RTU
Lietišķo datorzinātņu katedra

Uldis Sukovskis

Objektorientētā programmēšana

C++

P I E M Ē R I

Saturs

C++ operāciju prioritātes

The #1 category has the highest precedence, category #2 (Unary operators) takes second precedence, and so on to the Comma operator,which has lowest precedence. The operators within each category have equal precedence.

#	*Category*	*Operator*	*What it is (or does)*	*Associativity*
1.	Highest	`()`	Function call	left-to-right
		`[]`	Array subscript	
		`->`	C++ indirect component selector	
		`::`	C++ scope access/resolution	
		`.`	C++ direct component selector	
2.	Unary	`!`	Logical negation (NOT)	right-to-left
		`~`	Bitwise (1's) complement	
		`+`	Unary plus	
		`-`	Unary minus	
		`++`	Preincrement or postincrement	
		`--`	Predecrement or postdecrement	
		`&`	Address	
		`*`	Indirection	
		`sizeof`	(returns size of operand, in bytes)	
		`new`	(dynamically allocates C++ storage)	
		`delete`	(dynamically deallocates C++ storage)	
3.	Member access	`.*`	C++ dereference	left-to-right
		`->*`	C++ dereference	
4.	Multiplicative	`*`	Multiply	left-to-right
		`/`	Divide	
		`%`	Remainder (modulus)	
5.	Additive	`+`	Binary plus	left-to-right
		`-`	Binary minus	
6	Shift	`<<`	Shift left	left-to-right
		`>>`	Shift right	
7.	Relational	`<`	Less than	left-to-right
		`<=`	Less than or equal to	
		`>`	Greater than	
		`>=`	Greater than or equal to	
8.	Equality	`==`	Equal to	left-to-right
		`!=`	Not equal to	
9.		`&`	Bitwise AND	left-to-right
10.		`^`	Bitwise XOR	left-to-right
11.		`|`	Bitwise OR	left-to-right
12.		`&&`	Logical AND	left-to-right
13.		`||`	Logical OR	left-to-right
14.	Conditional	`?:`	(a ? x : y means "if a then x, else y")	right-to-left
15.	Assignment	`=`	Simple assignment	right-to-left
		`*=`	Assign product	
		`/=`	Assign quotient	
		`%=`	Assign remainder (modulus)	
		`+=`	Assign sum	
		`-=`	Assign difference	
		`&=`	Assign bitwise AND	
		`^=`	Assign bitwise XOR	
		`|=`	Assign bitwise OR	
		`<<=`	Assign left shift	
		`>>=`	Assign right shift	
16.	Comma	`,`	Evaluate	left-to-right

All of the operators in this table can be overloaded except the following:

.	C++ direct component selector
.*	C++ dereference
::	C++ scope access/resolution
?:	Conditional

Vienkāršas programmas piemērs

Triangle.h

```
class Triangle {
      public:
            int a, b, c;
            float area();
            int perimeter();
};
```

Triangle.cpp

```
#include <math.h>
#include "triang.h"

float Triangle::area()
{
      float p;
      p = (a + b + c) / 2.0;  // pusperimetrs
      return ( sqrt(p * (p - a) * (p - b) * (p - c)) );
}

int Triangle::perimeter()
{
      return (a + b + c);
}
```

TestTtriangle.cpp

```
#include <iostream.h>
#include "triang.h"

void main()
{
      Triangle t;
      // Malu garumi
      t.a = 50;
      t.b = 60;
      t.c = 35;

      float ta;
      ta = t.area();
      cout  << "Area of triangle is " << ta << '\n';
      cout  << "Perimeter of triangle is " << t.perimeter() << '\n';
}
```

Klases. Ģeometriski objekti

```
class Square {
public:
        int side;
        int area();
        int perimeter();
};

int Square::area()
{
        return side * side;
}

int Square::perimeter()
{
        return 4*side;
}
//-----------------------------------------------------------------------------------------------

class Circle {
public:
        int radius;
        float area();
        float perimeter();
};

float Circle::area()
{
         return 3.14159 * radius * radius;
}

float Circle::perimeter()
{
        return (2 * 3.14159 * radius);
}
//-----------------------------------------------------------------------------------------------

class Triangle {
public:
        int a, b, c;
        float area();
        int perimeter();
};

#include <math.h>
float Triangle::area()
{
        float p;
        p = (a + b + c) / 2.0;
        return ( sqrt(p * (p - a) * (p - b) * (p - c)) );
}

int Triangle::perimeter()
{
        return (a + b + c);
}
//-----------------------------------------------------------------------------------------------

#include <iostream.h>
void main()
{
        Circle c;
        c.radius = 10;
        cout << "Area of circle is " << c.area() << '\n';
        cout << "Perimeter of circle is " << c.perimeter() << '\n';
        Square s;
        s.side = 10;
        cout << "Area of square is " << s.area() << '\n';
        cout << "Perimeter of square is " << s.perimeter() << '\n';
        Triangle t;
        t.a = 10;       t.b = 10;       t.c = 10;
        cout << "Area of triangle is " << t.area() << '\n';
        cout << "Perimeter of triangle is " << t.perimeter() << '\n';
}
```

