Multimedia Programming 2004

Lecture 4

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

Structuring Your Program
Address List

Address List Example:

```
while (continue)
do
        read operation
        case add:
                read name
                read address
                read telephone number
                read comments
                add name … to list
        case delete:
                read name
                delete name from list
        case find:
                read name
                find name
                print name
        case list:
                print sorted address list
od
```

Structuring Your Data
Address List

```
record person
        name
        private address
        telephone number
        comment
end record
```

```
record address
        street
        zip code
        place
        country
end record
```

```
read_record(person)
{
        …
        read_record(address)
        read(comment)
        …
}
```

```
read_record(address)
{
        …
        read(street)
        read(zip code)
        read(place)
        read(country)
        …
}
```

Structuring Your Data: typedef struct Address List

```
record person
        name
        private address
        telephone number
        comment
end record
```

```
record address
        street
        zip code
        place
        country
end record
```

```
typedef struct
{
        string name;
        address priv_address;
        string tel_number;
        string comment;
} person;
```

```
typedef struct
{
        string street;
        string zip code;
        string place;
        string country;
} address;
```

Structuring Your Data: typedef struct Address List

```
typedef struct
{
        string Name;
        address Address;
        string Tel_Number;
        string Comment;
} person;
```

```
typedef struct
{
        string Street;
        string Zip_Code;
        string Place;
        string Country;
} address;
```

```
void print( person Person)
{
        printf( "Name: %s\n", Person.Name );
        print( Person.Address );
        printf( "Tel.: %s\n", Person.Tel_Number );
        printf( "Remarks: %s\n", Person.Comment );
}

Printf("Street: %s/n", Person.Address.Street);
```

Structuring Your Data: typedef struct Address List

```
typedef struct
{
        string Name;
        address Address;
        string Tel_Number;
        string Comment;
} person;
```

```
typedef struct
{
        string Street;
        string Zip_Code;
        string Place;
        string Country;
} address;
```

```
address Address = {"Dorpstraat 4", "2314 AA", "Amsterdam", "NL"};
person Person = {"J. Jansen", Address, "020-5430299", "Customer"};

print(Person);
read(Person); // ?how?
```

Functions

- void print_header(void)

No parameters, no results

- int main(void)

No parameters, result an int

- int smallest(int x, int y, int z)

Three integer parameters and result int

Functions: Parameters by Value

```
void swap(int x, int y)
{
        int temp;
        temp = y;
        y = x;
        x = temp;
}

void main(void)
{ int a =10, b = 11; swap(a,b); }
```

- Parameters are given by value: x and y can be seen as local variables that are initialized at the call of the function
- After swap(a,b) executed, the variables a and b passed through the parameters will still have their original values, as only their respective values have been passed to the local int x and int y!

Structuring Your Data: Address List

```
typedef struct
{
        string Name;
        address Address;
        string Tel_Number;
        string Comment;
} person;
```

```
typedef struct
{
        string Street;
        string Zip_Code;
        string Place;
        string Country;
} address;
```

```
void read(person Person) // ?how?
{
        scanf("%s", Person.Name);
        read(Address);
        scanf("%s", Person.Tel_Number);
        scanf("%s", Person.Comment);
}
```

// Will not work!

Pointers

Two unary operators
- * the *contents* of
- & the *address* of

The *address* of a variable a is the number of a memory location at which the value of a is stored:

byte a = 5;

&a → address = 10

*&a → 5

address	MEMORY
0	0x01
1	0xaa
2	0x34
3	0x35
4	0x56
5	0x11
6	0x34
7	0x55
8	0x45
9	0x00
10	0x05
11	0x00
12	0xde
…	…

&a := the address of variable a
*&a := the contents at the address of variable a

Pointers

Two unary operators
- * the *contents* of
- & the *address* of

The *address* of a variable a is the number of a memory location at which the value of a is stored:

int a = 1289;

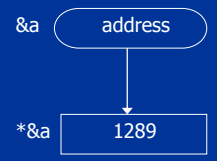
&a → address = 10

*&a → 1289 = 5*256+9

address	MEMORY
0	0x01
1	0xaa
2	0x34
3	0x35
4	0x56
5	0x11
6	0x34
7	0x55
8	0x45
9	0x00
10	0x05
11	0x09
12	0xde
…	…

byte order:
- most significant byte first: big-endian
- least significant byte first: little-endian

