Multimedia Programming 2004

Lecture 5

Erwin M. Bakker
Joachim Rijsdam

ANSI C Reserved Words

auto, break, case, char, const, continue, default, do, double, else, enum, extern, float, for, goto, if, int, long, register, return, short, signed, sizeof, static, struct, switch, typedef, union, unsigned, void, volatile, while

ANSI C++: class, private, public

Abstract Data Type: complex

```
typedef struct
{
    float Real;
    float Imaginary;
} complex

complex sum( complex z, complex w )
{
    complex s;
    s.Real = z.Real + w.Real;
    s.Imaginary = z.Imaginary + w.Imaginary;
    return s;
}

complex Sum, z1, z2;
...
z1 = …; z2 = …;
…
Sum = sum(z,1,z2);
```

Arrays

```
int i=1;
char s[10];          // s   now contains the address of   s[0]

s          is equivalent to   &s[0]
s + i      is equivalent to   &s[i]
&s[i]      is equivalent to   &s[0] + i      ( note, the compiler knows it is the
           address of an int => &s[0] + i  means i  integer locations further)

int a[10];

for ( i=0; i<10; i++ )  scanf("%d", &a[i] );

// does the same as
for ( i=0; i<10; i++ )  scanf("%d", a + i );

// and is equivalent to
for  (i=0; i<10; i++ )  scanf("%d", &a[0] + i );
```

Arrays

```
int minimum( int *b, int n)        // also:     int minimum( int b[], int n)
{
        int min, i;
        min = *b;
        min = b[0];              // means the same as previous statement
        for ( i=1;  i<n;  i++)
        {
                if ( *(b+i) < min ) min = *(b+i);

                if ( b[i] < min )
                    min = b[i];          // is equivalent to previous statement
        }
        return min;
}

int main(void)
{
    int a[10];          // a contains the address of a[0] !!!
    …
    printf( "Smallest value: %d\n", minimum(a,10) );
}
```

Dynamic Arrays

```
int myfunction(int b)
{
   int c[10000], i;

   for (i=0; i<b; i++)
   {
        scanf("%d",&c[i]);
   }
   ….

   return 0;
}
```

```
int myfunction(int b)
{
    int *d, i;
    d = new int[b];
    for ( i=0; i<b; i++)
    {
        scanf("%d",&d[i]);
    }
    …..
    delete d;
    return 0;
}
```

Multi Dimensional Arrays

```
float table[100][3];
float mtable[100][3][5];

int myfunction( float a[][3] );
int myfunction( float a[][3][5] );

// Initialization
int a[2][3] = {{1,2,3}, {3,4,5}};
char names[3][20] = {"pete","jane","bill"};

// Dynamic
int *list[30];              // 30 pointers to integers
for (i=0; i<30; i++)
{
     scanf("%d", &length);
     list[i] = new int[length];
}
…
delete [30] list;
```

C++ Classes

- Abstract Data Types: Struct + their operations
- C++ class is a concept to define data structures and their operations
- The actual implementation of the abstract data type can be hidden to the user!

C++ Classes: List Example

- Abstract Data Type List:
- Data:
  - Name
  - Address
  - Comments
- Basic operations:
  - Create: Create a list
  - Add: Add data to list
  - Delete: Delete data from the list
  - Find: Find data in the list
  - List: Make a print of the contents of the list
- We would like to hide the implementation to the user of our list

C++ Classes: CList

```
class CList                     \\ Declaration of CList
{
public:
   CList();                     \\ Mandatory constructor
   ~CList();                    \\ Mandatory destructor

   void Add(int element);       \\ The public operations
   void Delete(int element);    \\ Public member functions
   bool Find(int element);
   bool IsEmpty();
   void List()

private:                        \\ Private declarations
   int element_list[MAX_ELEMENTS]; \\ Data
   int number_of_elements;

   int FindPlace(int element);  \\ Private member function
};
```

C++ Classes: CList

```
class CList                     \\ Declaration of CList
void Add(int element);          \\ Public member functions
void Delete(int element);
bool Find(int element);
bool IsEmpty();
void List();
```

Is everything the user needs to know to use the class CList

```
int main( void )
{
    CList list;

    list.Add(2);
    list.Add(5);
    list.Delete(2);
    list.List();
    return 0;
}
```