Klases locekļu pieejamība (*private* un *public*)

```
class Square {
private:
      int side;
public:
      void setSide(int x);
      int getSide();
      int area();
      int perimeter();
};



int Square::area()
{
      return side * side;
}

int Square::perimeter()
{
      return 4*side;
}

void Square::setSide(int x)
{
      side = x;
}
int Square::getSide()
{
      return side;
}



#include <iostream.h>
void main()
{
      Square s;
      s.setSide(20);
      cout << "Side = " << s.getSide() << '\n';
      cout << "Area = " << s.area() << '\n';
      cout << "Perimeter = " << s.perimeter() << '\n';
}
```

Klases locekļu pieejamība (*turpinājums*)

```
class Triangle {
	int a, b, c;
public:
	int setSides(int x, int y, int z);
	void getSides(int *x, int *y, int *z);
	float area();
	int perimeter();
};


#include <math.h>
float Triangle::area()
{
	float p, s;
	p = perimeter() / 2.0;
	s = (p * (p - a) * (p - b) * (p - c));
	if ( s >= 0 ) return ( sqrt(p * (p - a) * (p - b) * (p - c)) );
	else          return -1;
}

int Triangle::perimeter()
{
	return (a + b + c);
}

int Triangle::setSides(int x, int y, int z)
{
	a = x;
	b = y;
	c = z;
	if ( a + b < c || b + c < a || a + c < b ) return 0;
	return 1;
}

void Triangle:: getSides(int *x, int *y, int *z)

{
	*x = a;
	*y = b;
	*z = c;
}



#include <iostream.h>

void main()
{
	Triangle t1;

	if ( t1.setSides(10, 10, 10) ){
		cout << "Area of the triangle 1 is " << t1.area() << '\n';
	}
	else cout << "Invalid sides! \n\a";


	Triangle t2;
	if ( t2.setSides(4, 5, 10) ){
		cout << "Area of the triangle 2 is " << t2.area() << '\n';
	}
	else cout << "Invalid sides! \n\a";

	int s1, s2, s3;
	t1.getSides(&s1, &s2, &s3);
	cout << "Sides of the triangle 1 are : " << s1 << ", " << s2 << ", " << s3 << '\n';

	t2.getSides(&s1, &s2, &s3);
	cout << "Sides of the triangle 2 are : " << s1 << ", " << s2 << ", " << s3 << '\n';
	}
```

Iebūvētās (*inline*) klases funkcijas

```
class Square {
private:
      int side;
public:
      void setSide(int x) { side = x; }   // inline function
      int getSide() { return side; }      // inline function
      int area();
      int perimeter();                    // inline function
};



int Square::area()
{
      return side * side;
}

inline int Square::perimeter()
{
      return 4*side;
}



#include <iostream.h>
void main()
{
      Square s;
      s.setSide(20);
      cout << "Side = " << s.getSide() << '\n';
      cout << "Area = " << s.area() << '\n';
      cout << "Perimeter = " << s.perimeter() << '\n';
}
```

Ievades plūsma no faila

```
// Each line of the input file test.dat contains three elements
// separated with whitespaces: registration number, weight and name.
// For instance:
// 9400132  60.6  Peterson
// 9500276  91.5  Morgan

#include <fstream.h>
#include <stdlib.h>

void main()
{
	const int MaxNameLength = 21;
	const char dataFile[] = "test.dat";

	struct person{
		unsigned long regNum;
		char name[MaxNameLength];
		float weight;
	};

	person s[10];		// Array of structures
	ifstream infile;		// Input file
	infile.open(dataFile);	// Open file

	if (infile.bad()) { // If file opening fails, then exit
		cout << "Error opening file " << dataFile << "!\a\n";
		exit(1);
	}

	// Read up to 10 lines from data file
	int n = 0;	// Actual number of lines
	while ( n < 10 ) {
		infile >> s[n].regNum;
		if (!infile.good()) break; // Break loop if input fails
		infile >> s[n].weight;
		infile.width(MaxNameLength);	// To avoid overflow of name
		infile >> s[n].name;
		n++;
	}
	infile.close();

	// Output of data stored in the elements of array
	int i;
	for( i = 0 ; i < n ; ++i ){
		cout << i << "\t" <<
			s[i].regNum << "\t" <<
			s[i].weight << "\t" <<
			s[i].name << "\n";
	}

	// Find min. weight
	float minWeight = s[0].weight;
	for( i = 1 ; i < n ; ++i ){
		if (s[i].weight < minWeight) minWeight = s[i].weight;
	}
	cout << "\nMin. weight is " << minWeight;

	// Calculate total and average weight
	float totalWeight = 0;
	for( i = 0 ; i < n ; ++i ){
		totalWeight += s[i].weight;
	}
	cout << "\nTotal weight is " << totalWeight;
	cout << "\nAverage weight is " << totalWeight/n;
}
```

