1. 摆放顺序应该为catch(int \*)，catch(const void \*)，catch(…)；不可以，catch(const void \*)会捕获任何指针异常，将catch(int \*)放在后面将导致其无法捕获int\*类型异常。
2. 左值
3. **不能**使用 \*const\_cast<int \*>(&x) = 3 修改x的值，**不能使用**\*(int \*)(&x) = 3修改x的值
4. static\_cast用来处理隐式转换,就是C语言里的强制转换，可以将int转为float，也可以将char\*转为int，将指向基类的指针转为一个指向子类的指针，同时可以将non-const转为const对象，但是它不能将一个const对象转为non-const；const\_cast用来将对象的常量属性转除，使常量可以被修改，而且type必须是指针，引用，或者指向对象类型成员的指针。
5. x指以值方式捕获变量x，y是调用时传入的参数，该表达式返回一个int型的值。F名义上是个函数，但实际上创建该表达式的同时，创建了一个匿名类，并创建了对象f。
6. ①template T<类型实参列表> 显式实例化

②在变量、函数参数、返回类型等定义时实例化类模板

1. ①g2 g3 a2 a3 a6一样 ②g4 a4一样

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 变量 | g1 | g2 | g3 | g4 | a1 | a2 | a3 | a4 | a5 | a6 |
| 第一条语句后 | 10 | 11 | 11 | 14 | 10 | 11 | 11 | 14 | 11 | 无 |
| 变量 | g1 | g2 | g3 | g4 | a1 | a2 | a3 | a4 | a5 | a6 |
| 第二条语句后 | 10 | 12 | 12 | 14 | 10 | 12 | 12 | 14 | 12 | 12 |

template <typename T>

class ARRAY {

T\* const data;

const int n;

public:

ARRAY(int n);

ARRAY(const T\* a, int n);

ARRAY(const ARRAY& a);

ARRAY(ARRAY&& a) noexcept;

~ARRAY();

operator int()const;

T& operator [](int index);

ARRAY& operator=(const ARRAY& a);

ARRAY& operator=(ARRAY&& a) noexcept;

bool operator==(const ARRAY& a)const;

ARRAY operator+(const ARRAY& a)const;

ARRAY& operator+=(const ARRAY& a);

void print()const noexcept;

};

template <typename T>

ARRAY<T>::ARRAY(int n) :n(n), data(new T[n])

{

if (!data)

throw "overflow";

}

template <typename T>

ARRAY<T>::ARRAY(const T\* a, int n) :n(n), data(new T[n])

{

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

data[i] = a[i];

}

template <typename T>

ARRAY<T>::ARRAY(const ARRAY& a) : n(a.n), data(new T[a.n])

{

if (!data)

throw "overflow";

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

data[i] = a.data[i];

}

template <typename T>

ARRAY<T>::ARRAY(ARRAY&& a) noexcept : n(a.n), data(a.data)

{

\*const\_cast<int\*>(&(a.n)) = 0;

\*const\_cast<T\*\*>(&(a.data)) = nullptr;

}

template <typename T>

ARRAY<T>::~ARRAY()

{

delete[] data;

}

template <typename T>

ARRAY<T>::operator int()const

{

return n;

}

template <typename T>

T& ARRAY<T>::operator [](int index)

{

if (index < 0 || index >= n)

throw "Array boundary exceeded";

return data[index];

}

template <typename T>

ARRAY<T>& ARRAY<T>::operator=(const ARRAY& a)

{

if (this != &a)

{

if (data != nullptr)

delete[] data;

\*const\_cast<int\*>(&(n)) = a.n;

data = new T[a.n];

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

data[i] = a.data[i];

}

return \*this;

}

template <typename T>

ARRAY<T>& ARRAY<T>::operator=(ARRAY&& a) noexcept

{

if (this != &a)

{

if (data != nullptr)

delete[] data;

\*const\_cast<int\*>(&(n)) = a.n;

\*const\_cast<int\*>(&(a.n)) = 0;

\*const\_cast<T\*\*>(&(data)) = a.data;

\*const\_cast<T\*\*>(&(a.data)) = nullptr;

}

return \*this;

}

template <typename T>

bool ARRAY<T>::operator==(const ARRAY& a)const

{

if (n != a.n) return false;

else

{

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

if (data[i] != a.data[i])

return false;

}

return true;

}

template <typename T>

ARRAY<T> ARRAY<T>::operator+(const ARRAY& a)const

{

ARRAY A(n + a.n);

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

A.data[i] = data[i];

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

A.data[i + n] = a.data[i];

if ((int)A != n + a.n)

throw "Add Failed";

return A;

}

template <typename T>

ARRAY<T>& ARRAY<T>::operator+=(const ARRAY& a)

{

\*this = \*this + a;

return \*this;

}

template <typename T>

void ARRAY<T>::print()const noexcept

{

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

cout << data[i] << ' ';

cout << endl;

}

int main()

{

//用int实例化

ARRAY<int> INT(10);

try { INT[10] = 2; }

catch (const char\* p)

{

cout << p << endl;

}

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

INT[i] = i + 1;

INT.print();

//用char实例化

const char\* a = "A Test String";

