**Указатель this**

При обращении к переменным-членам класса из кода, находящегося вне класса, всегда указывается объект класса, к которому эти переменные относятся. Таким образом, компилятор всегда может определить, с какой копией переменных-членов класса надо работать.

При обращении к переменной-члену класса *внутри функции-члена* класса экземпляр класса не указывается, так как компилятор передает функции-члену класса скрытый указатель на объект. Обратиться явно к скрытому указателю можно с использованием предопределенного указателя ***this***. Это указатель, который автоматически передается любой *функции-члену* при ее вызове и указывает на объект, генерирующий вызов.

**Замечание**: Указатель this передается только функциям-членам. Дружественным функциям указатель this не передается.

**Пример**:

#include<iostream>

using namespace std;

int iVar = 5;

class myclass {

int iVar;

public:

void show(int j);

};

void myclass::show(int k) {

this->iVar = k;

cout << "iVar(global)=" << ::iVar << endl;

cout << "iVar(local)=" << iVar << endl;

cout << "iVar=" << this->iVar << endl;

}

int main()

{

myclass ob;

ob.show(10);

system("pause");

return 0;

}

**Статические, постоянные и модифицируемые члены класса**

В языке Си достаточно часто используются глобальные переменные. Однако, использование их в языке C++ практически нарушает один из основных принципов OOП – принцип инкапсуляции. Таким образом, использование в C++ статических переменных – членов класса является альтернативой к использованию глобальных переменных. Статическая переменная – член класса, по сути, является глобальной переменной, область видимости которой ограничена классом, в котором она определена. Отметим также, что доступ к ней может быть управляемым: открытым, закрытым или защищенным.

Обычно каждый объект класса имеет собственную копию переменных-членов класса. Однако, если переменная объявлена с использованием спецификатора *static*, существует только единственный экземпляр этой переменной класса, независимо от числа созданных объектов класса. Каждый объект класса, совместно с другими, использует эту переменную. Кроме этого, одна и та же статическая переменная – член класса, используется всеми классами, производными от класса, в котором статическая переменная определена.

Для полного определения статической переменной – члена класса необходимо:

1. объявить статическую переменную внутри класса;
2. определить и инициализировать её вне класса, используя операцию расширения области видимости.

Выполнение второго пункта связано с тем, что статическая переменная не принадлежит какому-то конкретному объекту класса, а всему классу. Статическую переменную (функцию) вызывают с использованием операции расширения области видимости, без указания какого-либо объекта класса.

Статические функции-члены класса могут *непосредственно* ссылаться только на статические переменные и статические функции, принадлежащие ее классу.

У статической функции-члена нет указателя this, они не могут быть виртуальными, не могут объявляться со спецификатором const и volatile (переменный).

**Пример**:

#include<iostream>

using namespace std;

class myclass {

static int i;

public:

myclass(int j) { i = j; }

myclass() {}

void show() { cout << "i=" << i << endl; }

static int show\_static() { return i; }

};

int myclass::i = 5;

int main()

{

cout << "i=" << myclass::show\_static() << endl;

myclass ob1(10), ob2;

ob1.show();

ob2.show();

myclass ob3(15);

cout << "i=" << myclass::show\_static() << endl;

system("pause");

return 0;

}**Результат выполнения программы**:

i=5

i=10

i=10

i=15

**Комментарий к программе**: Переменная класса i существует еще до создания объекта класса (инициализация вне класса) и уже может быть использована с применением статической функции. Объект ob2 не инициализировал переменную i, но, так как она статическая, то у нее есть значение.

Функции – члены класса могут быть объявлены постоянными с использование спецификатора **const.**

Функция - член объявляется постоянной, если необходимо предотвратить изменение вызвавшего ее объекта.

Постоянный объект не может вызвать непостоянную функцию-член, но постоянная функция-член может вызываться как постоянными, так и не постоянными объектами.

Ключевой слово *const* стоит за списком параметров функции.

Если все-таки необходимо, чтобы функция-член, оставаясь постоянной, все же могла изменить один или несколько членов класса, то используют ключевое слово **mutable**, для модифицирования членов класса. Модифицированный член класса можно изменить с помощью постоянной функции-члена.

**Пример**: // *Объявление постоянных функций-членов*

#include <iostream>

using namespace std;

class myclass {

// int i;

mutable int i;

public:

int geti() const {

return i;

}

void seti(int x) const {

i = x;

}

};

int main()

{

myclass ob;

ob.seti(1900);

cout << ob.geti() << endl;

system("pause");

return 0;

}

**Результат выполнения программы**:

1900

**Операторы динамического выделения памяти new и delete**

В C++ для динамического выделения памяти используется два оператора: **new** и **delete.**

Формат этих операторов:

pVar=new type;

delete pVar;

Здесь type – спецификатор типа объекта, для которого выделяется память, а pVar – указатель на этот тип;

*new* – оператор, который возвращает указатель на динамически выделяемую память, достаточную для хранения объекта типа *type;*

*delete* – оператор, который освобождает эту память.

Если памяти не хватает, то *new* возвращает нулевой указатель, либо будет сгенерирована исключительная ситуация.

Рассмотрим инициализацию динамически размещаемого объекта и создание динамически размещаемых массивов объектов.

