Лабораторна робота №

Динамічне виділення пам'яті і виключення в C ++

Мета роботи:

1) вивчити менеджер пам'яті C ++

2) вивчити різні способи обробки виключень;

3) отримати практичні навички програмування задач з виділенням пам'яті і обробкою винятків.

ТЕОРЕТИЧНІ ВІДОМОСТІ

Виділення пам'яті

У мові програмування C ++ оператор new забезпечує виділення динамічної пам'яті в купі. new намагається виділити достатньо пам'яті в купі для розміщення нових даних і, в разі успіху, повертає адресу пам'яті. Однак, якщо new не може виділити пам'ять в купі, то він генерує (throw) виключення типу std::bad_alloc. Це усуває необхідність явної перевірки результату виділення.

Синтаксис new виглядає наступним чином: **p_var = new typename;**

де **p_var** - раніше оголошений покажчик типу typename. typename може бути будь-яким фундаментальним типом даних або визначеним користувачем типом (включаючи, enum, class і struct). Якщо typename - це тип класу або структури, то він повинен мати доступний конструктор за замовчуванням, який буде викликаний для створення об'єкта.

Для ініціалізації нової змінної, створеної за допомогою new потрібно використовувати наступний синтаксис:

p_var = new type (initializer);

де initializer - початкове значення, присвоєне нової змінної, а якщо type - тип класу, то initializer - аргумент (и) конструктора.

new може також створювати масив**:**

p_var = new type [size];

В даному випадку, size вказує розмірність (довжину) створюваного одновимірного масиву. Адреса першого елемента повертається і поміщається в p_var, тому p_var [n] означає значення n-го елемента (рахуючи від нульової позиції)

Пам'ять, виділена за допомогою new, повинна бути звільнена за допомогою delete, щоб уникнути витік пам'яті. Масиви, виділені (створені) за допомогою new[], повинні звільнятися (знищуватися) за допомогою delete[].

int * p_scalar = new int (5);

int * p_array = new int [5];

Ініціалізатор не можуть бути вказані для масивів, створених за допомогою new. Всі елементи масиву ініціалізуються за допомогою конструктора за замовчуванням для даного типу. Якщо тип не має конструктора за замовчуванням, виділена область пам'яті не буде проініціалізувати.

Існує особлива форма оператора new, звана як Placement new. Даний оператор не виділяє пам'ять, а отримує своїм аргументом адресу на вже виділену якимось чином пам'ять (наприклад, в стеку або через malloc). Відбувається розміщення (ініціалізація) об'єкта шляхом виклику конструктора, і об'єкт створюється в пам'яті за вказаною адресою. Часто такий метод застосовують,

коли у класу немає конструктора за замовчуванням і при цьому потрібно створити масив об'єктів.

Приклад виклику виглядає наступним чином:

```
class A {
public:
  A (int x) {}
  ~ A () {}
};
const int n = 50;
A * placementMemory = static_cast <A*> (operator new[] (n*sizeof (A)));
for(int i = 0; i <n; i ++) {
new (placementMemory+i) A (rand ());
// Тут пам'ять для об'єкта не виділяється, але ініціалізовуються
}
// !! деініціалізацію пам'яті
for (int i = 0; i <n; i ++){
placementMemory[i].~A();
}
operator delete [] (placementMemory);
```

Оскільки при виділенні пам'яті тип створюваного об'єкта (ів) не зазначалася, компілятор не викликатиме деструктор для кожного об'єкта масиву, тому це потрібно зробити вручну, перед звільненням блоку пам'яті.

Перевірка виділення пам'яті

У компіляторах, які дотримуються стандарту ISO C++, в разі якщо недостатньо пам'яті для виділення, то генерується виключення типу std::bad_alloc. Виконання всього подальшого коду припиняється, поки помилку не буде оброблена в блоці try-catch або станеться екстрене завершення програми. Програма не потребує перевірки значення покажчика; якщо не було згенеровано виняток, то виділення пройшло успішно. Реалізовані операції визначаються в заголовку <new>. У більшості реалізацій C ++ оператор new також може бути перевантажений для визначення особливого поведінки.

Звільнення пам'яті

У мові програмування C ++ оператор delete (або delete []) повертає пам'ять, виділену оператором new, назад в купу. Виклик delete повинен відбуватися для кожного виклику new, щоб уникнути витоку пам'яті. Після виклику delete об'єкт, який вказує на цю ділянку пам'яті, стає некоректним і не повинен більше використовуватися. Багато програмістів привласнюють 0 (нуль-покажчик) вказівниками після використання delete, щоб мінімізувати кількість помилок програмування. Однак потрібно зазначити, що видалення нуль-покажчика фактично не має ефекту, так що немає необхідності перевіряти нуль-покажчик перед викликом delete.