C++ Classes Declaration

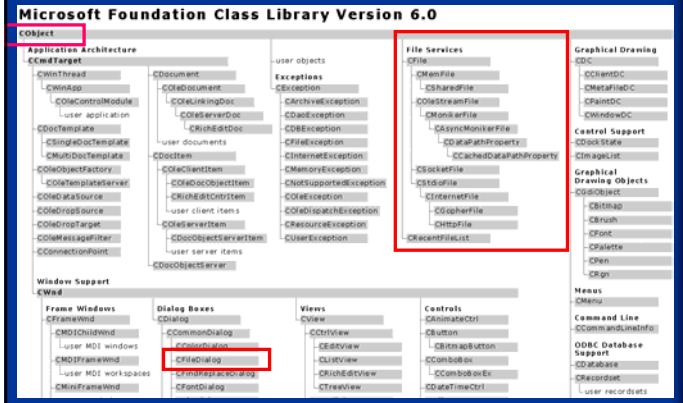
```
Class CMyClass
{
public:
      CMyClass();
      ~CMyClass();

      void memberfunction_this(int element);
      void memberfunction_that(int element);
      …
      int my_data;
      int my_array[10];
      int *my_pointers;
      …

private:
      int my_private_memberfunctions();
      …
      int my_private_data;
      …

Public:
      … etc.
};
```

```
CMyClass Examp;
Examp.my_data = 1;
Examp.memberfunction_this(3);
```

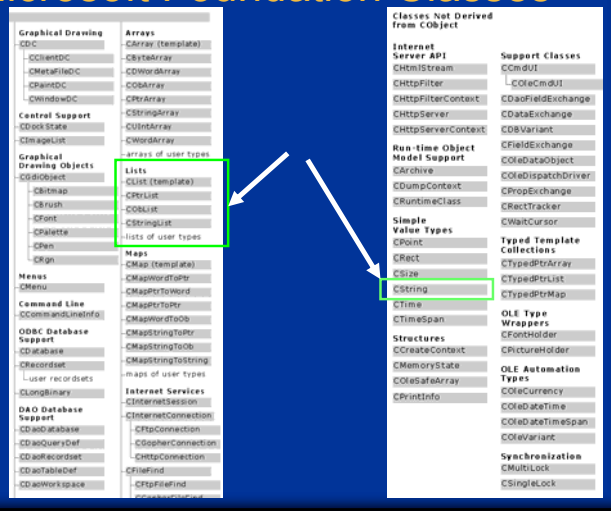
Using Existing C++ Classes

- CString
- CFile
- CList
- Microsoft Foundation Class Library (MFC)
- Etc.

Microsoft Foundation Classes



Microsoft Foundation Classes



CString

Construction

- CString Constructs **CString** objects in various ways.

The String as an Array

- GetLength Returns the number of characters in a **CString** object. For multibyte characters, counts each 8-bit character; that is, a lead and trail byte in one multibyte character are counted as two characters.
- IsEmpty Tests whether a **CString** object contains no characters.
- Empty Forces a string to have 0 length.
- GetAt Returns the character at a given position.
- operator [] Returns the character at a given position — operator substitution for **GetAt**.
- SetAt Sets a character at a given position.
- operator LPCTSTR Directly accesses characters stored in a **CString** object as a C-style string.

Assignment/Concatenation

- operator = Assigns a new value to a **CString** object.
- operator + Concatenates two strings and returns a new string.
- operator += Concatenates a new string to the end of an existing string.

CString

Comparison

- operator == <, etc. Comparison operators (case sensitive).
- Compare Compares two strings (case sensitive).
- CompareNoCase Compares two strings (case insensitive).
- Collate Compares two strings (case sensitive, uses locale-specific information).
- CollateNoCase Compares two strings (case insensitive, uses locale-specific information).

Extraction

- Mid Extracts the middle part of a string (like the Basic MID$ function).
- Left Extracts the left part of a string (like the Basic LEFT$ function).
- Right Extracts the right part of a string (like the Basic RIGHT$ function).
- SpanIncluding Extracts a substring that contains only the characters in a set.
- SpanExcluding Extracts a substring that contains only the characters not in a set.

CString

Other Conversions

- MakeUpper Converts all the characters in this string to uppercase characters.
- MakeLower Converts all the characters in this string to lowercase characters.
- MakeReverse Reverses the characters in this string.
- Replace Replaces indicated characters with other characters.
- Remove Removes indicated characters from a string.
- Insert Inserts a single character or a substring at the given index within the string.
- Delete Deletes a character or characters from a string.
- Format Format the string as **sprintf** does.
- FormatV Formats the string as **vsprintf** does.
- TrimLeft Trim leading whitespace characters from the string.
- TrimRight Trim trailing whitespace characters from the string.
- FormatMessage Formats a message string.

CString

Searching

- Find Finds a character or substring inside a larger string.
- ReverseFind Finds a character inside a larger string; starts from the end.
- FindOneOf Finds the first matching character from a set.

