }
```

Клас захищає всі свої елементи даних, оголошуючи їх приватними. Для доступу до елементів даних програма повинна використати інтерфейсні функції.

Приклад:

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

class employee
{
 public:
   int assign_values(char *, long, float);
   void show_employee(void);
   int change_salary(float);
   long get_id(void);
 private:
   char name [64];
   long employee_id;
   float salary;
};
int employee::assign_values(char *emp_name, long
                        emp_id, float emp_salary)
{
   strcpy(name, emp_name);
   employee_id = emp_id;
   if (emp_salary < 50000.0)
   {
      salary = emp_salary;
      return(0); // Успішно
    }
   else return(-1); // Неприпустимий оклад
}

void employee::show_employee(void)
{
   cout << "Службовець: " << name << endl;
   cout << "Номер службовця: " << employee_id <<endl;
   cout << "Оклад: " << salary << endl;
 }
```

```
int employee::change_salary(float new_salary)
{
   if (new_salary < 50000.0)
   {
      salary = new_salary;
    return(0); // Успішно
   }
   else return(-1); // Неприпустимий оклад
}

long employee::get_id(void)
{     return(employee_id) ;  }

void main(void)
{
   setlocale(LC_ALL, "Ukrainian");
   employee worker;
   if (worker.assign_values("Данило Охрімович", 101,
                                        10101.0) == 0)
     cout <<"Службовцеві призначені значення" <<endl;
     worker.show_employee();
     if (worker.change_salary(35000.00) == 0)
     {
        cout << "Призначений новий оклад" << endl;
        worker.show_employee();
      } else
   cout << "Зазначений неприпустимий оклад" << endl;
 cin.get();
}
```

Метод `assign_values` ініціалізує приватні дані класу. Метод використовує оператори `if`, щоб переконатися, що присвоюється припустимий оклад. Метод `show_employee` у цьому випадку виводить приватні елементи даних. Методи `change_salary` й `get_id` являють собою інтерфейсні функції, що забезпечують програмі доступ до приватних даних.

Використання оператора глобального дозволу для елементів класу

Імена параметрів функції в попередньому прикладі мають приставку `emp_`:

```
int   employee::assign_values(char   *emp_name,   long
emp_id, float emp_salary)
```

Дана приставка `emp_` використовувалася для уникнення конфлікту між іменами параметрів й іменами елементів класу. Якщо подібний конфлікт імен все ж відбувається, його можна уникнути шляхом використання оператора глобального дозволу `(::)`. Наступна функція використовує оператор глобального дозволу та ім'я класу перед ім'ям елементів класу. Виходячи із цього стає зрозумілим які імена відповідають класу `employee`:

```
int employee::assign_values(char *name, long
employee_id, float salary)
{
   strcpy(employee::name, name) ;
   employee::employee_id = employee_id;
   if (salary < 50000.0) { employee::salary = salary;
   return(0); // Успішно }
   else return(-1); // Неприпустимий оклад
}
```

При створенні функцій, що працюють із елементами класу необхідно використовувати ім'я класу і оператор глобального дозволу, щоб у такий спосіб уникнути конфлікту імен.

Примітка. При створенні функцій-елементів класу можливі ситуації, коли ім'я локальної змінної, яка використовується всередині функції, конфліктує з ім'ям елемента класу. За замовчуванням ім'я локальної змінної буде перевизначати ім'я елемента класу. Коли відбувається такий конфлікт імен, функція може використати ім'я класу і оператор глобального дозволу для доступу до елементів класу, як показано нижче:

```
class_naine::member_name = some_value;
```

Примітка. Вважається гарним тоном програмування здійснювати організацію проекта з використання класів у наступний спосіб:

- оголосити класи в окремому заголовочному файлі .h
- опис функцій-членів класу здійснювати в окремому .cpp файлі
- функцію main() з використанням класів та їх членів-даних та членів-функцій описувати в окремому .cpp файлі.

Попередній приклад матиме наступний вигляд.

