```
/*
 * advanced programming
 * Alireza Akhavan Pour
 * akhavan@alirezaWeb.com
 * date: 1395/02/29
 */
int main()
{
      cout<< "In the name of God";
      Lecture_25 ();
      return 0;
}

string Lecture_25()
{
      cout << "Linked list with iterator" <<endl;
      cout << "Template" <<endl;
      return ":)"
}
```

1

دلیل استفاده از کلاس‌های تو در تو (nested class)

http://stackoverflow.com/questions/4571355/why-would-one-use-nested-classes-in-c

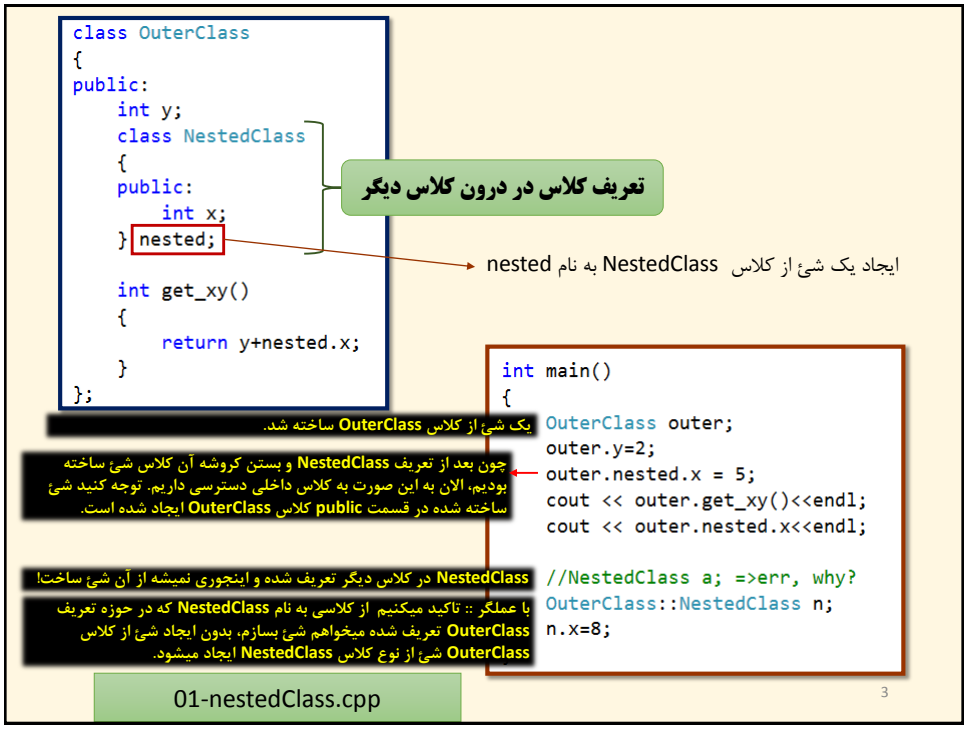
- **Can someone please point me towards some nice resources for understanding and using nested classes?... I'm still having trouble understanding their purpose?**

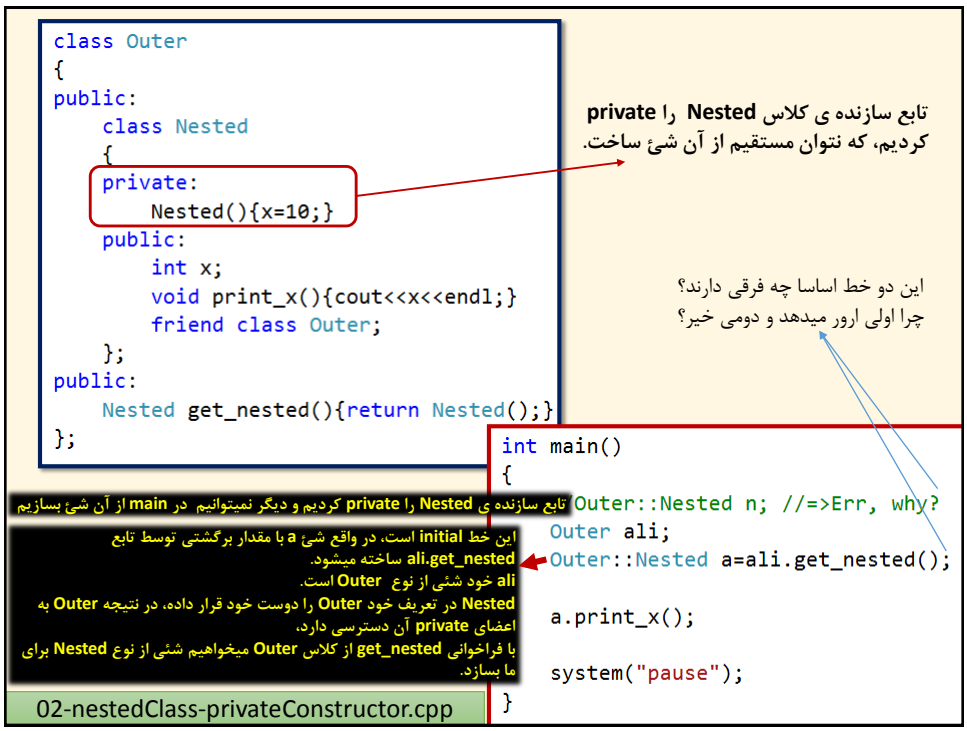
✓Nested classes are cool for hiding implementation details

