

Unit 2

C++ Language I

Basics on C++ Programming

Takayuki Dan Kimura

- ◆ Input & Output
- ◆ Enhancements to C
- ◆ Pointers and References
- ◆ Class (Type)
- ◆ Needs for Protection
- ◆ Encapsulation
- ◆ Access Functions
- ◆ Constant Objects
- ◆ Queue Class

Input & Output

```
#include
<iostream.h>

void main()
{
    cout << "hello";
}
```

Program 1

Stream-based I/O

No format string

No printf

cin = input

cout = output

```
#include <iostream.h>

void main () {
    int x = 1, y = 1;
    int n;
    cout << "limit: ";
    cin >> n;

    for ( int i = 0; i < n; i++ ) {
        cout << i << ": " << y << ";\n";
        x = x + y;
        y = x - y;
    }
}
```

Program 2

limit: 10

0: 1;

1: 1;

2: 2;

3: 3;

4: 5;

5: 8;

6: 13;

7: 21;

8: 34;

9: 55;

Enhancements to C

Variable declarations

```
for ( int i = 0; i , 10; i++ )  
{ int t = a[i]; a[i] = b; b = t; };
```

Scope operator

```
#include  
int k = 2;  
void main() {  
    int i = 3;  
    cout << :: k << k;  
};
```

Program 3

Default function arguments

```
void foo( int i = 5, float a = 0.03 );  
  
foo( 8, 1.5 );           // foo( 8, 1.5 )  
foo( 3 );                // foo( 3, 0.03 )  
foo( );                  // foo( 5, 0.03 )
```

Enumerations

```
enum grade { fail, poor,  
average, good, excellent }  
// 0, 1, 2, 3, 4, 5
```

```
enum things  
{ a = 5, b, c = 1, d, e }  
// 5, 6, 1, 2, 3
```

The Const qualifier

```
const int size = 5;  
    char a[size];  
    cout << sizeof a;
```

Funcion Overloading

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

void display_time( const struct tm *tim )
{ cout << "1. It is now " << asctime( tim ); }

void display_time( const time_t *tim )
{ cout << "2. It is now " << ctime( tim ); }

void main ()
{ time_t    tim = time( NULL );
  struct tm *Ltime = localtime( &tim );

  display_time ( Ltime );
  display_time( &tim ); }

/*  output:
1. It is now Sun May 19 19:10:17 1996
2. It is now Sun May 19 19:10:17 1996
*/
```

Same function names but
different parameter lists

Different actual parameters

Program 4

References (Aliases for a variable)

References and Addresses

```
void main ()
{
    int  act = 123;
        cout << act;
    int  &refact = act;
        cout << '\n' << act << " " << refact;
    act++;
        cout << '\n' << act << " " << refact;
    refact++;
        cout << '\n' << act << " " << refact;
        cout << '\n' << &act << " - " << &refact;
}
```

Program 5

```
123
123 123
124 124
125 125
0x34E2 - 0x34E2
```

Pointers & References

```
void pp( char *s, int x )
{
    cout << endl << s << x;
};

void main()
{
    int    *x,
    int    y = 1;
    int    &z = y;

    pp( "y ", y );           // y 1
    pp( "&y ", (int) &y );    // &y 13546
    pp( "z ", (int) z );      // z 1
    pp( "&z ", (int) &z );    // &z 13546

    z++;
    *x = y;
    pp( "x ", (int) x );      // x 13563
    pp( "*x ", *x );          // *x 2
    pp( "&x ", (int) &x );    // &x 13548

    y = *x + 3;
    pp( "y ", y );           // y 5
    pp( "z ", (int) z );      // z 5
    pp( "&z ", (int) &z );    // &z 13546
}
```

Program 6

Parameter Passing by Reference

Efficient (as pointers); No special notation (->)