Заголовочний файл з оголошенням класу cl.h:

```
class employee
{
 public:
   int assign_values(char *, long, float);
   void show_employee(void);
   int change_salary(float);
```

```
   long get_id(void);
 private:
   char name [64];
   long employee_id;
   float salary;
      };
```

Опис функцій-членів класу cl.cpp:

```
#include "cl.h"
#include <iostream>
#include <cstring>
using namespace std;

int employee::assign_values(char *emp_name, long
                        emp_id, float emp_salary)
{
   strcpy(name, emp_name);
   employee_id = emp_id;
   if (emp_salary < 50000.0)
   {
      salary = emp_salary;
      return(0); // Успішно
    }
   else return(-1); // Неприпустимий оклад
}

void employee::show_employee(void)
{
   cout << "Службовець: " << name << endl;
   cout << "Номер службовця: " << employee_id <<endl;
   cout << "Оклад: " << salary << endl;
 }

int employee::change_salary(float new_salary)
{
   if (new_salary < 50000.0)
   {
      salary = new_salary;
    return(0); // Успішно
   }
   else return(-1); // Неприпустимий оклад
}
```

```
long employee::get_id(void)
{     return(employee_id);  }
```

Функція main() з використанням класів, їх членів-даних та членів-функцій f.cpp:

```
#include <iostream>
#include <cstring>
#include "cl.h"
using namespace std;

void main(void)
{
   setlocale(LC_ALL, "Ukrainian");
   employee worker;
   if (worker.assign_values("Данило Охрімович", 101,
10101.0) == 0)
     cout <<"Службовцеві призначені значення" <<endl;
     worker.show_employee();
     if (worker.change_salary(35000.00) == 0)
     {
        cout << "Призначений новий оклад" << endl;
        worker.show_employee();
      } else
   cout << "Зазначений неприпустимий оклад" << endl;
 cin.get();
}
```

Використання вільної пам'яті в C++

Оператор C++ new дозволяє програмам виділяти пам'ять під час виконання. Для використання оператора new необхідно вказати кількість байтів пам'яті, яка потрібна програмі. Припустимо, наприклад, що програмі необхідний 50-байтний масив. Використовуючи оператор new, можна замовити цю пам'ять, як показано нижче:

```
char *buffer = new char[50];
```

Якщо оператор new успішно виділяє пам'ять, він повертає покажчик на початок області цієї пам'яті. У цьому випадку, оскільки програма виділяє пам'ять для зберігання масиву символів, вона присвоює, покажчик змінної, як визначений покажчик на тип char. Якщо оператор new не може виділити заданий обсяг пам'яті, він поверне Null-покажчик, який містить значення 0. Щоразу, коли програми динамічно виділяють пам'ять із використанням оператора new, вони повинні перевіряти, що вертається оператором new для визначення, чи не рівно значення NULL.

Замість редагування й перекомпілювання програм, які просто запитують пам'ять із запасом, слід створювати програми таким чином, щоб вони виділяли необхідну їм пам'ять динамічно під час виконання, використовуючи оператор new. У цьому випадку програми можуть адаптувати використання пам'яті відповідно до потреб, що змінилися, уникаючи необхідності редагувати й перекомпілювати програму.

Наприклад, наступна програма використовує оператор new для одержання покажчика на 100-байтний масив:

```
#include <iostream>
using namespace std;
void main(void)
{
   char *pointer = new char[100];
   if (pointer != NULL) cout << "Пам'ять успішно
виділена" << endl;
   else cout << "Помилка виділення пам'яті" << endl;
   cin.get();
}
```

Дана програма відразу перевіряє значення, присвоєне оператором new змінній-покажчику. Якщо покажчик містить значення NULL, то це означає, що оператор new не зміг виділити необхідний обсяг пам'яті. Якщо ж покажчик містить не NULL, то оператор new успішно виділив пам'ять і покажчик містить адреса початку блоку пам'яті. Якщо new не може задовольнити запит на виділення пам'яті, то він поверне NULL