Archive/Dump

- operator <<Inserts a **CString** object to an archive or dump context.
- operator >>Extracts a **CString** object from an archive.

CString

Buffer Access

- GetBuffer Returns a pointer to the characters in the **CString**.
- GetBufferSetLength Returns a pointer to the characters in the **CString**, truncating to the specified length.
- ReleaseBuffer Releases control of the buffer returned by
- GetBuffer.FreeExtra Removes any overhead of this string object by freeing any extra memory previously allocated to the string.
- LockBuffer Disables reference counting and protects the string in the buffer.
- UnlockBuffer Enables reference counting and releases the string in the buffer.

Windows-Specific

- AllocSysString Allocates a **BSTR** from **CString** data.
- SetSysString Sets an existing **BSTR** object with data from a **CString** object.
- LoadString Loads an existing **CString** object from a Windows resource.
- AnsiToOem Makes an in-place conversion from the ANSI character set to the OEM character set.
- OemToAnsi Makes an in-place conversion from the OEM character set to the ANSI character set.

CList: member functions

Construction
- **CList** Constructs a **CList** object.

Head/Tail Access
- **GetHead** Returns the head element of the list (cannot be empty).
- **GetTail** Returns the tail element of the list (cannot be empty).

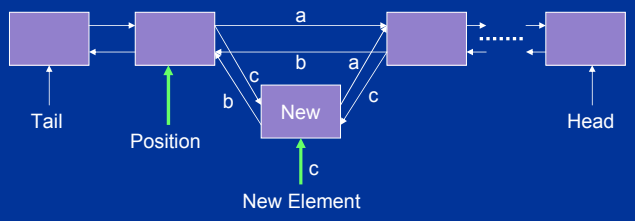
Operations
- **RemoveHead** Removes the element from the head of the list.
- **RemoveTail** Removes the element from the tail of the list.
- **AddHead** Adds an element (or all the elements in another list) to the head of the list (makes a new head).
- **AddTail** Adds an element (or all the elements in another list) to the tail of the list (makes a new tail).
- **RemoveAll** Removes all the elements from this list.

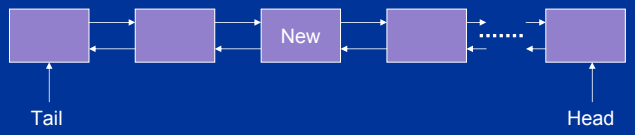
Status
- **GetCount** Returns the number of elements in this list.
- **IsEmpty** Tests for the empty list condition (no elements).

CList

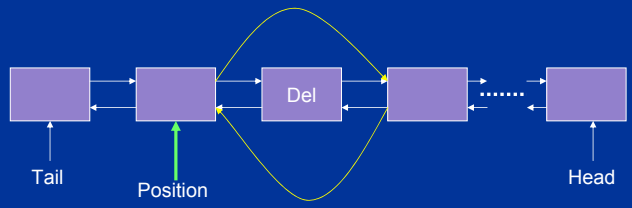
- The **CList** class supports ordered lists of non-unique objects accessible sequentially or by value.
- **CList** lists behave like doubly-linked lists.

CList: Insert After Position





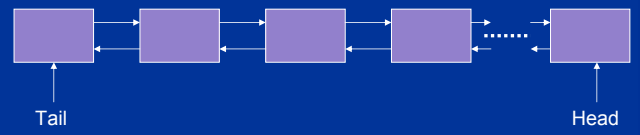
CList: Deletion After Position



CList

The **CList** class supports ordered lists of nonunique objects accessible sequentially or by value. **CList** lists behave like doubly-linked lists.

template< class *TYPE*, **class** *ARG_TYPE* **>class CList : public CObject**



CList: member functions

Iteration
- **GetHeadPosition** Returns the position of the head element of the list.
- **GetTailPosition** Returns the position of the tail element of the list.
- **GetNext**Gets the next element for iterating.
- **GetPrev** Gets the previous element for iterating.

Retrieval/Modification
- **GetAt** Gets the element at a given position.
- **SetAt** Sets the element at a given position.
- **RemoveAt** Removes an element from this list, specified by position.

Insertion
- **InsertBefore** Inserts a new element before a given position.
- **InsertAfter** Inserts a new element after a given position.

Searching
- **Find** Gets the position of an element specified by pointer value.
- **FindIndex** Gets the position of an element specified by a zero-based index.

Structured Programming

- Layout (your own style but consequent)
- Comment to clarify your code
- Naming: variables, functions, classes, etc. should be meaningful
- Appropriate control structures (if then, while, for) should be used
- Functions, and member functions should make the right division of tasks (careful top-down design)
- Data abstraction