Формат:

pVar=new type(начальное значение);

Для массива:

pVar=new type [size];

delete [] pVar;

**Пример**:// *Динамическое выделение памяти*

#include<iostream>

#include<cmath>

using namespace std;

class compl0{

float re;

float im;

public:

compl0(float x, float y) : re(x), im(y) {}

float mod() { return sqrt(re\*re + im \* im); }

};

class compl1 {

float re;

float im;

public:

void set\_xy(float x, float y) { re = x; im = y; }

~compl1() { cout << "Delete ..." << endl; }

float mod() { return sqrt(re\*re + im \* im); }

};

int main()

{

// переменные

int n;

cout << "Input n" << endl;

cin >> n;

// одна переменная

float \*p\_var1;

p\_var1 = new float;

\*p\_var1 = 125;

cout << "\*p\_var1=" << \*p\_var1 << endl;

// одна переменная с инициализацией

float \*p\_var2;

p\_var2 = new float(25);

cout << "\*p\_var2=" << \*p\_var2 << endl;

// динамический массив

float \*p\_var3;

p\_var3 = new float[n];

for (int i = 0; i < n; i++)

p\_var3[i] = i \* i;

for (int i = 0; i < n; i++)

cout << "\*p\_var3=" << p\_var3[i] << endl;

// объекты

// один объект с инициализацией

compl0 \*p\_ob1;

p\_ob1 = new compl0 (4, 5);

cout << p\_ob1->mod() << endl;

// массив объектов

compl1 \*p\_ob2;

p\_ob2 = new compl1[n];

for (int i = 0; i < n; i++)

p\_ob2[i].set\_xy(i, i + 2);

for (int i = 0; i < n; i++)

cout << p\_ob2[i].mod() << endl;

delete p\_var1;

delete p\_var2;

delete[] p\_var3;

delete p\_ob1;

delete[] p\_ob2;

return 0;

}

**Пример**:// *Работа с двумерными динамическими массивами*

#include<iostream>

using namespace std;

int main()

{

int i, j, i1, m, n, k;

cout << "Number of str: \n"; // число строк матрицы A

cin >> m;

cout << " Number of col: \n"; // число столбцов матрицы A

cin >> n; cout << " Number of col1: \n"; // число столбцов матрицы B

cin >> k;

float \*\*A, \*\*B, \*\*C;

A = new float \*[m];

B = new float \*[n];

C = new float \*[m];

for (i = 0; i < m; i++) // Массив данных

{

A[i] = new float[n];

C[i] = new float[k];

}

for (i = 0; i < n; i++) // Массив данных

{

B[i] = new float[k];

}

for (i = 0; i < m; i++)

for (j = 0; j < n; j++)

A[i][j] = i + j;

for (i = 0; i < n; i++)

for (j = 0; j < k; j++)

B[i][j] = i + j;

for (i = 0; i < m; i++)

for (j = 0; j < k; j++)

{

C[i][j] = 0;

for (i1 = 0; i1 < n; i1++)

C[i][j] += A[i][i1] \* B[i1][j];

}

for (i = 0; i < m; i++) {

for (j = 0; j < k; j++) {

cout << C[i][j] << " ";

}

cout << endl;

}

system("pause");

return 0;

}

**Указатели на члены класса и объекты.**

В языке C++ существует особый тип указателя, который ссылается на член класса вообще, а не на конкретный объект класса. Его называют *указателем на член класса*. Этот указатель задает смещение внутри объекта класса. Чтобы обратиться к члену класса с помощью указателя на него, надо применить операторы: ".\*" и "->\*".

**Замечание**: Указатели на члены класса в повседневном программировании практически не используются.

**Пример**: // *Указатели на члены класса (оператор ".\*")*

#include<iostream>

using namespace std;

class myclass {

public:

int j;

myclass(int i) : j(i) {}

int sq() { return j \* j; }

};

int main()

{

int myclass::\*d;

int (myclass::\*f)();

myclass ob1(3), ob2(5);

d = &myclass::j;

f = &myclass::sq;

cout << "j1=" << ob1.\*d << " j2=" << ob2.\*d << endl;

cout << "sq1=" << (ob1.\*f)() << " sq2=" << (ob2.\*f)() << endl;

system("pause");

return 0;

}

**Пример**: // *Указатели на члены класса (оператор "->\*")*

#include<iostream>

using namespace std;

class myclass {

public:

int j;

myclass(int i) : j(i) {}

int sq() { return j \* j; }

};

int main()

{

int myclass::\*d;

int (myclass::\*f)();

myclass ob1(3), ob2(5);

myclass \*p1, \*p2;

p1 = &ob1;

p2 = &ob2;

d = &myclass::j;

f = &myclass::sq;

cout << "j1=" << p1->\*d << " j2=" << p2->\*d << endl;

cout << "sq1=" << (p1->\*f)() << " sq2=" << (p2->\*f)() << endl;

system("pause");

return 0;

}

Для доступа к членам класса через указатель на объект используется оператор "->", а не оператор" ." .

**Пример**: // *Указатели на объект*

#include<iostream>

using namespace std;

class myclass {

public:

int j;

myclass(int i) : j(i) {}

int sq() { return j \* j; }

};

int main()

{

myclass ob1(3), ob2(5);

myclass \*p1, \*p2;

p1 = &ob1;

p2 = &ob2;

cout << "j1=" << p1->j << " j2=" << p2->j << endl;

cout << "sq1=" << p1->sq() << " sq2=" << p2->sq() << endl;

system("pause");

return 0;

}

Все три рассмотренные выше программы выполняют одни и те же действия.

***Замечание:*** Настроить вывод в Visual Studio (консольное окно)

Отладка -> Имя проекта: свойства -> система -> подсистема -> консоль (/Subsystem:console)