```
struct {    int    k;
          char  s[1000]; }

a = { 5, "big data structure here" };

void  valf( A v );           // call by value
void  ptrf( A *p );          // call by pointer
void  reff( A &r );           // call by reference

void main () {
    valf( a );                // passing the variable
    ptrf( &a );               // passing the pointer
    reff( a );                // passing the reference
}

void  valf( A v )             // passing by value
{ cout << v.k << v.s << endl; };

void  ptrf( A *p )            // passing by pointer
{ cout << p->k << p->s << endl; };

void  reff( A &r )            // passing by reference
{ cout << r.k << r.s << endl; };
```

reference to

address of

Program 7

Misleading Syntax and Side Effects

- Misleading syntax

```
valf( a );           // no modification of a is allowed
ptrf( &a );          // modification of a is possible
reff( a );           // no modification of a is allowed?
```

- Dangerous side effects

```
void print( int &x ) { cout << x; x = 0; };

void main( ) { int a = 5; print( a ); } // a is modified
```

- Return by reference

```
int    a = 0;           // Global variable
int    &f( ) { return a; };

void main( )
{ int i = f( ); f( ) = 5; } // a = 5
```

- Guidelines

- If the function **modifies** the parameter, use a **pointer**.
- If the function **does not**, use a **reference** to a constant.
- Use a reference for parameter passing only.

Class (Type)

= **struct** + encapsulation (protection wall)

```
struct A
{
    int x;
    int y;
};

A a;

a.x = a.y + 1;
```

=

```
class A
{
    public:
        int x;
        int y;
};

A a;

a.x = a.y + 1;
```

```
class A
{
    int x;
    int y;
};

A a;

a.x = a.y + 1;
```

```
class A
{
    int x;
    int y;
    public:
        int z;
};

A a;

a.x = a.y + 1;
a.z = a.z + 1;
```

Needs for Protection

What is wrong about this program?

```
#include <stdio.h>

struct date {
    int month;
    int day;
    int year;
}

void show_date( date &dt ) {
    static  char *name[] =
        { "zero", "January", "February", "March", "April", "May",
          "June", "July", "August", "September", "October",
          "November", "December"
        };
    printf( "%s %d, %d", name[dt.month], dt.day, dt.year ); }

void main() {
    date my_date,      your_date = { 31, 7, 1996 };
    my_date.day = 22;  my_date.month = 7;  my_date.year = 1996;

    show_date( my_date );
    show_date( your_date ); }

/*      Output
July 22, 1996
Application Error!
*/
```

Program 8

no control over accessing variables in **date**

C++ Solution (Creation of New Data Type)

```
class date {
public:
    date( int mn, int dy, int yr );           // Constructor (initialization)
    void show();                             // show_date( )
    ~date();                                 // Destructor (termination)
private:
    int month;
    int day;
    int year; };

inline int max( int a, int b ) { if( a > b ) return a; return b; }
inline int min( int a, int b ) { if( a < b ) return a; return b; }

date::date( int mn, int dy, int yr ) {
    static int length[] = { 0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };

    month = max( 1, mn );                    month = min( month, 12 );
    day = max( 1, dy );                      day = min( day, length[month] );
    year = max( 1, yr ); } // range checking

void date::show() {
    static char *name[] = {
        "zero", "January", "February", "March", "April", "May",
        "June", "July", "August", "September", "October",
        "November", "December" };

    cout << name[month] << ' ' << day << ", " << year; }

date::~date() { } // Do nothing

void main() {
    date my_date( 7, 22, 1996 );              // no assignment operation is allowed on
    date your_date( 31, 7, 1996 );            // the variables in the private part of date

    my_date.show(); cout << "\n";             // July 22, 1996
    your_date.show(); }                      // December 7, 1996
```

Program 9

Notes:

- *Class members*
 - * *Data members* month, day, year
 - * *Function members* date(), show(), ~date()
- *Member functions*
 - * *Constructor* date()
 - = non-value-returning initialization function with the same name as the class name, to be called automatically whenever an instance of the class is declared
 - * *Destructor* ~date()
 - = non-value-returning termination function to be called automatically whenever a class object goes out of scope
- *Class member visibility*
 - * *Public members* date(), show(), ~date()
 - = can be accessed by member functions and other functions in which a class object is in scope
 - = the class's interface
 - * *Private members* month, day, year
 - = can be accessed only by member functions
 - = the class's implementation
- *Scope definition* date::show()
 - = the name of a class followed by the scope operator to define the scope of member function definition

Emulation of Class in C