- **اساسا چرا ما کلاسی را باید در کلاس دیگر تعریف کنیم؟**

✓برای مخفی سازی پیاده سازی کلاس؛ برای مثال در مثال لیست استفاده کننده ی کلاس نیازی ندارد در مورد پیاده سازی node اطلاعی داشته باشد. (در راستای **encapsulation**)

2


class OuterClass
{
public:
int y;
class NestedClass
{
public:
int x;
} nested;
int get_xy()
{
return y+nested.x;
}
};
تعریف کلاس در درون کلاس دیگر
ایجاد یک شئ از کلاس NestedClass به نام nested
int main()
{
OuterClass outer;
outer.y=2;
outer.nested.x = 5;
cout << outer.get_xy()<<endl;
cout << outer.nested.x<<endl;
//NestedClass a; =>err, why?
OuterClass::NestedClass n;
n.x=8;
یک شئ از کلاس OuterClass ساخته شد.
چون بعد از تعریف NestedClass و بستن کروشه آن کلاس شئ ساخته بودیم، الان به این صورت به کلاس داخلی دسترسی داریم. توجه کنید شئ ساخته شده در قسمت public کلاس OuterClass ایجاد شده است.
NestedClass در کلاس دیگر تعریف شده و اینجوری نمیشه از آن شئ ساخت!
با عملگر :: تاکید میکنیم از کلاسی به نام NestedClass که در حوزه تعریف OuterClass تعریف شده میخواهم شئ بسازم، بدون ایجاد شئ از کلاس OuterClass شئ از نوع کلاس NestedClass ایجاد میشود.
01-nestedClass.cpp
3


class Outer
{
public:
class Nested
{
private:
Nested(){x=10;}
public:
int x;
void print_x(){cout<<x<<endl;}
friend class Outer;
};
public:
Nested get_nested(){return Nested();}
};
تابع سازنده ی کلاس Nested را private کردیم، که نتوان مستقیم از آن شئ ساخت.
این دو خط اساسا چه فرقی دارند؟
چرا اولی ارور میدهد و دومی خیر؟
int main()
{
Outer::Nested n; //=>Err, why?
Outer ali;
Outer::Nested a=ali.get_nested();
a.print_x();
system("pause");
}
تابع سازنده ی Nested را private کردیم و دیگر نمیتوانیم در main از آن شئ بسازیم
این خط initial است، در واقع شئ a با مقدار برگشتی توسط تابع ali.get_nested ساخته میشود.
ali خود شئی از نوع Outer است.
Nested در تعریف خود Outer را دوست خود قرار داده، در نتیجه Outer به اعضای private آن دسترسی دارد.
با فراخوانی get_nested از کلاس Outer میخواهیم شئی از نوع Nested برای ما بسازد.
02-nestedClass-privateConstructor.cpp