Ievades plūsma no teksta faila un izvade binārā failā

```
// Each line of the input file test.dat contains three elements
// separated with whitespaces: registration number, weight and name.
// For instance:
// 9400132  60.6  Peterson
// 9500276  91.5  Morgan


#include <fstream.h>
#include <stdlib.h>

void main()
{
        const int MaxNameLength = 20;
        const char dataFile[] = "test.dat";
        const char diskFile[] = "disk.dat";

        struct person{
                unsigned long regNum;
                char name[MaxNameLength+1];
                float weight;
        };

        person s[10];          // Array of structures

        ifstream infile;
        infile.open(dataFile);

        if (infile.bad()) {
                cout << "Error opening file " << dataFile << "!\a\n";
                exit(1);
        }

        // Read up to 10 lines from data file
        int n = 0;     // Actual number of lines
        while ( n < 10 ) {
                infile >> s[n].regNum;
                if (!infile.good()) break;
                infile >> s[n].weight;
                infile.width(MaxNameLength+1);        //to avoid overflow of name
                infile >> s[n].name;
                n++;
        }
        infile.close();

        ofstream outfile;
        outfile.open(diskFile);

        // Output of data stored in the array to the binary disk file
        int i;
        for( i = 0 ; i < n ; ++i ){
                outfile.write( (char*)&s[i].regNum, sizeof(person) );
        }

        outfile.close();

        // Find person according to registration number entered
        infile.open(diskFile);
        long queryNum;
        person p;
        while (1){
                cout << "\n\nEnter registration number (or 0 to quit):";
                cin >> queryNum;
                if (queryNum == 0) break;
                infile.seekg(0);
                for( i = 0 ; i < n ; ++i ){
                        infile.read( (char*)&p.regNum, sizeof(person) );
                        if (p.regNum == queryNum){
                                cout << '\n' << p.name;
                                break;
                        }
                }
                if (i ==n ) cout << "\n\aNot found!";
        }
        cout << "\n\aBye...";
}
```

Vienkāršas simbolu virknes klases realizācija

```
//An elementary implementation of the string class

class MyString{
private:
      char s[256];
      int len;
public:
      void assign(char *str);
      int length() {return len;};
      void print();
};

// Assign new string value
void MyString::assign(char *str)
{
       strcpy(s, str);        //copy string
       len = strlen(str);     //assign length
}

// Output of string value with newline
void MyString::print()
{
      cout << s << "\n";
}



// Test MyString class

void main()
{
      MyString one, two;
      char myName[40] = "My name is Charles";

      one.assign("My name is John");
      two.assign(myName);

      one.print();
      two.print();
      two.printLength();
}
```

Vienkāršas simbolu virknes klases realizācija ar konstruktoriem un destruktoru

```
class MyString{
private:
        char *ps;
        int size;
        int len;
public:
        MyString();
        MyString(int maxLength);
        MyString(char *str);
        ~MyString();
        void assign(char *str);
        int length() {return len;};
        void print();
};

// Constructors
MyString::MyString()
{
        ps = new char[256];
        size = 256;
        len = 0;
}

MyString::MyString( int maxLength )
{       if (maxLength < 1) {
                cout << "Illegal string size : " << maxLength;
                exit(0);
        }
        ps = new char[maxLength + 1];
        size = maxLength + 1;
        len = 0;
}

MyString::MyString( char *str )
{
        len = strlen( str );
        ps = new char[len+1];
        strcpy( ps, str );
        size = len+1;
}

// Destructor
MyString::~MyString()
{
        delete ps;
}

// Assign new string value
void MyString::assign(char *str)
{
        strncpy(ps, str, size);       //copy string
        ps[size] = '\0';              //assure terminal 0
        len = strlen(str);            //assign length
}

// Output of string value with newline
void MyString::print()
{
        cout << ps << "\n";
}

// Test MyString class
void main()
{       char myName[40] = "My name is Benjamin";
        MyString a, b(10), c("My name is Charles");
        a.assign("My name is Albert");
        b.assign(myName);
        a.print();
        b.print();
        c.print();
        MyString *q;
        q = new MyString(25);
        q->assign("My name is David");
        q->print();
}
```

Vienkāršas steka klases realizācija

stack.h

```
#ifndef boolean_enum
enum boolean{ FALSE = 0, TRUE = 1 };
#define boolean_enum
#endif

const int MAX_STACK_LENGTH = 256;

class Stack{
	private:
		char s[MAX_STACK_LENGTH];
		int top;
		enum {EMPTY = -1, FULL = MAX_STACK_LENGTH-1};
	public:
		void push( char c ) { top++; s[top] = c;}
		char pop() { return( s[top--] ); }
		char pop( int n );
		void reset() { top = EMPTY; }
		char topOf() { return( s[top] ); }
		boolean isFull() { return(boolean)(top == FULL); }
		boolean isEmpty() { return(boolean)(top == EMPTY); }
		void print(int n = 10); //for debugging only
		void clean(); //for debugging only
};
```

stack.cpp

```
#include "stack.h"
#include <iostream.h>

char Stack::pop( int n )
{
	while( n-- > 1 )
		top--;
	return( s[top--] );
}
void Stack::clean()
{
	for(int i=0; i<MAX_STACK_LENGTH; i++)
		s[i] = ' ';
	top = EMPTY;
}

void Stack::print(int n)
{
	cout << '\n';
	for(int i=0; i < n && i < MAX_STACK_LENGTH; i++)
		cout << s[i];
}
```

mystack.cpp

```
#include <iostream.h>
#include "stack.h"
#include "ustack.h"

void main()
{
	cout << "01234567890123456789012345678 9";
	Stack s1;
	s1.reset();
	s1.clean();
	s1.push('x');
	s1.print();
	s1.push('y');
	s1.push('z');
	s1.print();
	cout << '\n' << s1.pop();
	s1.print();
	cout << '\n' << s1.topOf();
	cout << '\n' << s1.pop(2);
	s1.print();
}
```