При використанні оператора new для виділення пам'яті може трапитися так, що запит не може бути задоволений, оскільки немає достатнього обсягу вільної пам'яті. Якщо оператор new не здатний виділити необхідну пам'ять, він присвоює покажчику значення NULL. Перевіряючи значення покажчика, як показано в попередній програмі, можна визначити, чи був задоволений запит на пам'ять. Наприклад наступний оператор, використовує new для видялення пам'яті під масив з 500 значень із плаваючою крапкою:

```
float *array = new float[100];
```

Щоб визначити, чи виділив оператор new пам'ять, програма повинна зрівняти значення покажчика з NULL, як показано нижче:

```
if (array != NULL) cout << "Пам'ять виділена успішно"
<< endl;
   else cout <<"new не може виділити пам'ять"<< endl;
```

Попередня програма використовувала оператор new для виділення 100 байт пам'яті. Оскільки ця програма "жорстко закодована"

на обсяг необхідної пам'яті, можливо, буде потрібно її редагувати й перекомпілювати, якщо виникне необхідність, щоб програма виділила менше або більше пам'яті. Одна із причин динамічного виділення пам'яті полягає в тому, щоб позбутися необхідності редагування та перекомпіляції програму при зміні вимог до обсягу пам'яті. Наступна програма запитує в користувача кількість байт пам'яті, яку необхідно виділити, і потім виділяє пам'ять, використовуючи оператор `new`:

```
#include <iostream>
using namespace std;
void main(void)
{
   int size;
   char *pointer;
   cout << "Уведіть розмір масиву, до 30000: ";
   cin >> size;
   if (size <= 30000)
   {
      pointer = new char[size];
      if (pointer != NULL) cout << "Пам'ять виділена
успішно" << endl;
      else cout<<"Неможливо виділити пам'ять"<<endl;
   }
   cin.get();
}
```

Коли програми використовують оператор `new` для динамічного виділення пам'яті, то часто відомо, скільки пам'яті необхідно виділити. Наприклад, якщо програма виділяє пам'ять для зберігання інформації про службовців, вона зберегла кількість службовців у файлі. При запуску вона може прочитати кількість службовців з файлу, а потім виділити відповідну кількість пам'яті.

Наступний приклад виділяє щоразу пам'ять для 10000 символів доти, поки оператор `new` не зможе більше виділити пам'ять із вільної доступної пам'яті. Вона утримує виділену пам'ять, поки не використовує всю доступну вільну пам'ять. Якщо програма успішно виділяє пам'ять, вона видає про це повідомленням. Якщо пам'ять більше не може бути виділена, то програма виводить повідомлення про помилку і завершується:

```
#include <iostream>
using namespace std;
void main(void)
{
   char * pointer;
```

```
   do
   {
      pointer = new char[10000];
      if (pointer != NULL) cout << "Виділено 10000
байт" << endl;
      else cout << "Більше немає пам'яті" << endl;
   } while (pointer != NULL);
   cin.get();
}
```

Вільна пам'ять

При кожному запуску програми компілятор C++ встановлює окрему область не використовуваної пам'яті, яка називається вільною пам'яттю. Використовуючи оператор `new`, програма може виділити пам'ять із цієї вільної пам'яті під час виконання. Використовуючи вільну пам'ять для виділення необхідної пам'яті, програми не обмежені фіксованими розмірами масивів. Розмір вільної пам'яті може змінюватися залежно від операційної системи й моделі пам'яті компілятора. Якщо збільшується кількість динамічної пам'яті, необхідної для програми, необхідно переконатися, що відсутнє обмеження вільної пам'яті системи.

