Multimedia Programming 2004

Lecture 3

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

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 );
}
```

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);
}
```

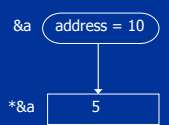
// Will not work!

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

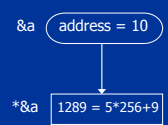
address	MEMORY
0	0x01
1	0xaa
2	0x34
3	0x35
4	0x56
5	0x11
6	0x34
7	0x55
8	0x45
9	0x00
10	0x05
11	0x00
12	0xde
…	…

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:

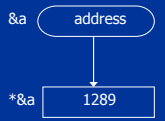
int a = 1289;



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

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
…	…

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;
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 that, the compiler knows it is the
                                              address of an int => i  int's 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 a[10];

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

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

//and is equivalent to
for (i=0; i<10; i++) printf("%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<10;  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 *lists[30];              // 30 pointers to integers
for (i=0; i<30; i++)
{
    scanf("%d",&length);
    list[i] = new int[length];
}
…
delete [30] list;
```

Sound & Sound Formats

- DirectMusic & DirectSound
- Sound Formats
  - Wav
  - Midi (Musical Instrument Digital Interface)
  - DLS (Downloadable Sounds)
  - DirectMusic Producer Segments
  - Loader for new formats

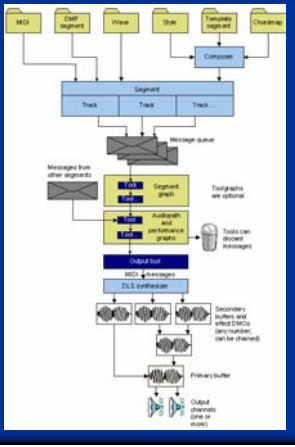
DirectMusic vs DirectSound

- DirectMusic more oriented to high level playing of arrangements and compositional effects (MIDI, DLS)
- DirectSound has sound recording and low level buffer operations (full duplex capabilities)

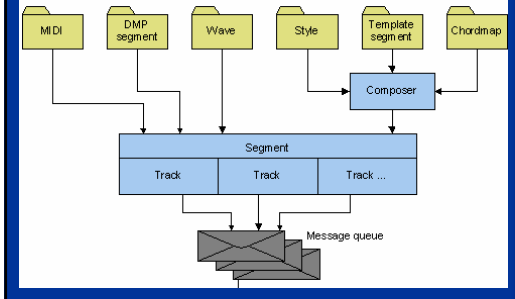
Playing Audio Files

- Initialize:
  - create instances of the loader and the performance
  - initialize the performance
- Load a File:
  - load the file and get an interface to that file (segment)
- Play the File:
  - download all kind of settings from the segment to the performance
  - play the segment
- Close Down:
  - close all the stuff

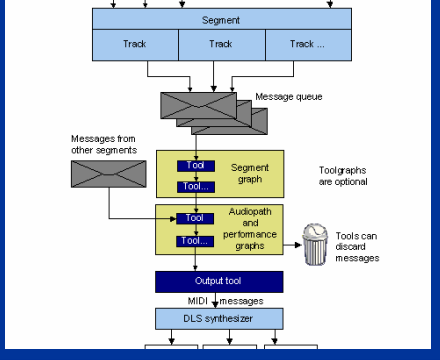
Sound Data Flow



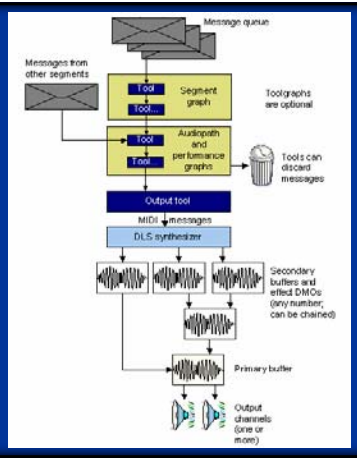
Sound Input



Sound Internal



Sound Output



Elements of a DirectMusic Application

Key concepts and code objects of DirectMusic:

- Loader
- Segments and Segment States
- Performance
- Messages
- Performance Channels
- Downloadable Sounds
- Instruments and Downloading
- Audio paths and Buffers
- Audio Scripts

Sound Data Flow

Key concepts and code objects

- Loader
- Segments and Segment States
- Performance
- Messages
- Performance Channels
- Downloadable Sounds
- Instruments and Downloading
- Audio paths and Buffers
- Audio Scripts