Vienkāršas steka klases realizācija ar konstruktoriem un destruktoru

ustack.h

```
#ifndef boolean_enum
enum boolean{ FALSE = 0, TRUE = 1 };
#define boolean_enum
#endif
class UStack{
	private:
		char *s;
		int top;
		int max_len;
		enum {EMPTY = -1};
	public:
		UStack();
		UStack( int n );
		~UStack();
		boolean push( char c );
		boolean push( char *cp, int n);
		char pop() { return( s[top--] ); }
		char pop( int n );
		void reset() { top = EMPTY; }
		char topOf() { return( s[top] ); }
		boolean isFull() { return(boolean)(top == max_len-1); }
		boolean isEmpty() { return(boolean)(top == EMPTY); }
		void print(int n = 10); //for debugging only
		void clean(); //for debugging only
};
```

ustack.cpp

```
#include <iostream.h>
#include "ustack.h"

UStack::UStack()
{
	max_len = 256;
	s = new char[max_len];
	top = EMPTY;
}

UStack::UStack( int size )
{
	max_len = size;
	s = new char[size];
	top = EMPTY;
}

UStack::~UStack()
{
	delete s;
}


boolean UStack::push( char c)
{
	if ( !isFull() ){
		top++;
		s[top] = c;
	 return TRUE;
	}
	else return FALSE;
}


boolean UStack::push( char *cp, int n )
{
	if ( (top + n) > max_len ) return FALSE;
	for ( int i = 0 ; i < n ; i++)
		push( *(cp+i) );
	return TRUE;
}
```

```
char UStack::pop( int n )
{
        while( n-- > 1 && top-- > 1 );
        return( s[top--] );
}

void UStack::clean()
{
        for(int i = 0; i<max_len; i++)
                s[i] = ' ';
        top = EMPTY;
}

void UStack::print(int n)
{
        cout << '\n';
        for(int i = 0; i < n && i < max_len; i++)
                cout<<s[i];
}
```

mystack.cpp

```
#include <iostream.h>
#include "stack.h"
#include "ustack.h"

void main()
{
        cout << "012345678901234567890123456789";

        UStack s2(10);

        s2.clean();
        s2.push('a');
        s2.print();

        char str[] = "ABCDEFGHIJKLMN";
        boolean ret;

        s2.push( str, 5);
        s2.print();

        cout << '\n' << s2.pop();
        cout << '\n' << s2.pop(2);

        ret = s2.push( str, 10);

        cout << "  ret = " << ret;
        cout << '\n' << s2.pop();
        cout << '\n' << s2.pop(10);
}
```

Operāciju pārdefinēšana (*Vect* klases realizācija)

```
#include <iostream.h>
#include <stdlib.h>

class Vect{
private:
        int *p;
        int size;
public:
        Vect();                                 // create a size 16 vector
        Vect(int n);                            // create a size n vector
        Vect(const Vect& v);                    // create and initialize by vector
        Vect(const int a[], int n);             // create and initialize by array
        ~Vect() { delete p; }                   // inline destructor

        int& operator [](int i);                // overloaded [] : range checked element of vector
        Vect& operator =(const Vect& v);        // overloaded =  : assignment of vector

        int sizeOf() { return size; }           // size of vector
        void print(int n);                      // print first n elements
        void print();                           // print all elements
};

Vect::Vect()
{
        size = 16;
        p = new int[size];
}

Vect::Vect(int n)
{
        if ( n <= 0 ){
                cout <<  "Illegal Vect size: " << n << '\n';
                exit(1);
        }
        size = n;
        p = new int[size];
}

Vect::Vect(const Vect& v)
{
        size = v.size;
        p = new int[size];
        for( int i = 0; i < size; ++i) p[i] = v.p[i];
}

Vect::Vect(const int a[], int n)
{
        if ( n <= 0 ){
                cout <<  "Illegal Vect size: " << n << '\n';
                exit(1);
        }
        size = n;
        p = new int[size];
        for( int i = 0; i < size; ++i) p[i] = a[i];
}

int& Vect::operator [](int i)
{
        if ( i < 0 || i > size-1 ){
                cout <<  "Illegal Vect index: " << i << '\n';
                exit(2);
        }
        return ( p[i] );
}

Vect& Vect::operator =(const Vect& v)
{
        int s = (size < v.size) ? size : v.size;
        for( int i = 0; i < s; ++i) p[i] = v.p[i];
        return ( *this );
}
```

```
void Vect::print(int n)
{
        int s = ( n < size ) ? n : size;
        for( int i = 0; i < s; ++i) cout << p[i] << ' ';
        cout << '\n';
}

void Vect::print()
{
        print(size);
}



void main(void)
{
        Vect v1;
        Vect v2(7);

        int i;
        for (i = 0; i < v1.sizeOf(); ++i) v1[i] = i + 100;
        v1.print( v1.sizeOf() );

        for (i = 0; i < v2.sizeOf(); ++i) v2[i] = i + 200;
        v2.print();



        int a[] = { 11, 12, 13, 14, 15, 16, 17, 18, 19 };
        Vect v3( a, 5);
        v3.print();

        v3 = v1;        // use overloaded assignment

        v3.print();

        for (i = 0; i < 10 ; ++i) v3[i] = i;          // use overloaded []

        v3.print();

}
```

```
100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115
200 201 202 203 204 205 206
11 12 13 14 15
100 101 102 103 104
Illegal Vect index: 5
```