```
struct date {  
    int month;  
    int day;  
    int year;  
  
    void (*show)( date &dt );  
}  
  
void show_date( date &dt ) {    ...    }  
  
void main() {  
    date my_date, your_date = { 8, 13, 1996, &show_date };  
  
    my_date.day = 22;    my_date.month = 7;    my_date.year = 1996;  
    my_date.show = &show_date;  
  
    (* my_date.show)( my_date );    // July 22, 1996  
    show_date( my_date );    // July 22, 1996  
  
    (*my_date.show)( your_date );    // August 13, 1996  
}
```

Program 10

- No protection for invalid assignments
- No association between member functions and data
 - * Necessary to pass data for a member function call
 - * A member function can be applied to other objects
- Unclean syntax: (*show)(date &dt), (*my_date.show)(your_date)

Merit of Encapsulation (Maintainability)

Same interface with different implementation

```
class date {  
public:                                     // same interface as in Program 9  
    date( int mn, int dy, int yr );  
    void show();  
    ~date();  
private:  
    int monthday;                        // different implementation  
    int year; };  
...  
static int length[] = { 0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };  
  
date::date( int mn, int dy, int yr ) {  
    int    month, day;  
    month = max( 1, mn );      month = min( month, 12 );  
    day = max( 1, dy );      day = min( day, length[month] );  
    monthday = 0; for ( int i = 0; i < month; i++ ) monthday += length[i];  
    monthday += day;  
    year = max( 1, yr ); }  
  
void date::show() { static char *name[] = { "zero", "January", ... , "December" };  
    int    month = 0, day;  
    int    m = monthday;  
    while ( m > 0 ) { month++; m -= length[month]; }  
    day = m + length[month];  
    cout << name[month] << ' ' << day << ", " << year; }  
...  
  
void main() {                               // same as Program 9  
    date my_date( 7, 22, 1996 );  
    date your_date( 31, 7, 1996 );  
  
    my_date.show(); cout << '\n';           // July 22, 1996  
    your_date.show(); }                     // December 7, 1996
```

Program 11

Access Functions

```
class date {
public:
    date( int mn, int dy, int yr );
    void  show();
    ~date();
    date();                                // Overloaded default constructor
    int   getMonth();                      // Member functions
    int   getDay();
    int   getYear();
    void  setMonth( int mn );
    void  setDay( int dy );
    void  setYear( int yr );
private:
    int monthday;
    int year; };

...
static int length[] = { 0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };
static int month, day;

int  daysSoFar( int mn, int dy ) { int md = 0;    // local function
    for ( int i = 0; i < mn; i++ ) md += length[i]; md += dy; return md; }

int  date::getMonth() { int  m = monthday;
    month = 0; while ( m > 0 ) { month++; m -= length[month]; };
    return month; }

int  date::getDay() { int    m = monthday;
    month = 0; while ( m > 0 ) { month++; m -= length[month]; };
    return m + length[month]; }

inline int  date::getYear() { return year; }

void  date::setMonth( int mn ) {
    day = getDay();
    month = max( 1, mn ); month = min( month, 12 );
    monthday = daysSoFar( month, day ); }
```

Program 12 (1)