خلاصه Nested Class

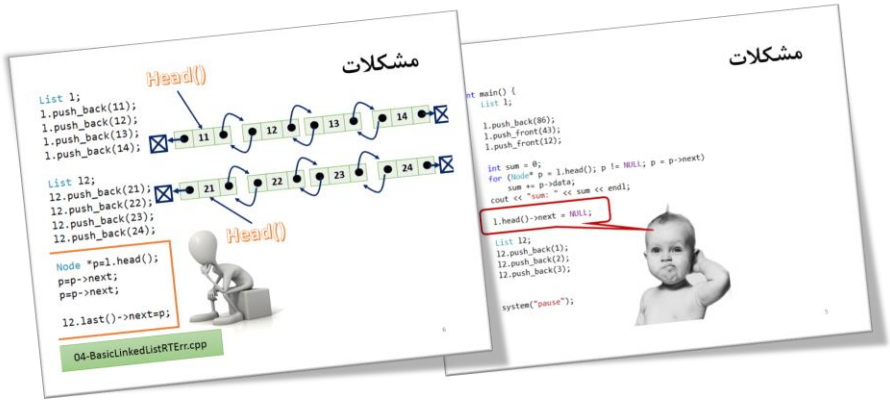
- اساسا چرا یک کلاس را به صورت nested و درون کلاس دیگر تعریف میکنیم؟
- اگر تابع سازنده ی کلاسی private باشد چه میشود؟
- تفاوت (**تعریف متغیر و assignment**) با (**initial**) چیست؟

Rectangle a=b; //initialization **VS** Rectangle a;//تعریف متغیر
a=b; //assignment

- در سی پلاس پلاس (برخلاف جاوا و ...) حتی اگر کلاسی درو کلاس دیگر تعریف شده باشد، کلاس دربرگیرنده به اعضای خصوصی کلاس nested دسترسی نخواهد داشت و اگر میخواهم دسترسی بدهیم باید آن را friend تعریف کنیم.

رفع مشکلات linked list جلسه قبل!

در جلسه‌ی قبل دیدیم که دسترسی به عناصر یک لیست پیوندی از طریق دسترسی آزاد به Nodeهای آن مشکل‌ساز است. این مشکل از طریق **تکرارکننده‌ها** حل می‌شود. در این مثال دسترسی به عناصر لیست پیوندی از طریق تکرارکننده مهیا می‌شود.



03-IteratorLinkedList.cpp

```
class List {
private:
    class Node {
    public:
        Node(int d, Node *n = NULL, Node *p = NULL)
            : data(d), next(n), prev(p) {}

        int data;
        Node *next;
        Node *prev;
    };
public:
    class Iterator {
    public:
        int next_element() {
            int to_be_returned = current->data;
            current = current->next;
            return to_be_returned;
        }
        bool has_more_elements() {
            return current != NULL;
        }
    private:
        Node *current;
        Iterator(Node* n) { current = n; }
        friend class List;
    };
            ...
```

این کلاس در داخل کلاس List و در بخش خصوصی آن تعریف شده است. به همین دلیل، سایر کلاس‌ها نمی‌توانند به آن دسترسی پیدا کنند.

این کلاس داخل کلاس List و به شکل عمومی تعریف شده. به همین دلیل دسترسی به آن برای بقیه‌ی کلاس‌ها ممکن است.

برگرداندن عنصر جاری و رفتن به عنصر بعدی

آیا عنصر دیگری وجود دارد؟

عنصری جاری

سازنده‌ی کلاس به شکل خصوصی تعریف شده. به همین دلیل کسی نمی‌تواند از این کلاس شیء بسازد ...

... مگر کلاس Listکه دوست این کلاس است!

7

```
class List {
    ...

public:
    List();
    ~List();
    void print();
    void push_front(int x);
    void push_back(int x);
    void clear();
    Iterator get_iterator() {
        return Iterator(_head);
    }
private:
    Node* _head;
    Node* _last;
};
```

برگرداندن یک iterator جدید که به ابتدای لیست اشاره می‌کند.

8

```cpp
int main() {
    List l;

    l.push_back(86);
    l.push_front(43);
    l.push_front(12);

    l.print();
    cout << endl;

    int sum = 0;

    List::Iterator it = l.get_iterator();
    while (it.has_more_elements())
        sum += it.next_element();

    cout << sum << endl;
}
```

چون Iterator داخل کلاس List تعریف شده، بیرون از آن کلاس باید به صورت List::Iterator استفاده شود.

کد کامل را در 03-IteratorLinkedList.cpp بخوانید.