Operāciju pārdefinēšana (*Clock* klases realizācija)

```
#include <iostream.h>

class Clock{
        unsigned long tot_secs;
        unsigned long secs;
        unsigned long mins;
        unsigned long hours;
        unsigned long days;
public:
        Clock(unsigned long);
        void tick();
        Clock operator ++();
        Clock operator ++(int);              // postfix operator
        Clock operator +(Clock);
//      Clock operator *(unsigned int); // this multiply operator can be used only in form: clock*i
        friend Clock operator *(unsigned int, Clock c);      // use friend functions instead
        friend Clock operator *(Clock c, unsigned int);      // of class member function

        void print();
};

// Constructor with initialization
inline Clock::Clock(unsigned long i)
{
        tot_secs = i;
        secs = tot_secs % 60;
        mins = (tot_secs / 60) % 60;
        hours = (tot_secs / 3600) % 24;
        days = tot_secs / 86400;
}

// Add one second
void Clock::tick()
{
        Clock temp = Clock( ++tot_secs );
        secs = temp.secs;
        mins = temp.mins;
        hours = temp.hours;
        days = temp.days;
}

// Overload prefix operator ++clock
Clock Clock::operator ++()
{
        this->tick();
        return (*this);
}

// Overload postfix operator clock++
// Parameter int is used by compiler as postfix indicator only
Clock Clock::operator ++(int)
{
        Clock temp = *this;
        this->tick();
        return (temp);
}

// Overload operator clock + clock
Clock Clock::operator +(Clock c)
{
        unsigned long total = tot_secs + c.tot_secs;
        Clock temp(total);
        return (temp);
}

/*********************************************************************
// member function overloads operator * has Clock& as the first argument
// and therefore can be used only in form  clock * integer
Clock Clock::operator *(unsigned long m)
{
        unsigned long t = m * tot_secs;
        Clock temp(t);
        return (temp);
}
*********************************************************************/

void Clock::print()
{
        cout    << days << " d :" << hours << " h :"
                        << mins << " m :" << secs  << " s\n";
}
```

```
// Friend of class Clock overloads  clock * integer
Clock operator *(Clock c, unsigned int m)
{
        unsigned long t = m * c.tot_secs;
        Clock temp(t);
        return (temp);
}
// Friend of class Clock overloads  integer * clock
Clock operator *(unsigned int m, Clock c)
{
        return (c * m);
}

//=============================================================================================

void main(void)
{
        Clock t1(59);
        Clock t2(172799);
        t1.print();
        t2.print();
        ++t1;
        t2++;                   // warning if no postfix operator overloaded
        t1.print();
        t2.print();
        (t1 + t2).print();      // can be used, but is not good style

        Clock t3 = t1;          // default copy constructor Clock::Clock(Clock&)
        t3.print();
        t3 = t2 + 30;           // user-defined type conversion via
                                // single argument constructor Clock(unsigned long)
        t3.print();

//      Clock t4;       // error, constructor Clock::Clock() not defined
//      t4 = t3 - 30;   // error, operation not defined

        t3 = t2 * 2;
        t3.print();
        t3 = 2 * t2;    // error "Illegal structure operation", when using
                        // member function for the operator overloading. Member function
                        // must have the first argument of its class.
        t3.print();

        // test prefix and postfix operators ++
        t1 = 600;
        t3 = ++t1;
        t1.print();
        t3.print();
        t1 = 600;
        t3 = t1++;
        t1.print();
        t3.print();
}
```

```
0 d :0 h :0 m :59 s
1 d :23 h :59 m :59 s
0 d :0 h :1 m :0 s
2 d :0 h :0 m :0 s
2 d :0 h :1 m :0 s
0 d :0 h :1 m :0 s
2 d :0 h :0 m :30 s
4 d :0 h :0 m :0 s
4 d :0 h :0 m :0 s
0 d :0 h :10 m :1 s
0 d :0 h :10 m :1 s
0 d :0 h :10 m :1 s
0 d :0 h :10 m :0 s
```

Bāzes un atvasinātās klases

```
#include <iostream.h>
//=================================================================================================
class RoadVehicle{
private:
        int wheels;
        int passengers;
public:
        void setWheels(int num) { wheels = num; }
        int getWheels() { return wheels; }
        void setPass(int num) { passengers = num; }
        int getPass() { return passengers; }
};
//=================================================================================================
class Truck : public RoadVehicle{
private:
        float load;
public:
        void setLoad(float weight) { load = weight; }
        float getLoad() { return load; }
        void print();
};

void Truck::print()
{
        cout << "Truck :" << '\n';
        cout << "wheels " << getWheels() << '\n';
        cout << "passengers " << getPass() << '\n';
        cout << "load " << getLoad() << " t" << "\n\n";
}
//=================================================================================================
class Car : public RoadVehicle{
private:
        int doors;
public:
        void setDoors(int n) { doors = n; }
        int getDoors() { return doors; }
        void print();
};

void Car::print()
{
        cout << "Car :" << '\n';
        cout << "wheels " << getWheels() << '\n';
        cout << "passengers " << getPass() << '\n';
        cout << "doors " << getDoors() << "\n\n";
}
/*-------------------------------------------------------------------------------------------
// Incorrect version of the member function Car::print()
void Car::print()
{       cout << "Car :" << '\n';
        cout << "wheels " << wheels << '\n';         // wheels is not accessible
        cout << "passengers " << passengers << '\n'; // passengers is not accessible
        cout << "doors " << doors << "\n\n";
}
-------------------------------------------------------------------------------------------*/

void main()
{       Truck t1, t2;
        Car c;

        t1.setWheels(4); t1.setPass(2); t1.setLoad(3.5);
        t2.setWheels(10); t2.setPass(3); t2.setLoad(12);
        t1.print();
        t2.print();

        c.setWheels(4); c.setPass(5); c.setDoors(5);
        c.print();

        // Use base class pointer to access derived class objects
        RoadVehicle *p;
        p = &t1;
        p->setPass(1);
//      p->setLoad(2.5);       // error : setLoad() is not a member of the base class
//      p->print();            // error : print() is not a member of the base class

        p = &c;
        p->setPass(4);
//      p->print();            // error : print() is not a member of the base class
        t1.print();
        c.print();
}
```

```
Truck :
wheels 4
passengers 2
load 3.5 t

Truck :
wheels 10
passengers 3
load 12 t

Car :
wheels 4
passengers 5
doors 5

Truck :
wheels 4
passengers 1
load 3.5 t

Car :
wheels 4
passengers 4
doors 5
```