Фрагмент коду в якості прикладу:

int * p_var = NULL; // Оголошення нового покажчика
p_var = new int; // Пам'ять динамічно виділяється
/ *решта коду * /
delete p_var; // Пам'ять звільняється
p_var = NULL; // Покажчик замінюється на 0 (нуль-покажчик)

Масиви, створені (виділені) за допомогою new [], аналогічним чином повинні бути знищені (освобождени) за допомогою delete []:

int size = 10;
int * p_var = NULL; // Оголошення нового покажчика
p_var = new int [size]; // пам'ять динамічно виділяється
/ *решта коду * /
delete [] p_var; // Пам'ять звільняється
p_var = NULL; // Покажчик замінюється на 0 (нуль-покажчик)

Виклик delete [] для масиву об'єктів призведе до виклику деструктора для кожного об'єкта перед звільненням пам'яті, виділеної під масив.

Винятки в C ++

Винятки - виникнення непередбачених помилкових ситуацій, наприклад ділення на нуль при операціях з плаваючою точкою. Зазвичай ці умови завершують програму користувача з системним повідомленням про помилку. Обробка винятків в C ++ дає можливість програмісту відновлювати програму з цих умов і продовжувати її виконання.

Мова C ++ має чутливий до контексту механізм обробки особливих ситуацій.

Контекст для установки виключення - це блок try.

Обробники оголошені в кінці блоку try з використанням ключового слова catch.

Простий приклад:

```
vect::vect (int n){
  if (n <1) throw (n);
  p = new int [n];
  if (p == 0)
        throw ( "FREE STORE EXHAUSTED");
}
void g (){
  try {
        vect a(n), b(n);
        }
  catch (int n) {...} // відстежує всі неправильні розміри
  catch (char * error) {...} // відстежує перевищення вільної пам'яті
  }
```

Встановлені винятки

Синтаксично вираження throw виникає в двох формах:

throw;

throw вираз;

Вираз throw встановлює виняток. Вираз throw без аргументу повторно встановлює поточний виняток. Зазвичай воно використовується, коли для подальшої обробки виключення необхідний другий обробник, що викликається з першого.

```
void foo (){
  int i;
  throw (i);
}
main (){
  try {
        foo ();
  }
  catch (int i) {...}
}
```

Якщо користувач хоче виводити додаткову інформацію або використовувати її для прийняття рішення щодо дій обробника, то допустимо формування у вигляді об'єкта.

```
enum error {bounds, heap, other};
class vect_error {
  private:
        error e_type;
        int ub, index, size;
  public:
        vect_error (error, int, int); // Пакет поза заданих меж
        vect_error (error, int); // Пакет поза пам'яті
}
```

Тепер вираз throw може бути більш інформативним

```
…
throw vect_error(bounds, i, ub);
…
```

Блок try

Синтаксично блок try має таку форму

```
try
  складовий оператор
  список обробників
```

Блок try - контекст для прийняття рішення про те, які обробники викликаються для встановленого виключення.

```
try {
  …
  throw ( "SOS");
  …
  io_condition.eof (argv [i]);
  throw (eof);
  …
```

}
catch (const char *) {...}
catch (io_condition & x) {...}
Вираз throw відповідає аргументу catch, якщо він:
- точно відповідає.
- загальний базовий клас породженого типу являє собою те, що встановлюється.
- об'єкт встановленого типу є типом покажчика, що перетворюються в тип покажчика, що є аргументом catch.

Обробники catch

Синтаксично обробник catch має наступну форму catch (формальний аргумент) складовою оператор

```
catch (char * message){
  cerr << message << endl;
}
catch (...){// дію за замовчуванням
cerr << "THAT'S ALL FOLKS." <<endl;
abort ();
}
```

Специфікація виключення

Синтаксис

Заголовок_функції throw (список типів)

void foo () throw (int, over_flow);

void noex (int i) throw ();

Terminate() і unexpected()

Оброблювач terminate() викликається, коли для обробки винятку не поставлений інший обробник. За замовчуванням викликається функція abort().