```
void date::setDay( int dy ) {
    month = getMonth();
    day = max( 1, dy );          day = min( day, length[month] );
    monthday = daysSoFar( month, day ); }

void date::setYear( int yr ) { year = max( 1, yr ); }

date::date( int mn, int dy, int yr ) {
    month = max( 1, mn );        month = min( month, 12 );
    day = max( 1, dy );          day = min( day, length[month] );
    monthday = daysSoFar( month, day );
    year = max( 1, yr ); }

date::date() { monthday = year = 1; }           // default constructor

void date::show() {
    static char *name[] = { "zero", "January", ... , "December" };
    month = getMonth();
    day = getDay();
    cout << name[month] << ' ' << day << ", " << year << '\n' }
...
void main() {
    date    my_date( 7, 22, 1996 );
    date    * mydatePtr = &my_date;
    date    dates[3];                                // need default constructor

    mydatePtr->show();                                // July 22, 1996

    dates[0] = my_date;
    dates[1].setYear( 1996 );
    for ( int i = 0; i < 3; i++ ) dates[i].show();
    // July 22, 1996      January 1, 1996      January 1, 1

    date    your_date;                                // need default constructor
    date    &the_date = your_date;
            the_date.setMonth( my_date.getMonth() );
            the_date.setDay( 45 );
            the_date.setYear( my_date.getYear() );
    your_date.show(); }                                // July 31, 1996
```

Program 12 (2)

Constant Objects (Constant Member Functions)

Access functions of a constant objects should be read-only.

Do we need to define another class? No!

```
class date { public: ...
    int    getMonth() const;    // available for constant objects (read-only)
    int    getDay() const;
    int    getYear() const;
    void   setMonth( int mn ); // not available for constant objects
    void   setDay( int dy );
    void   setYear( int yr );
    void   show() const;
    ... };

...
int    date::getMonth() const { ... }
int    date::getDay() const { ... }
int    date::getYear() const { ... }
...
void   date::show() const { ... }
...
void main() {
const date    my_date( 7, 22, 1996 );    // my_date cannot be modified
    date      your_date;                // your-date can be modified

    //my_date.setYear( 1995 ); compilation error
    // non-const functions of a const object are not accessible

    your_date.setMonth( my_date.getMonth() );
    your_date.setDay( 45 );
    your_date.setYear( my_date.getYear() );
    your_date.show();                // July 31, 1996

    your_date.setYear( 1995 );
    // non-const functions of a non-const object are accessible
    your_date.show(); }                // July 31, 1995
```

Program 13

Sneak Access by Reference (Bad Programming!)

```
class date
{ public:
    ...
    int    &mnth();          // get/set month
private:
    int month;
    int day;
    int year; };

...
int &date::mnth() {
    month = max( 1, month );
    month = min( month, 12 );
    return month; }

...
void main() {
    date    my_date( 7, 22, 1996 );
            my_date.show();                // July 22, 1996

    cout << "\nmonth: " << my_date.mnth() << "\n";    // month: 7

    my_date.mnth() = my_date.mnth() + 3;
    my_date.show();                }        // October 22, 1996
```

Program 14

- **mnth** returns reference to a private data member **month**.
⇒ **mnth** acts like a public data member for month
- Encapsulation is broken.
⇒ illegal value assignment to **month** is possible
⇒ **show()** may cause run-time error (for name[month]).

Header and Source Files

```
class date
{
public:
    date( int mn, int dy, int yr );
    date(); // default constructor

    int    getMonth() const;
    int    getDay() const;
    int    getYear() const;
    void    setMonth( int mn );
    void    setDay( int dy );
    void    setYear( int yr );

    void    show() const;
    ~date();
private:
    int monthday;
    int year;
};
```

DATE.H file

```
#include <iostream.h>
#include "DATE.H"

    inline int max( int a, int b ) { ... }
    inline int min( int a, int b ) { ... }
    static int length[] = { 0, 31, ..., 31 };
    static int month, day;
    int daysSoFar( int mn, int dy ) { ... }
int date::getMonth() const { ... }
int date::getDay() const { ... }
int date::getYear() const { ... }
void date::setMonth( int mn ) { ... }
void date::setDay( int dy ) { ... }
void date::setYear( int yr ) { ... }
date::date( int mn, int dy, int yr ) { ... }
date::date() { ... }
void date::show() const { ... }
date::~date() { }
```

DATE.CPP file