Pointers



Two unary operators
- * the contents of
- & the address of

```
void swap(int *x, int *y)      // The contents  *x  of address  x  is of type int
{                              // The contents  *y  of address  y  is of type int
        int temp;
        temp = *y;             // - temp becomes the contents of address  y
        *y = *x;               // - address  y  is equal to  &a  the address of a
        *x = temp;             //   thus  temp  becomes the contents of the address of a (=10)
}                              // - the contents of the address of  y  becomes
                               //   the contents of the address of  x
                               // - the contents of the address of  x  becomes  temp
void main(void)
{
    int a =10, b = 11;
    swap( &a, &b );     // the local variable x (address of type int) becomes the address of a
    printf("a: %d    b: %d", a, b);
}
```

Output: a: 11 b: 10

Pointers Special

```
int *p;                 // p is called a pointer variable
int q;
p =&q;
q = 10;
scanf("%d", p);
scanf("%d", &q);
printf("p: %d   q: %d", *p, q);

// equivalent declaration for p:
typedef int *ptr;
ptr p;

// we also could have declared:
int *p, q;              // one pointer to int, and one int

// NOTE: this is not equal to:
int *p, *q;             // two integer pointers
```

Pointers of Type VOID

```
int k;
char *pc;// pointer to a char

although internal address coding are equal for  addresses of char and
    int :
pc = &k;            // error, or warning at least
pc = (char *)&k;    // cast to pointer to  a char

int i, j;
void *p_general;   // address of arbitrary type
p_general = &i;
j = *p_general;                 // error: * can not be applied to void type
j = *(int *)p_general;          // after a cast to (int *), a pointer to int
                                // * can be applied !
                                // so that the value of  i  is assigned to  j
```

C++ Complex Data Types

- Basic Data Types
  - char, enum, short, int, long, float, double, long double, unsigned, signed, bool
- Complex Data Types
  - Pointers √
  - Arrays
  - Structs √
  - Classes
- Implementation of Abstract Data Types
  - complex numbers
  - real vectors
  - etc,