ARRAY<char> CHAR(a, strlen(a));

try { CHAR[-1] = 'c'; }

catch (const char\* p)

{

cout << p << endl;

}

CHAR.print();

try { CHAR[4] = 'x'; }

catch (const char\* p)

{

cout << p << endl;

}

CHAR.print();

//用double实例化

double double1[4] = { 1.0,2.0,3.0,4.0 };

double double2[4] = { 1.0,2.0,3.0,4.0 };

double double3[5] = { 1.0,2.0,3.0,4.0,5.0 };

ARRAY<double> DOUBLE1(double1, 4);

ARRAY<double> DOUBLE2(double2, 4);

ARRAY<double> DOUBLE3(double3, 5);

try {

cout << "ARRAY<double> DOUBLE4 = DOUBLE1 + DOUBLE3" << endl;

ARRAY<double> DOUBLE4 = DOUBLE1 + DOUBLE3;

DOUBLE4.print();

}

catch (const char\* p)

{

cout << p << endl;

}

DOUBLE1.print();

DOUBLE2.print();

DOUBLE3.print();

try {

cout << "DOUBLE1+=DOUBLE2" << endl;

DOUBLE1 += DOUBLE2;

}

catch (const char\* p)

{

cout << p << endl;

}

DOUBLE1.print();

DOUBLE2.print();

DOUBLE3.print();

return 0;

}

template <typename T>

class ARRAY

{

T\* const e;

const int r, c;

public:

ARRAY(int r, int c);

ARRAY(T\* a, int r, int c);

ARRAY(const ARRAY& a);

ARRAY(ARRAY&& a)

noexcept;

~ARRAY() noexcept;

T\* const operator[](int r);

ARRAY& operator=(const ARRAY& a);

ARRAY& operator=(ARRAY&& a) noexcept;

void print(void) const noexcept;

};

template <typename T>

ARRAY<T>::ARRAY(int r, int c) : r(r), c(c), e(new T[r \* c])

{

}

template <typename T>

ARRAY<T>::ARRAY(T\* a, int r, int c) : r(r), c(c), e(new T[r \* c])

{

for (int i = 0; i < r \* c; i++)

e[i] = a[i];

}

template <typename T>

ARRAY<T>::ARRAY(const ARRAY& a) : e(new T[a.r \* a.c]), r(a.r), c(a.c) {

for (int i = 0; i < a.r \* a.c; i++)

e[i] = a.e[i];

}

template <typename T>

ARRAY<T>::ARRAY(ARRAY&& a) noexcept : r(a.r), c(a.c), e(a.e) {

\*(const\_cast<int\*>(&a.r)) = 0;

\*(const\_cast<int\*>(&a.c)) = 0;

\*(const\_cast<T\*\*>(&a.e)) = nullptr;

}

template <typename T>

ARRAY<T>::~ARRAY() noexcept

{

if (e != nullptr)

delete[] e;

\*(const\_cast<int\*>(&r)) = 0;

\*(const\_cast<int\*>(&c)) = 0;

\*(const\_cast<T\*\*>(&e)) = nullptr;

}

template <typename T>

T\* const ARRAY<T>::operator[](int r) {

if (r < this->r && r >= 0)

return &e[r \* c];

else

throw "Operator[] ERROR!";

}

template <typename T>

ARRAY<T>& ARRAY<T>::operator=(const ARRAY& a) {

if (this != &a)

{

if (e != nullptr)

delete[] e;

\*(const\_cast<int\*>(&r)) = a.r;

\*(const\_cast<int\*>(&c)) = a.c;

\*(const\_cast<T\*\*>(&e)) = new T[r \* c];

for (int i = 0; i < a.r; i++)

for (int j = 0; j < a.c; j++)

e[i \* a.c + j] = a.e[i \* a.c + j];

}

return \*this;

}

template <typename T>

ARRAY<T>& ARRAY<T>::operator=(ARRAY&& a) noexcept {

if (this != &a)

{

if (e != nullptr)

delete[] e;

\*(const\_cast<int\*>(&r)) = a.r;

\*(const\_cast<int\*>(&c)) = a.c;

\*(const\_cast<T\*\*>(&e)) = a.e;

\*(const\_cast<int\*>(&a.r)) = 0;

\*(const\_cast<int\*>(&a.c)) = 0;

\*(const\_cast<T\*\*>(&a.e)) = nullptr;

}

return \*this;

}

template <typename T>

void ARRAY<T>::print(void) const noexcept

{

for (int i = 0; i < r \* c; i++)

{

cout << setw(3) << e[i];

if (i != 0 && (i + 1) % c == 0)

cout << endl;

}

}

int main(void)

{

int num = 1;

ARRAY<int> INT(4, 4);

for (int i = 0; i < 4; i++)

for (int j = 0; j < 4; j++)

\*(INT[i] + j) = num++;

INT.print();

char MAT[12];

for (int i = 0; i < 12; i++)

MAT[i] = 'a' + i;

ARRAY<char> CHAR(MAT, 3, 4);

CHAR.print();

try

{

\*(CHAR[-1]) = 0;

}

catch (const char\* p)

{

cout << p << endl;

}

try

{

\*(CHAR[12]) = 0;

}

catch (const char\* p)

{

cout << p << endl;

}

return 0;

}