```
#include "DATE.H"

void main() {
const date my_date( 7, 22, 1996 );
    date your_date;

    your_date.setMonth(
        my_date.getMonth() );
    your_date.setDay( 45 );
    your_date.setYear(
        my_date.getYear() );
    your_date.show();

    your_date.setYear( 1995 );
    your_date.show();
}
```

TEST1.CPP file

```
#include "DATE.H"

void main()
{
    date today;

    today.setMonth( 7 );
    today.setDay( 22 );
    today.setYear( 1996 );
    today.show();
}
```

TEST2.CPP file

Composition

= making a new class by using other classes as components

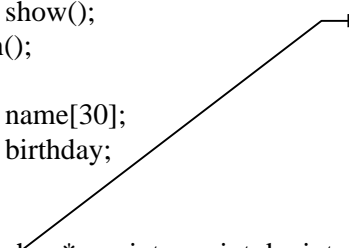
```
#include <iostream.h>
#include <string.h>
#include "date.h"

class person { public:
    person( char *nm, int mn, int dy, int yr );
    char    *getName() const;
    date    getBirthday() const;
    void    setName( char *nm );
    void    setBirthday( date bd );
    void    show();
    ~person();
private:
    char    name[30];
    const    date    birthday;
};

person::person( char *nm, int mn, int dy, int yr )
    :birthday( mn, dy, yr ) // member initializer, if const birthday, then indispensable
{    strncpy( name, nm, 30 ); }

inline    char    *person::getName() const { return name; }
inline    date    person::getBirthday() const { return birthday; }
void    person::setName( char *nm ) {    strncpy( name, nm, 30 ); }
void    person::setBirthday( date bd ) {    birthday = bd; }
void    person::show() {    cout << "\n" << name << " ";    birthday.show(); }
person::~~person() { }

void main() {
    person    boss( "Kimura", 1, 12, 1938 );           // date object is created
    boss.show();                                     // Kimura January 12, 1938
    date bd = boss.getBirthday();
    bd.setMonth( bd.getMonth() + 2 );
    boss.setName( "Kida" );
    boss.setBirthday ( bd );
    boss.show(); }                                  // Kida March 12, 1938
```



Date class constructor is called before the Person constructor begins execution

Program 15

More on Encapsulation (Queue without Type)

```
// Queue with no encapsulation
// only one queue is generated

#include <iostream.h>
#define    size 3

int  head = 0;
int  tail = 0;
int  length = 0;
int  buf[size];

void enq( int x ) {
    if ( length < size ) { buf[tail] = x;    tail = (tail + 1) % size; length++; }
    else { cout << "overflow" << endl; }; };

int deq() {
    if ( length > 0 ) { int temp = buf[head]; head = (head + 1) % size;
        length--; return( temp ); }
    else { cout << "underflow" << endl; return( 0 ); };

void main()
{
    enq( 9 ); enq( 8 ); enq( 7 ); enq( 6 );           // overflow

    buf[0] = 4; // should not be allowed

    cout << deq() << endl;           // 4
    cout << deq() << endl;           // 8
    cout << deq() << endl;           // 7
    cout << deq() << endl;           // underflow
                                         // 0
}
```

Program 16

Queue with Type

```
// Queue as a type
// More than one queue can be generated
#include <iostream.h>

struct    Queue { int  size, head, tail, length, *buf; };

void      enq( Queue &q, int x ) {
    if ( q.length < q.size ) { q.buf[q.tail] = x;  q.tail = (q.tail + 1) % q.size;  q.length++; }
    else { cout << "overflow" << endl; }; };