Оператор C++ `new` дозволяє програмам виділяти пам'ять динамічно під час виконання. Якщо програмі більше не потрібна виділена пам'ять, вона повинна її звільнити, використовуючи оператор `delete`. Для звільнення пам'яті з використанням оператора `delete` необхідно вказати цьому оператору покажчик на дану область пам'яті, як показано нижче:

```
delete pointer;
```

Наступна програма використовує оператор `delete` для звільнення виділеної за допомогою оператора `new` пам'яті:

```
#include <iostream>
#include <cstring>
using namespace std;
void main(void)
{
   char *pointer = new char[100];
   strcpy(pointer, "Вчимося програмувати мовою C++");
   cout << pointer << endl;
   delete pointer;
   cin.get();
}
```

За замовчуванням, якщо програма не звільняє виділену їй пам'ять до свого завершення, операційна система автоматично звільняє цю

пам'ять після завершення програми. Однак якщо програма використовує оператор `delete` для звільнення пам'яті в міру того, як вона (пам'ять) стає непотрібною, то ця пам'ять знову стає доступною для інших цілей (можливо, для програми, яка знову буде використовувати оператор `new`, або для операційної системи).

Приклад. Програма виділяє пам'ять для зберігання масиву з 1000 цілочисельних значень. Потім вона заносить в масив значення від 1 до 1000, виводячи їх на екран. Потім програма звільняє цю пам'ять і виділяє пам'ять для масиву з 2000 значень із плаваючою крапкою, заносячи в масив значення від 1.0 до 2000.0:

```
#include <iostream>
using namespace std;
void main(void)
{
   int *int_array = new int[1000];
   float *float_array;
   int i;
   if (int_array != NULL)
   {
      for(i = 0;i < 1000;i++) int_array[i] = i+1;
      for(i = 0;i < 1000;i++) cout<<int_array[i]<<"";
      delete int_array;
   }
   float_array = new float[2000];
   if (float_array != NULL)
   {
      for (i =0;i<2000;i++) float_array[i]=(i+1)*1.0;
      for (i = 0; i < 2000; i++) cout <<
float_array[i] << " " ;
      delete float_array;
   }
   cin.get();
}
```

Як правило, програми повинні звільняти пам'ять за допомогою оператора `delete` у міру того, як пам'ять стає непотрібною.

Динамічна пам'ять та класи

Розглянемо створення об’єкту в динамічній пам’яті – екземпляр створеного класу. Оскільки звернення відбувається через вказівник, то доступ до даних членів-функцій виконується через оператор непрямого доступу “ ->”

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

class SimpleCat
{
public:
  int itsAge;
  int GetAge()
  {         return itsAge;  }

 void SetAge(int age)
 {    itsAge=age; }
};

void main()
{
SimpleCat *Frisky = new SimpleCat;
Frisky->SetAge(5);
cout<<Frisky->itsAge<<endl;
cout<<Frisky->GetAge()<<endl;
cin.get();
}
```

Проте наступний приклад демонструє інкапсуляцію – приховування даних – неможливість доступу до прихованих даних.

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

class SimpleCat
{
public:
    int GetAge()
  {         return itsAge;  }

 void SetAge(int age)
 {    itsAge=age; }
private: int itsAge;
};

void main()
{
SimpleCat *Frisky = new SimpleCat;
Frisky->SetAge(5);
```

```
cout<<Frisky->itsAge<<endl; // Помилка доступу до
                            // закритих даних
cout<<Frisky->GetAge()<<endl;
cin.get();
}
```

Ще один приклад використання операторів new() та delete():

```
#include <iostream>
using namespace std;
class Distance
{
private:
int feet;
float inches;
public:
void GetDist()
{
cout <<"Введіть фути"<<endl;
cin >> feet;
cout <<"Введіть дюйми"<<endl;
cin >> inches;
}
void ShowDist()
    {
      cout << feet << endl;
    cout<< inches<<endl;
    }
};
void main()
{
setlocale(LC_ALL,"Ukrainian");
Distance Dist; //створення обєкту типу Distance
Dist.GetDist(); //отримання даних
Dist.ShowDist(); //відображення даних
//створення вказівника на об'єкт типу Distance
Distance* pDist;
pDist = new Distance(); // створення об'єкту Distance
pDist->GetDist(); //отримання даних
pDist->ShowDist();//відображення даних
if (pDist) delete pDist; //звільнення пам'яті
cin.get();
cin.get();
}
```

ЗАВДАННЯ ДО ЛАБОРАТОРНОЇ РОБОТИ №2

Перший рівень

1. Визначити користувацький клас згідно варіанта завдань (не менше 5 полів).
2. Визначити в класі компонента-функції для запису та перегляду полів даних.