9

template in C++

قالب‌ها در C++

قبلا دیدیم کلاس **Vector** میتواند به صورت های

vector <int>

vector <string>

یا ...

تعریف شود، یعنی نوعی که **vector** نگاه میدارد را تعیین میکنیم!

در **c++** این کار با قالب برای کلاس‌هایی که خودمان هم مینویسیم امکان پذیر است.

10

:Function Template

```cpp
#include <iostream>
#include <string>

using namespace std;

template <typename T>
T const& Max (T const& a, T const& b)
{
    return a < b ? b:a;
}
int main ()
{

    int i = 39;
    int j = 20;
    cout << "Max(i, j): " << Max(i, j) << endl;

    double f1 = 13.5;
    double f2 = 20.7;
    cout << "Max(f1, f2): " << Max(f1, f2) << endl;

    string s1 = "Hello";
    string s2 = "World";
    cout << "Max(s1, s2): " << Max(s1, s2) << endl;

   return 0;
}
```

```
Max(i, j): 39
Max(f1, f2): 20.7
Max(s1, s2): World
```

میتوان به جای کلمه کلیدی typename
از کلمه کلیدی class استفاده کرد.

04-FunctionTemplate.cpp

11

:Class template

```cpp
// class templates
#include <iostream>
using namespace std;

template <typename T>
class mypair {
    T a, b;
  public:
    mypair (T first, T second)
      {a=first; b=second;}
    T getmax ();
};

template <typename T>
T mypair<T>::getmax ()
{
  T retval;
  retval = a>b? a : b;
  return retval;
}

int main () {
  mypair <int> myobject (100, 75);
  cout << myobject.getmax();
  return 0;
}
```

```
100
```

به جای typename استفاده از class هم رایج است.

داخل این کلاس می‌توان از Tبه عنوان نام یک تایپ استفاده کرد.

فیلدهایی از تایپ T (a و b از نوع T هستند)

اگر بدنه‌ی متدی بیرون از کلاس تعریف می‌شود باید این عبارت قبل از تعریف آن ذکر شود.

استفاده از الگوی mypair با تایپ int

05-classTemplate.cpp

12

یک الگوی ساده:

در این مثال یک کلاس بسیار ساده که صرفاً یک مقدار را در خودش نگه‌می‌دارد پیاده‌سازی می‌شود. هر بار استفاده از این الگو باعث می‌شود یک کپی از آن ایجاد شود که تایپ مشخص‌شده (مثلاً int) به جای T جایگزین شده است.

```
template<typename T>
class Cell {
public:
    Cell(T d) : data(d) {}
    T get_value() const;
    void set_value(T x);
private:
    T data;
};

template<typename T>
T Cell<T>::get_value() const {
    return data;
}

template<typename T>
void Cell<T>::set_value(T x) {
    data = x;
}
```

```
int main() {
    Cell<int> a(10);
    a.set_value(12);
    cout << a.get_value() << endl;

    string s = "JJ";
    Cell<string> b(s);
    cout << b.get_value() << endl;
}
```

استفاده از الگوی Cell با تایپ int

استفاده از الگوی Cell با تایپ string

06-GenericCell.cpp

13

الگوی زوج مرتب

این الگو زوج مرتبی از دو تایپ متفاوت را ذخیره می‌کند. هدف این مثال نشان‌دادن حالتی است که یک الگو بیش از یک پارامتر تایپ دارد. تعریف کامل‌تری از این الگو در کتابخانه‌ی <utility> به نام pair موجود است.

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

template<typename U, typename V>
class Pair {
public:
    Pair(U f, V s) : first(f), second(s) {}
    U get_first() { return first; }
    V get_second() { return second; }
private:
    U first;
    V second;
};

int main()
{
    Pair<string, double> record("Ali", 18.5);
    cout << record.get_first();
}
```

در اینجا به جای U از تایپ string و به جای V از double استفاده می‌شود.

07-Pair.cpp

14

الگوهای توابع

این مثال مشابه اسلاید ۱۱ نشان می‌دهد چگونه توابعی تعریف کنیم که با تایپ‌های مختلف کار می‌کنند.
الگوی print_array برای آرایه‌هایی از تایپ‌های متفاوت قابل استفاده است.

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

template<typename T>
void print_array(T a[], int n) {
    for (int i = 0; i < n; i++)
        cout << a[i] << ' ';
}

int main()
{
    int a[3] = {1, 3, 4};
    char c[2] = {'b', 'a'};

    print_array<int>(a, 3);
    print_array<char>(c, 2);

    print_array(a, 3);
    print_array(c, 2);
}
```

یک الگوی تابع که بر اساس تایپ عناصر آرایه عمومی تعریف شده است.