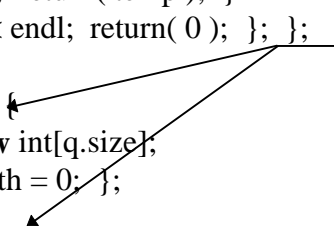
int       deq( Queue &q ) {
    if ( q.length > 0 ) { int temp = q.buf[q.head]; q.head = (q.head + 1) % q.size;
                        q.length--; return( temp ); }
    else { cout << "underflow" << endl; return( 0 ); }; };

void      init( Queue &q, int s ) {
    q.size = s; q.buf = new int[q.size];
    q.head = q.tail = q.length = 0; };

void      done( Queue &q ) { delete q.buf; };

void      show( Queue &q, char *s ) {
    cout    << s << endl
            << "size: " << q.size
            << " head: " << q.head
            << " tail: " << q.tail
            << " length: " << q.length << endl;
    for ( int i = 0; i < q.size; i++ ) cout << q.buf[i] << endl;
    cout    << endl; };

```



Memory allocation
operations in C++

Program 17 (1)

```
void main()
{
    Queue q1, q2;
    init( q1, 3 );
    init( q2, 5 );

    enq( q1, 9 );
    enq( q1, 8 );
    enq( q1, 7 );
    show( q1, "q1: " );

    q1.buf[1] = 6; // should not be allowed

    enq( q2, deq( q1 ) );
    enq( q2, deq( q1 ) );
    show( q2, "q2: " );
    show( q1, "q1: " );

    done( q1 );
    done( q2 );
};
```

Program 17 (2)

```
q1:
size: 3 head: 0 tail: 0 length: 3
9
8
7

q2:
size: 5 head: 0 tail: 2 length: 2
9
6
14260
0
0

q1:
size: 3 head: 2 tail: 0 length: 1
9
6
7
```

Output

Queue (Abstract Data Type)

```
class Queue {
public:
    Queue( const int max );      ~Queue( );
    void enq( int x );
    int  deq();
    void show( char *s );
private:
    int  size;          int  length;
    int  head;          int  tail;
    int  *buf;  }

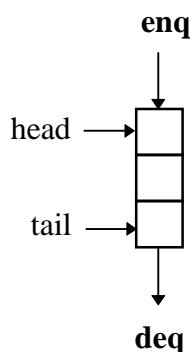
Queue::Queue( const int max ) {
    size = max;
    buf = new int[size];
    head = tail = length = 0; }

Queue::~Queue() { delete [] buf; }

void Queue::enq( int x ) {
    if ( length < size ) {
        buf[ tail ] = x;
        tail = ( tail + 1 ) % size;
        length++; }
    else { cout << "overflow" << endl; }; }

int Queue::deq() {
    if ( length > 0 ) {
        int temp = buf[ head ];
        head = ( head + 1 ) % size;
        length--;
        return( temp ); }
    else { cout << "underflow" << endl; return( 0 ); }; }

void Queue::show( char *s ) {
    cout << s;
    for ( int i = 0; i < length; i++ )
        cout << buf[ (head+i) % size ] << ' ';
    cout << endl; };
```



```
void main()\n{\n    Queue q1( 3 );\n    Queue q2( 5 );\n\n    q1.show( "q1: " );\n        // q1:\n    q1.enq(9);\n    q1.enq(8);\n    q1.enq(7);\n    q1.enq(6); // overflow\n\n    int t = q1.deq();\n    t = q1.deq();\n    q1.enq(5);\n\n    q1.show( "q1: " );\n        // q1: 7 5\n\n    // q1.buf[1] = 6;\n    // 'buf' : not accessible\n\n    q2.enq( q1.deq() );\n    q2.enq( q1.deq() );\n    q2.enq( q1.deq() );\n        // underflow\n    q2.show( "q2: " );\n        // q2: 7 5 0\n    q1.show( "q1: " );\n        // q1:\n}
```

Bounded Queue!

Program 18