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);
```

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
- Funtions, and member functions should make the right division of tasks (careful top-down design)
- Data abstraction

DirectX 9b

DIRECTSHOW

DirectShow Video Formats

- DirectShow is an open architecture, it can support any format as long as there are filters to parse and decode it. (*= Windows Media® Format SDK needed to support this format.)
- Windows Media® Audio (WMA), Windows Media® Video (WMV), Advanced Systems Format (ASF)*
- Motion Picture Experts Group (MPEG)
- Audio-Video Interleaved (AVI)
- QuickTime (version 2 and lower)
- WAV, AIFF, AU, SND, MIDI

Compression formats:

- Windows Media Video, ISO MPEG-4 video version 1.0, Microsoft MPEG-4 version 3, Sipro Labs ACELP*
- Windows Media Audio*
- MPEG Audio Layer-3 (MP3) (decompression only)
- Digital Video (DV)
- MPEG-1 (decompression only)
- MJPEG
- Cinepak
- DirectShow-compatible MPEG-2 decoders are available from third parties

DirectShow (Headers and Libs)

Header Files

- **Dshow.h**
- Some DirectShow interfaces require additional header files. These requirements are noted in the interface reference.

Library Files

- **Strmiids.lib** Exports class identifiers (CLSIDs) and interface identifiers (IIDs).
- **Quartz.lib** Exports the AMGetErrorText function. If you do not call this function, this library is not required.

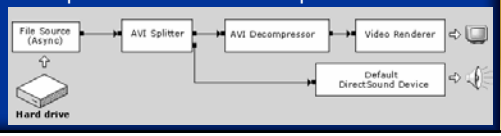
Note: In your build environment, place the DirectX SDK Include and Lib directories first in the Visual Studio search path. This ensures that you are using the most recent versions of these files.

Direct Show Application Programming

Filters and Filter Graphs

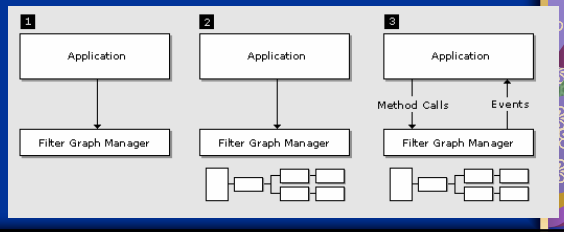
The building block of DirectShow is a software component called a *filter*. A filter is a software component that performs some operation on a multimedia stream. For example, DirectShow filters can

- read files
- get video from a video capture device
- decode various stream formats, such as MPEG-1 video
- pass data to the graphics or sound card
- Filters receive input and produce output. For example, if a filter decodes MPEG-1 video, the input is the MPEG-encoded stream and the output is a series of uncompressed video frames.



Writing a DirectShow Application

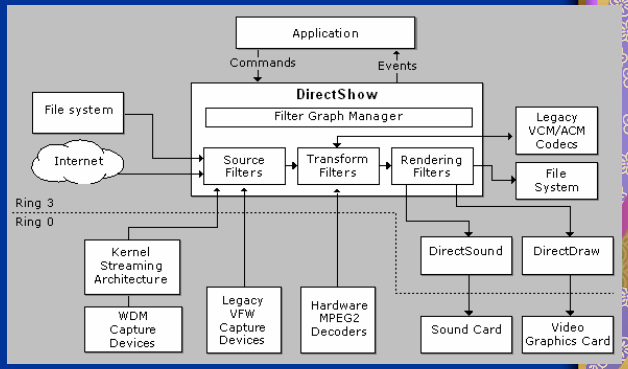
- The application creates an instance of the Filter Graph Manager.
- The application uses the Filter Graph Manager to build a filter graph. The exact set of filters in the graph will depend on the application.
- The application uses the Filter Graph Manager to control the filter graph and stream data through the filters. Throughout this process, the application will also respond to events from the Filter Graph Manager.
- When processing is completed, the application releases the Filter Graph Manager and all of the filters.



DirectShow and COM

- DirectShow is based on COM; the Filter Graph Manager and the filters are all COM objects. You should have a general understanding of COM client programming before you begin programming DirectShow. The article "Using COM" in the DirectX SDK documentation is a good overview of the subject. Many books about COM programming are also available.

Video Data Flow



DirectShow Play a File

- DirectShow application always performs the same basic steps:
- Creates an instance of the Filter Graph Manager.
- Uses the Filter Graph Manager to build a filter graph.
- Runs the graph, which causes data to move through the filters.

DirectShow Play a File

Start by calling **CoInitialize** to initialize the COM library:

- HRESULT hr = CoInitialize(NULL);

Call **CoCreateInstance** to create the Filter Graph Manager:

- IGraphBuilder *pGraph;
- HRESULT hr = CoCreateInstance(CLSID_FilterGraph, NULL, CLSCTX_INPROC_SERVER, IID_IGraphBuilder, (void **)&pGraph);

The Filter Graph Manager is provided by an in-process DLL, so the execution context is CLSCTX_INPROC_SERVER.

The call to **CoCreateInstance** returns the **IGraphBuilder** interface, which mostly contains methods for building the filter graph.

DirectShow Play a File

Two other interfaces are needed for this example:

- IMediaControl controls streaming. It contains methods for stopping and starting the graph.
- IMediaEvent has methods for getting events from the Filter Graph Manager. In this example, the interface is used to wait for playback to complete.

Both of these interfaces are exposed by the Filter Graph Manager. Use the returned **IGraphBuilder** pointer to query for them:

- IMediaControl *pControl; IMediaEvent *pEvent;
- hr = pGraph->QueryInterface(IID_IMediaControl, (void **)&pControl);
- hr = pGraph->QueryInterface(IID_IMediaEvent, (void **)&pEvent);

DirectShow Play a File

Now you can build the filter graph. For file playback, this is done by a single method call:

- hr = pGraph->RenderFile(L"C:\\Example.avi", NULL);

The IGraphBuilder::RenderFile method builds a filter graph that can play the specified file. The first parameter is the file name, represented as a wide character (2-byte) string. The second parameter is reserved and must equal NULL.

- This method can fail if the specified file does not exist, or the file format is not recognized. Assuming that the method succeeds, however, the filter graph is now ready for playback.

DirectShow Play a File

To run the graph, call the IMediaControl::Run method:

- hr = pControl->Run();

When the filter graph runs, data moves through the filters and is rendered as video and audio. Playback occurs on a separate thread.

You can wait for playback to complete by calling the IMediaEvent::WaitForCompletion method:

- long evCode = 0;
- pEvent->WaitForCompletion(INFINITE, &evCode);

This method blocks until the file is done playing, or until the specified time-out interval elapses. The value INFINITE means the application blocks indefinitely until the file is done playing.

When the application is finished, release the interface pointers and close the COM library:

- pControl->Release(); pEvent->Release(); pGraph->Release(); CoUninitialize();

Assignments (Problems)

- assignments 1-4 are related to playing video files
- assignment 5 is about Visual C++ classes and pointers
- the advanced assignment is about sound and video ‘thumbnails’