Virtuālās funkcijas (Transporta objekti)

```
class RoadVehicle{
private:
        int wheels;
        int passengers;
public:
        void setWheels(int num) { wheels = num; }
        int getWheels() { return wheels; }
        void setPass(int num) { passengers = num; }
        int getPass() { return passengers; }
        virtual void print();
};

class Truck : public RoadVehicle{
private:
        float load;
public:
        void setLoad(float weight) { load = weight; }
        float getLoad() { return load; }
        void print();
};

class Car : public RoadVehicle{
private:
        int doors;
public:
        void setDoors(int n) { doors = n; }
        int getDoors() { return doors; }
        void print();
};
//*****************************************************************************
#include <iostream.h>
void RoadVehicle::print()
{
        cout << "wheels " << wheels << '\n';
        cout << "passengers " << passengers << '\n';
}

void Truck::print()
{
        cout << "Truck :" << '\n';
        cout << "wheels " << getWheels() << '\n';
        cout << "passengers " << getPass() << '\n';
        cout << "load " << getLoad() << " t" << "\n\n";
}

void Car::print()
{
        cout << "Car :" << '\n';
        RoadVehicle::print();                // use base class member function
        cout << "doors " << getDoors() << "\n\n";
}
//*****************************************************************************
void main()
{
        Truck t1, t2;  Car c;

        t1.setWheels(4); t1.setPass(2); t1.setLoad(3.5);
        t2.setWheels(10); t2.setPass(3); t2.setLoad(12);
        t1.print();
        t2.print();

        c.setWheels(4); c.setPass(5); c.setDoors(5);
        c.print();
        // Use base class pointer to access derived class objects
        RoadVehicle *p;
        Truck *tp;
        p = &t1;
        p->setPass(1);
        tp = (Truck*)p;        // use pointer of derived class
        tp->setLoad(2.5);
        p->print();            // print() is virtual member function of base class
        p = &c;
        p->setPass(4);
        ((Car*)p)->setDoors(4);// use typecast to access derived class member
        p->print();            // print() is virtual member function of base class
}
```

```
Truck :
wheels 4
passengers 2
load 3.5 t

Truck :
wheels 10
passengers 3
load 12 t

Car :
wheels 4
passengers 5
doors 5

Truck :
wheels 4
passengers 1
load 2.5 t

Car :
wheels 4
passengers 4
doors 4
```

Virtuālās funkcijas (Ģeometriskie objekti)

figure.h

```
#ifndef figure_h
#define figure_h
class Figure{
protected:
        double x;
        double y;
public:
        Figure();
        Figure(double, double);
        void setCoord(double x , double y) { this->x = x; this->y = y; }
        void getCoord(double *x , double *y) { *x = this->x; *y = this->y; }
        virtual double area() = 0;
};
#endif
```

figure.cpp

```
#include "figure.h"
Figure::Figure()
{
        x = y = 0.0;
}

Figure::Figure(double x, double y)
{
        setCoord( x, y);
}
```

rectang.h

```
#include "figure.h"
#ifndef rectang_h
#define rectang_h
class Rectangle : public Figure {
private:
        double w;
        double h;
public:
        Rectangle();
        Rectangle(double w, double h);
        Rectangle(double x, double y, double w, double h);
        void setDim(double x, double y, double w, double h);
        void getSize( double *w, double *h);
        void resize(double w, double h);
        virtual double area();
};
#endif
```

rectang.cpp

```
#include "rectang.h"
Rectangle::Rectangle()
{       w = h = 0.0;
}

Rectangle::Rectangle(double w, double h) : Figure(0.0, 0.0)
{       this->w = w;
        this->h = h;
}

Rectangle::Rectangle(double x, double y, double w, double h) : Figure(x, y)
{       this->w = w;
        this->h = h;
}

void Rectangle::setDim(double x, double y, double w, double h)
{       this->x = x;
        this->y = y;
        resize(w, h);
}


void Rectangle::getSize(double *w, double *h)
{
        *w = this->w;
        *h = this->h;
}
```

```
void Rectangle::resize(double w, double h)
{
        this->w = w;
        this->h = h;
}


double Rectangle::area()
{
        return ( w * h );
}
```

circle.h

```
#include "figure.h"
#ifndef circle_h
#define circle_h
class Circle : public Figure {
private:
        double r;
public:
        Circle();
        Circle(double r);
        Circle(double x, double y, double r);
        double getSize() { return r; }
        void resize(double r);
        virtual double area();
};
#endif
```

