1. **할당**

New/ Delete

변수형 할당 : int \*pNum; pNum = new int(1);

괄호 안은 초기화할 값.

C와 다르게, 구조체에 생성자가 만들어 질 수 있다. 3번째로 실행이 된다.

초기화 리스트: 구조체에서 const로 선언이된 경우 생성자에서 값지정이 되지 않기 때문에 초기화 리스트에서 설정할 수 있다.

//할당이 첫 번째.

SampleStruct() : **\_a(10), \_b(3.00f), \_c('b')** //초기화 리스트 2번째로 실행됨.

{

cout<<"생성자 실행됬음" <<endl;

\_b = 3.14f;

\_c = 'a';

\_d = 3.141592;

}

//C와 다르게, 생성자가 들어갈수 있다. 3번째로 실행이 된다.

~SampleStruct()

{

cout<<"소멸되었음"<<endl;

* 멀티쓰레드 처리할 때는 C++이 빠르다.
* C의 delete[] 는 C++에서 delete[원소],,, 메모리 반납 delete꼭 하자.
* 입력/출력은 printf/scanf를 쓰는게 더 빠르다.
* void Func(int A, int B = 0) 인자 초기화 가능하다 Default값은 무조건 오른쪽

끝에 있어야한다.

1. **오버로딩**

C에서는 오버로딩이 가능하지 않지만 C++에서는 오버로딩이 가능하다. 즉, 함수 이름은 같으나 인자가 다른경우 선언이 가능하다.

.

void Func(int A, int B = 0)

{

cout<<"int : "<<A<<endl;

cout<<B<<endl;

}

void Func(int A)

{

cout<<"Sample"<<endl;

}

//중복. 불가함. 첫번째 함수인경우는 인자초기화를 했을 때 두번째 인자는 꼭 안써도 되기 떄문에 두함수를 구별할 수가 없음. 불가.

새로운 연산자 정의

CPosition operator +(CPosition A, CPosition B)

{

CPosition C;

C.x = A.x + B.x;

C.y = A.y + B.y;

return C;

}

class CPosition

{

public:

int x;

int y;

};

* CPosition SampleA = {5, 5}; 구조체 초기화.

Tip : cout << boolalpha << bool형 변수

* bool형을 true, false로 표현해준다.

대입연산자

class CPosition

{

public:

int x;

int y;

void operator = (CPosition A)

{

this->x = A.x + 1;

this->y = A.y - 1;

}

};

연산자 오버로딩 할 때, 구조체 안에다가 하면 인자 하나, this->로 접근해야 하며

구조체 밖에 있을 경우에는 인자 두개로 계산함.

#include <iostream>

using namespace std;

class CPosition

{

public:

int x;

int y;

void operator = (CPosition A)

{

this->x = A.x + 1;

this->y = A.y - 1;

}

CPosition operator +(CPosition B)

{

CPosition C;

C.x = this->x + B.x;

C.y = this->y + B.y;

return C;

}

};

//Position끼리 더하면 x는 x끼리 더해지고

//y는 y끼리 더해진다.

/\*산술 연산자\*/

CPosition operator -(CPosition A, CPosition B)

{

CPosition C;

C.x = A.x - B.x;

C.y = A.y - B.y;

return C;

}

/\*비교 연산자\*/

bool operator == (CPosition A, CPosition B)

{

return (A.x == B.x) && (A.y == B.y);

}

CPosition operator \* (CPosition A, CPosition B)

{

CPosition C;

C.x = A.x\*B.x;

C.y = A.y\*B.y;

return C;

}

int main()

{

CPosition Orin = {5,5};

CPosition Dest;

Dest = Orin;

cout<<Dest.x<<" "<<Dest.y;

return 0;

}

1. **참조 (Reference)**

Call by Value / Call by Address / Call by Reference

C에서는 Call by Reference는 불가능하다.

참조와 포인터의 차이점은 포인터는 NUll값을 가지지만 참조자는 안된다. 초기화 시에 참조자는 객체를 직접 입력받고 포인터는 객체의 주소값을 받는다. **참조자는 한번 가리킨 대상 변경이 불가능하다.**

레퍼런스는 메모리 할당이 안되서 포인터보다 훨씬 가볍다.

#include <iostream>

using namespace std;

//Call by Reference

void Func\_Reference( int &DataA, int &DataB)

{

int temp = DataA;

DataA = DataB;

DataB = temp;

}

//Call by Addess

void Func\_Pointer( int \*pDataA, int \*pDataB)

{

int temp = (\*pDataA);

(\*pDataA) = (\*pDataB);

(\*pDataB) = temp;

}

int main()

{

int DataA = 100;

int DataB = 50;

Func\_Reference(DataA, DataB);

printf("%d %d", DataA, DataB);

Func\_Pointer(&DataA, &DataB);

printf("%d %d", DataA, DataB);

return 0;

}

1. **자료구조**
2. List는 스택이나 큐로 사용가능하다.

for(auto iter = list\_Sample.begin(); iter!=list\_Sample.end(); iter++)

* Auto는 무슨형인지 지정안해도 자동으로 잡아준다.