استفاده از الگو برای تایپ int

لزومی به ذکر صریح تایپ قالب آن نیست، چرا؟!

08-FuncTemplate.cpp

پارامترهای غیرتایپ برای الگوها

می‌توان برای یک الگو، پارامترهایی غیر از تایپ نیز داشت. در این مثال، کلاسی برای پیاده‌سازی یک صف تعریف شده که عناصر را در آرایه‌ای که به طور ایستا تخصیص یافته نگهداری می‌کند. اندازه‌ی این آرایه در زمان کامپایل و به عنوان پارامتر الگو مشخص می‌شود. این روش اصلاً به عنوان روشی برای داشتن صف‌هایی با طول متفاوت مناسب نیست و صرفاً به عنوان مثالی برای پارامترهای غیرتایپ برای الگوها ذکر شده است. در صورت نیاز، توصیه می‌شود از روش تخصیص حافظه‌ی پویا که در مثال stack بیان شد استفاده کنید.

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

class queue_operation_exception {};

template<typename T, int S> class queue {
public:
    queue() { count = 0; }
    void enqueue(T x);
    T dequeue();
    int size() const { return elements.size(); }
private:
    T elements[S];
    int count;
};
```

این الگو پارامتری به نام S از نوع int دارد.

عناصر در آرایه‌ای ایستا به اندازه‌ی S ذخیره می‌شوند.

09-QueueTemplate.cpp

```cpp
template<class T, int S>
void queue<T, S>::enqueue(T x)
{
    if (count >= sizeof(elements))
        throw queue_operation_exception();
    elements[count++] = x;
}

template<class T, int S>
T queue<T, S>::dequeue()
{
    if (count == 0)
        throw queue_operation_exception();
    T result = elements[0];
    for (int i = 1; i < count; i++)
        elements[i-1] = elements[i];
    count--;
    return result;
}
```

09-QueueTemplate.cpp 17

```cpp
int main() {
    queue<int, 10> q;
    q.enqueue(10);
    q.enqueue(20);
    cout << q.dequeue() << endl;

    queue<string, 8> p;
    p.enqueue("salaam");
    p.enqueue("chetori?");
    cout << p.dequeue() << endl;
}
```

مقدار Kدر اینجا ۱۰ تعیین شده

آیا استفاده از این روش برای ایجاد آرایه‌های داینامیک که در زمان اجرا طولشان مشخص شود امکان پذیر است؟ چرا؟!

```cpp
int main() {
    int a;
    cin >>a;
    queue<int, a> q;
}
```

09-QueueTemplate.cpp 18

الگوی لیست پیوندی

در این مثال لیست پیوندی که قبلا نوشته شد (03-IteratorLinkedList.cpp) در قالب الگو پیاده‌سازی می‌شود.

```
template<typename T>
class List {
private:
    class Node {
    public:
        Node(T d, Node *n = NULL, Node *p = NULL)
            : data(d), next(n), prev(p) {}

        T data;
        Node *next;
        Node *prev;
    };

    ...
```

چون کلاس Node در داخل کلاس List تعریف شده، این کلاس نیز به پارامتر T دسترسی دارد.

10-LinkedListTemplate.cpp


```
    ...
int main() {
    List<int> l;

    l.push_back(86);
    l.push_front(43);
    l.push_front(12);

    l.print();
    cout << endl;

    int sum = 0;

    List<int>::Iterator it = l.get_iterator();
    while (it.has_more_elements())
        sum += it.next_element();

    cout << sum << endl;
}
```

به تایپ it دقت کنید.

کد کامل را در 10-LinkedListTemplate.cpp بخوانید.

منابع

- http://ramtung.ir/apnotes/html/14_LinkedLists.html
- http://ramtung.ir/apnotes/html/15_Templates.html
- http://www.cplusplus.com/forum/beginner/13703/
- http://www.tutorialspoint.com/cplusplus/cpp_templates.htm
- http://www.cplusplus.com/doc/tutorial/templates/

21