circle.cpp

```
#include "circle.h"
Circle::Circle()
{
        r = 0.0;
}

Circle::Circle(double r) : Figure(0.0, 0.0)
{
        this->r = r;
}

Circle::Circle(double x, double y, double r) : Figure(x, y)
{
        this->r = r;
}

void Circle::resize(double r)
{
        this->r = r;
}


double Circle::area()
{
        return ( 3.14159 * r * r );
}
```

figrtest.cpp

```
#include "figure.h"
#include "rectang.h"
#include "circle.h"

#include <iostream.h>

void main()
{
//      Figure fig;    // error: cannot create instance of abstract class

        Figure *fp1, *fp2;
        double x, y, w, h, r;
        void printArea( Figure* );

        Rectangle s;
        cout << "Rectangle s \n";
        cout << "Area of rectangle : " << s.area() << '\n';
        s.getCoord( &x, &y);
        s.getSize( &w, &h);
        cout << "x, y, w, h : "<< x << ", " << y << ", " << w << ", " << h << '\n';
        s.resize( 100.5, 10);
        s.setCoord(100, 200);
        s.getSize( &w, &h );
        s.getCoord( &x, &y );
        cout << "x, y, w, h : "<< x << ", " << y << ", " << w << ", " << h << '\n';
        cout << "Area of rectangle : " <<  s.area() << '\n';

        cout << "\nRectangle *fp1 \n";
        fp1 = new Rectangle(2, 3);
        fp1->getCoord( &x, &y);
        ((Rectangle*)fp1)->getSize( &w, &h);
        cout << "x, y, w, h : "<< x << ", " << y << ", " << w << ", " << h << '\n';
        cout << "Area of rectangle : " << fp1->area() << '\n';
        cout << "Area of rectangle : ";
        printArea( fp1 );
        delete fp1;

        cout << "\nCircle *fp2 \n";
        fp2 = new Circle(1);
        fp2->getCoord( &x, &y);
        r = ((Circle*)fp2)->getSize();
        cout << "x, y, r : "<< x << ", " << y << ", " << r <<  '\n';
        cout << "Area of circle : "<< fp2->area() << '\n';
        cout << "Area of circle : ";
        printArea( fp2 );
        delete fp2;
}

// Function printArea calls virtual member function area()
// of class Rectangle or Circle according to pointer received
void printArea( Figure *p )
{
        cout << p->area() << '\n';
}
```

```
Rectangle s
Area of rectangle : 0
x, y, w, h : 0, 0, 0, 0
x, y, w, h : 100, 200, 100.5, 10
Area of rectangle : 1005

Rectangle *fp1
x, y, w, h : 0, 0, 2, 3
Area of rectangle : 6
Area of rectangle : 6

Circle *fp2
x, y, r : 0, 0, 1
Area of circle : 3.14159
Area of circle : 3.14159
```

Daudzkārtējā mantošana

1. piemērs

```
#include <iostream.h>

class A {
public:
        int a;
        int b;
        A(){ a = 1; b = 2; }
};

class B {
public:
        int a;
        int c;
        B(){ a = 3; c = 4; }
};

class Sub : public A , public B {
public:
        int x;
        int y;
};

void main()
{
        Sub s;          // s inherits member a from class A and member a from class B
                        // and therefore reference s.a will be ambiguous

//      cout << s.a;            compiler error!

        cout << s.A::a << ',' << s.B::a << '\n';     // output: 1,3
        cout << s.b << ',' << s.c << '\n';           // output: 2,4
        cout << s.x << ',' << s.y << '\n';           // output: unpredictable
}
```

2. piemērs

```
#include <iostream.h>

class A{
public:
        int a;
        int b;
        A(){ a = 1; b = 2; }
};

class S1 : public A {
public:
        int x1;
        S1(){ x1 = 5;}
};

class S2 : public A {
public:
        int x2;
        S2(){ x2 = 6;}
};

class S12 : public S1, public S2 {
public:
        int x12;
        S12(){ x12 = 7;}
};

void main()
{
        S12 s;          // s inherits member a from class A through class S1
                        // and member a  from class A through class S2.
                        // There are two members of class S12 with the same name a.
        s.S2::a = 13;
        cout << s.S1::a << ',' << s.S2::a << '\n';   // output: 1,13
        cout << s.x1 << ',' << s.x2 << '\n';         // output: 5,6
}
```

3. piemērs

```
#include <iostream.h>

class A{
public:
        int a;
        int b;
        A(){ a = 1; b = 2; }
};

class Z1 : virtual public A {
public:
        int x1;
        Z1(){ x1 = 5;}
};

class Z2 : virtual public A {
public:
        int x2;
        Z2(){ x2 = 6;}
};

class Z12 : public Z1, public Z2 {
public:
        int x12;
        Z12(){ x12 = 7;}
};


void main()
{
        Z12 z;          // There is only one member z.a
        z.a = 13;
        cout << z.Z1::a << ',' << z.Z2::a << ',' << z.a << '\n';     // 13,13,13
        cout << z.x1 << ',' << z.x2 << '\n';                        // 5,6
}
```