Другий рівень

1. Визначити користувацький клас згідно варіанта завдань (не менше 5 полів).
2. Визначити в класі компонента-функції для запису у файл та читання з файлу полів даних.
3. У програмі необхідно передбачити розміщення об'єктів як у статичної, так й у динамічній пам'яті, а також створення масивів об'єктів.
4. Показати в програмі використання покажчика на об'єкт і покажчика на екземпляр класу.
5.Програма використає три файли: заголовний h-файл із визначенням класу, cpp-файл із реалізацією класу, cpp-файл демонстраційної програми.

Третій рівень

1. Визначити користувацький клас згідно варіанта завдань (не менше 8 полів).
2. Клас повинен містити 3 загальні та 5 приватні елементи. Відобразити використання усіх полів даних в методах.
3. В класі визначити 4 методи, які опрацьовують описані дані (два методи мають бути перевантаженими).
4. Визначити в класі методи для запису у файл та читання з файлу полів даних.
5. У програмі побудувати 5 об'єктів, розміщених у статичній пам’яті, та 5 об’єктів - у динамічній пам'яті.
6. В основній програмі визначити 2 масиви об'єктів. Продемонструвати роботу з об’єктами, розміщеними в даному масиві.
7. Визначити додатковий метод в класі, який виділяє випадкову величину динамічної пам’яті. В пам’яті розмістити множину значень одного з цілочисельних полів та відсортувати їх.
8. Показати в програмі використання покажчика на об'єкт і покажчика на екземпляр класу.

9.Програма використає три файли: заголовний h-файл із визначенням класу, cpp-файл із реалізацією класу, cpp-файл демонстраційної програми.

Варіанти завдань.

1. КАДРИ	2. СТУДЕНТ	3. СЛУЖБОВЕЦЬ
4. ІСПИТ	5. ВИРІБ	6. БІБЛІОТЕКА
9. КВИТАНЦІЯ	10. АДРЕСА	11. ТОВАР
12. АВТОМОБІЛЬ	13. ЦЕХ	14. ПЕРСОНА
15. КОРАБЕЛЬ	16. КРАЇНА	17. ТВАРИНА
18. КНИГА	19. ПК	20. ПРИНТЕР
21. РОБІТНИК	22. ВЕЛОСИПЕД	23. МОБІЛЬНИЙ ТЕЛЕФОН
24. ЛІФТ	25. ПЛАНШЕТ	26. БУДІВЛЯ
27. ПРОЦЕСОР	28. КРОСІВКИ	29. ФЛЕШКА
30. УНІВЕРСИТЕТ	31. ВІНЧЕСТЕР	32. ФОТОАПАРАТ
7. ДИСЦИПЛІНА		
8. ФАКУЛЬТЕТ		

Додаток. *Приклади підказки*

1. Приклад визначення класу.

```
const int LNAME=25;
class STUDENT{
char name[LNAME]; // ім'я
int age; // вік
float grade; // рейтинг
int GetAge();
float GetGrade();
void SetName(char*);
void SetAge(int);
void SetGrade(float);
void Set(char*,int,float);
void Show();
};
```

2. Приклади розміщення об'єктів у статичної та динамічній пам'яті, а також створення масивів об'єктів.

```
STUDENT gruppa[3];
gruppa[0].Set(“Іванов”,19,50);
STUDENT gruppa[3]={STUDENT(“Іванов”,19,50),
STUDENT(“Петрова”,18,25.5), STUDENT(“Сидоров”,18,45.5)};
STUDENT *p; p=new STUDENT [3];
p-> Set(“Іванов”,19,50);
```