Оброблювач unexpected() викликається, коли виключення не було в списку специфікації виключення

Приклад коду, що реалізує виняток

Приклад 1.

```
#include "vect.h"
void g (int n){
try {
// Блок try - контекст для прийняття рішення про те, які
// обробники викликаються для встановленого виключення vect a (n);
}
catch (int n) { // обробник
  cerr << "SIZE ERROR" << n << endl; g (10);
}
catch (const char * error) { // обробник
  cerr << error << endl; abort ();
}
catch (std :: bad_alloc x){
```

```
cerr << "out of memory" << endl;
}
}
void main (){
extern void g (int n); g (-1);
}
```

Файл vect.h:

```
#include <iostream.h>
class vect {
  private:
        int * p;
        int size;
  public:
        vect () {size = 11; p = new int [size];}
        vect (int n);
        ~vect () {delete [] p; }
        int& element (int i);
        int ub() const {return (size-1); }
  };
vect::vect (int n){
  if (n <1) throw (n); // Встановлюється виняток
  p = new int [n];
  if (p == 0)
       throw ( "FREE STORE EXHAUSTED"); // Встановлюється виняток для старих компіляторів
}
int& vect::element (int n){
  if (n <0 || n> size-1)
     throw("ILLEGAL NUMBER OF ELEMENT");//Встановлюється виняток
  return (p [n]);
}
```

Контрольні питання

1. Яку мету переслідує використання в програмі обробки винятків?
2. Як оформляється блок обробки винятків?
3. Що таке обробники винятків?

Варіанти завдань

1 Реалізуйте клас для зберігання цілих чисел без знака. Реалізуйте метод множення двох цілих. Згенеруйте і обробіть помилку переповнення зверху (overflow).

2 Реалізуйте клас для зберігання цілих чисел без знака. Реалізуйте метод складання двох цілих. Згенеруйте і обробіть помилку переповнення зверху (overflow).

3 Реалізуйте клас «двійкове дерево». Як ключ для розподілення даних використовуйте цілі числа, як самих даних - речові числа. Обробіть помилки динамічного виділення пам'яті. Перевизначите оператор[] для доступу до даних

дерева за значенням ключа, Згенеруйте і обробіть помилку відсутності потрібних елементів в дереві.

4 Реалізуйте клас «масив цілих чисел». Обробіть помилки динамічного виділення пам'яті. Перевизначите оператор ++ для покажчика на масив цілих, обробіть помилку виходу за межі масиву.

5 Реалізуйте клас для зберігання номера телефону. Обробіть помилки динамічного виділення пам'яті. Обробіть помилки створення номера в невірному форматі (допустимий формат - «+38 (095) 555-44-33».

6 Реалізуйте клас «список цілих чисел». Обробіть помилки динамічного виділення пам'яті. Перевантажте оператор [] і обробіть помилку виходу за межі списку.

7 Реалізуйте клас для зберігання дати. Опишіть метод, який повертає день тижня. Обробіть помилки динамічного виділення пам'яті. Обробіть помилки невірного дня чи місяця.

8 Реалізуйте клас для зберігання особистих даних (ПІБ). Обробіть помилки динамічного виділення пам'яті. Обробіть помилки завдання ПІБ (допустимий формат - «тільки букви, без деяких букв спочатку слова».

9 Реалізуйте клас «просунутий покажчик на об'єкт». Реалізуйте оператор "стрілка". Згенеруйте і обробіть помилки динамічного виділення пам'яті і помилки звернення за нульовим вказівником.

10 Реалізуйте клас для зберігання ланцюгових дробів (список цілих чисел що не повторюються). Опишіть методи для вставки нового числа і знаходження і-го числа. Згенеруйте і обробіть помилки динамічного виділення пам'яті і вставки невірного числа.

11 Реалізуйте клас «черга» з цілих чисел. Опишіть оператор [] для черги елементів, обробіть помилку виходу за межі черги.

12 Реалізуйте клас «бітова карта» на основі цілих чисел. Опишіть оператор [] для визначення і-го біта. Згенеруйте і обробіть помилку виходу за межі карти.

13 Реалізуйте клас для зберігання масиву географічних координат. Згенеруйте і обробіть помилки динамічного виділення пам'яті і помилки завдання невірної широти і / або довготи.

14 Реалізуйте клас «черга» з рядків. Реалізуйте методи для вставки в чергу і видалення. Згенеруйте і обробіть помилки динамічного виділення пам'яті, переповнення черги.

15 Реалізуйте клас для зберігання мережевого адреси в форматі IPv4. Обробіть помилки динамічного виділення пам'яті. Обробіть помилки завдання адреси (допустимий формат - чотири числа від 0 до 255, розділені крапками, виключаючи деякі неприпустимі комбінації.