Bāzes un atvasinātās klases. Konstruktoru un destruktoru darbība

figure.h

```
class Figure{
protected:
        double x;
        double y;
public:
        Figure();
        Figure(double, double);
        ~Figure();
        void setCoord(double x , double y) { this->x = x; this->y = y; }
        void getCoord(double *x , double *y) { *x = this->x; *y = this->y; }
        virtual double area() = 0;
};
```

rectang.h

```
#include "figure.h"
class Rectangle : public Figure {
private:
        double w;
        double h;
public:
        Rectangle();
        Rectangle(double w, double h);
        Rectangle(double x, double y, double w, double h);
        ~Rectangle();
        void setDim(double x, double y, double w, double h);
        void getSize( double *w, double *h);
        void resize(double w, double h);
        virtual double area();
};
```

rectfill.h

```
#include "rectang.h"
class RectangleFilled : public Rectangle {
private:
        int color;
public:
        RectangleFilled();
        RectangleFilled(double w, double h, int color);
        RectangleFilled(double x, double y, double w, double h, int color);
        ~RectangleFilled();
        int setColor(int c) { int prev = color; color = c; return prev;}
};
```

figure.cpp

```
#include "figure.h"
Figure::Figure()
{
        x = y = 0.0;
        cout << "Figure constructor\n";         //for demo only! }
}

Figure::~Figure()
{
             cout << "Figure destructor\n";  //for demo only!
}

Figure::Figure(double x, double y)
{
        setCoord( x, y);
}
```

rectang.cpp

```
#include "rectang.h"
#include <iostream.h>

Rectangle::Rectangle()
{
            w = h = 0.0;
            cout << "Rectangle constructor\n";      //for demo only!
}

Rectangle::~Rectangle()
{
            cout << "Rectangle destructor\n";       //for demo only!
}

Rectangle::Rectangle(double w, double h) : Figure(0.0, 0.0)
{
        this->w = w;
        this->h = h;
}
Rectangle::Rectangle(double x, double y, double w, double h) : Figure(x, y)
{
        this->w = w;
        this->h = h;
}
void Rectangle::setDim(double x, double y, double w, double h)
{
        this->x = x;
        this->y = y;
        resize(w, h);
}

void Rectangle::getSize(double *w, double *h)
{
        *w = this->w;
        *h = this->h;
}

void Rectangle::resize(double w, double h)
{
        this->w = w;
        this->h = h;
}


double Rectangle::area()
{
        return ( w * h );
}
```

rectfill.cpp

```
#include "rectfill.h"
#include <iostream.h>

RectangleFilled::RectangleFilled()
{
        this->color = 0;
        cout << "RectangleFilled constructor\n";     //for demo only!
}

RectangleFilled::~RectangleFilled()
{
        cout << "RectangleFilled destructor\n";      //for demo only!
}

RectangleFilled::RectangleFilled(double w, double h, int c) : Rectangle(w, h)
{
        this->color = c;
}

RectangleFilled::RectangleFilled(double x,double y,double w,double h, int c) :  Rectangle(x, y,
w, h)
{
        this->color = c;
}
```

condestr.cpp

```
#include "figure.h"
#include "rectang.h"
#include "rectfill.h"

#include <iostream.h>


void main()
{
        RectangleFilled *rfp;

        rfp = new RectangleFilled;

        Rectangle r;
        r.setDim(1,1,10,20);

        // ...

        rfp->setColor(15);
        rfp->setDim(0, 0, 3, 5);
        cout << "\nArea of filled rectangle is "<< rfp->area() << "\n\n";

        // ...

        delete rfp;
}
```

```
Figure constructor
Rectangle constructor
RectangleFilled constructor
Figure constructor
Rectangle constructor

Area of filled rectangle is 15

RectangleFilled destructor
Rectangle destructor
Figure destructor
Rectangle destructor
Figure destructor
```

Šabloni (Templates)

```
#include <iostream.h>

// Function template
template <class T>  T min( T a, T b)
        { return (a < b ? a : b); }

// Class template for stack implementation
template <class T>
class Stack{
private:
        T* stp;
        int size;
        int top;
public:
        Stack(int);
        void push(T);
        T pop();
};

template <class T> Stack<T>::Stack(int s)
{
        size = s;
        stp = new T[size];
        top = -1;
}

template <class T> void Stack<T>::push(T e)
{
        top++;
        stp[top] = e;
}

template <class T> T Stack<T>::pop()
{
        return stp[top--];
}

// My own class
class Auto{
private:
        long regnum;
        int  type;
public:
        Auto() { regnum = 0; type = 0; }
        Auto(long n, int t) { regnum = n; type = t; }
        long getRegnum() { return regnum; }
        int  getType() { return type; }
};

void main()
{// Using function template
        int i = 2;
        cout << min(i, 1) << '\n';
        double d = -5.5;
//      cout << min(d, 0) << '\n';    // error: Could not find match...!
        cout << min(d, 0.0) << '\n';
// Using class template
Stack <int> ist(12);
Stack<Auto> ast(12);
// or:
// typedef Stack<int> StackInt;
// typedef Stack<Auto> StackAuto;
// StackInt ist(12);
// StackAuto ast(12);
ist.push(100);
ist.push(200);
cout << ist.pop() << '\n';
cout << ist.pop() << '\n';
Auto a(1111, 1);
Auto b(2222, 2);
ast.push(a);
ast.push(b);
cout << ast.pop().getRegnum() << '\n';
cout << ast.pop().getRegnum() << '\n';
}
```

```
1
-5.5
200
100
2222
1111
```