For(auto iter : list\_Sample)  
{  
}

* 처음부터 끝까지 순회함. 배열에서도 가능하나 배열의 크기를 미리 알고 있어야 가능함.

1. Vector

배열 + list

1. 클래스의 상속

여러 클래스로부터 상속받는 것을 다중상속이라고 하는데 C++에서는 다중상속을 허용한다.

하지만 최대한 안 쓰는게 좋음.

class CChild : public CParentA, CParentB

{

public:

};

* 특징 : [캡슐화 – 액세스 한정자] [추상화 – 오버 라이딩(가상함수)] [다형성]

Protected는 상속을 받는 클래스들은 사용할수 있지만 클래스 외부에서는 사용할 수 없다.

상속 방식에 따라서 상속되는것들이 달라짐. private상속했다면 모두 private로 상속됨.

가상 함수 : 자식함수에서 재정의 가능.

#include <iostream>

#include <list>

#include <vector>

using namespace std;

class CParent

{

public:

CParent(){this->\_A = 0; this->\_B = 0;};

virtual void Calc()

{

this->\_A++;

}

virtual void Printf(){cout<<this->\_A<<endl;};

private:

int \_A;

float \_B;

};

class CChild : public CParent

{

private:

int \_C;

protected:

public:

CChild()

{

this->\_C = 0;

}

void Calc()

{

CParent::Calc();

this->\_C++;

}

void Printf()

{

CParent::Calc();

cout<<this->\_C<<endl;

}

};

int main()

{

CChild SAMPLE;

for(int i=0; i<100; i++)

{

SAMPLE.Calc();

SAMPLE.Printf();

}

return 0;

}

* 재정의 할 때 부모의 계산을 먼저 수행하고, 밑에 있는 것을 나읙 것으로 써야 한다. 왜냐하면 부모의 private에 접근 할 수 없기 때문에.

순수가상함수 : virtual void DecreaseHealth() = 0;

포인터로는 쓸수 있지만 변수로는 쓸 수 없다. 클래스를 포인터로 선언해야한다.

#include <iostream>

#include <list>

#include <vector>

using namespace std;

//추상클래스¨¬

class CUnit

{

private:

public:

int health;

int pos;

virtual void DecreaseHealth() = 0; //순수가상함수

};

class CKtlin : public CUnit

{

void DecreaseHealth()

{

this->health -= 10;

}

};

class CTimo : public CUnit

{

bool isHided;

void DecreaseHealth() { this->health -= 5; }

};

int main()

{

//다¥U양úc성ù¨¬

CUnit \*OBJECT = new CKtlin;

CUnit \*OBJECT2 = new CTimo;

return 0;

}

Dynamic Cast

CNami \*temp = dynamic\_cast<CNami\*>(OBJECTS[i]);

23-3 김동한

#include <iostream>

#include <list>

#include <vector>

#define MAXCOUNT 10

using namespace std;

class CUnit

{

public:

CUnit()

{

this->health = 100;

this->pos = rand() % 100;

}

int health;

int pos;

virtual void Move() = 0;

virtual void Introduce() = 0;

};

/\*Ktlin moves + 1\*/

class CKtlin : public CUnit

{

void Move(){this->pos+=1;}

void Introduce(){}

};

/\*Timo Moves -1\*/

class CTimo : public CUnit

{

void Move(){this->pos-=1;}

void Introduce(){}

};

/\*Nami moves \*/

class CNami : public CUnit

{

void Move(){this->pos+=2;}

void Introduce(){}

};

int main()

{

CUnit \*OBJECTS[MAXCOUNT];

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

{

switch(rand()%3)

{

case 0: OBJECTS[i] = new CKtlin; break;

case 1: OBJECTS[i] = new CTimo; break;

case 2: OBJECTS[i] = new CNami; break;

default: cout<< "Exception Error" << endl; break;

}

}

while(1)

{

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

{

CNami \*temp = dynamic\_cast<CNami\*>(OBJECTS[i]);

//나ø¨£미öI면¬e 값Æ¨£이I 들ìe어úi가Æ¢®고Æi 아ú¨¡니¥I면¬e NULL값Æ¨£이I 들ìe어úi간Æ¡Ì다¥U.

cout<<OBJECTS[i]->health<<endl;

if(temp != NULL) temp->pos = 0;

OBJECTS[i]->Move();

cout<<"위¡×치¢®"<<OBJECTS[i]->pos<<endl;

}

getchar();

}

return 0;

}