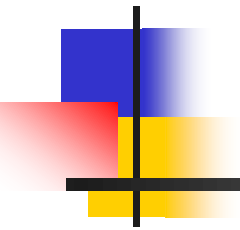

Chapter 1. Introduction to the Design and Specification of File Structures

Ki-Joon Han
School of Computer Science & Engineering
Konkuk University
kjhan@db.konkuk.ac.kr

Chapter Outline

1.1 The Heart of File Structure Design

1.2 A Short History of File Structure Design

1.3 A Conceptual Toolkit: File Structure Literacy

1.4 An Object-Oriented Toolkit: Making File Structure Usable

1.5 Using Objects in C++

1.1 The Heart of File Structure Design

- Disk
  - Very slow
  - Access time =>100만배 차이
    - RAM: 120 ns (20 sec) =>10ns 이하
    - Disk: 30 ms(5,000,000sec =58 days) : 25만배=>10ms 이하
  - Provide enormous, nonvolatile capability at low cost
- File structure
  - A combination of representation for data in files and of operations for accessing the data
- Good file structure design
  - Access to all the capacity without making applications spend a lot of time for the disk
  - Make an application hundreds of times faster
  - What is best for one situation may be terrible for another

1.2 A Short History of File Structure Design (1/2)

- Goals of R&D for file structure design problems
  - To get the information we need with one access to the disk
  - To find the target information with as few access as possible (i.e., 2 or 3 accesses)

  => To group information to get everything with only one access
- Sequential access -> Direct access
  - Tape -> Disk
  - Index : <key, pointer> in smaller file

1.2 A Short History of File Structure Design (2/2)

- B-tree
  - Balanced tree structure
  - Excellent access performance, but sequential access with a cost
- B^+-tree
  - Combination of a B-tree and a sequential linked list
  - Both sequential access and direct access
  - 3 or 4 accesses to the disk for millions of records
- Hashing
  - For files that do not change size greatly over time
  - One access to files
- Extendible, dynamic hashing
  - One or, at most, two disk accesses no matter how big the file become

1.3 A Conceptual Toolkit: File Structure Literacy

- Design tools for design problems
  - Decrease the number of disk accesses by collecting data into buffers, blocks, or buckets
  - Manage the growth of these collections by splitting them

  => Finding new ways to combine these basic tools of file design
- Conceptual tools
  - Methods of framing and addressing a design problem
  - Each tool combines ways of representing data with specific operations

1.4 An Object-Oriented Toolkits : Making File Structures Usable

- File structures usable in applications
  - Conceptual toolkit -> application programming interface (i.e., collections of data types and operations)
- Object-Oriented approach
  - Data types and operations are presented as *class* definitions
  - The C++ programming language is used in this text
- File structure
  - Represented by one or more classes of objects

1.5 Using Objects in C++ (1/4)

- Class
  - data attributes (*data members*)
  - functions (*member functions* or *methods*)
- Features of C++
  - class definitions, constructors, public and private sections, and operator overloading
  - inheritance, virtual functions, and templates
- This book
  - illustrate the principles of object-oriented design through implementations of file structures and file operations as C++ classes

1.5 Using Objects in C++ (2/4)

- C++ class **Person**

```
class Person
{   public:
      char LastName[11], FirstName[11], Address[16];      // data members
      char City[16], State[3], Zipcode[10];
      person();                                           // method
                                                          // default constructor
};
```

- Object creation

```
Person p;                          // automatic creation
Person * p_ptr = new Person        // dynamic creation
```

- **person** constructor

```
Person :: person()                 // set each field to an empty string
{
      LastName[0] = 0; FirstName[0] = 0; Address[0] = 0;
      City[0] = 0; State[0] = 0; Zipcode[0] = 0;
}
```

1.5 Using Objects in C++ (3/4)

```
class String
{public:
    String ( );                      // default constructor
    String (const String&);          // copy constructor
    String (const char *);           // create from C string
    ~ String ( );                    // destructor
    String & operator = (const String &);        // assignment
    int operator == (const String &) const;      // equality
    char * operator char*( )         // conversion to char *
          {return strdup(string);} // inline body of method
 private:
    char * string;    // represent value as C string
    int MaxLength;
};
```

=> a conversion operator (operator char *)

- allow the use of the value of a String object as a C string

1.5 Using Objects in C++ (4/4)

```
String s1("abcdefg");   //uses String::String (const char*)
char str[10];
strcpy (str, s1);           //uses String::operator char *()
```

=> create a String object s1 and copy its value to normal C string

```
s2 = s1;
```

=> copy the value of s1.string (a pointer) to s2.string

=> s1 and s2 become aliases

```
String & String:: operator = (const String & str)
{ //code for assignment operator
        strcpy(string, str.string)
        return *this;
}
String s1, s2;
s1 = s2;    //using overloaded assignment
```

=> does not create the alias problem