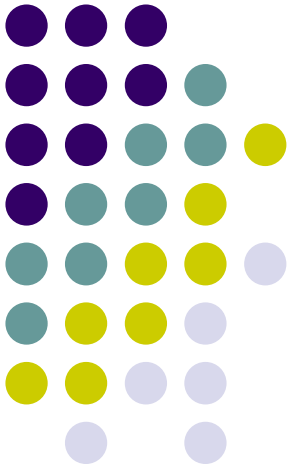
File Structures

Ch01. A. Introduction to the Design and Specification of File Structures



2020. Spring

Instructor: Joonho Kwon

jhkwon@pusan.ac.kr

Data Science Lab @ PNU

Slides are based on the textbook.

datalab

data science laboratory

References

- For the basic C++ concepts
  - https://www3.ntu.edu.sg/home/ehchua/programming/cpp/cp3_OOP.html

Chapter Objectives

- Introduce the primary design issues characterizing file structure design
- Survey the history of file structure design
- Introduce the notions of file structure literacy and of a *conceptual toolkit* for file structure design
- Discuss the need for specification of data structures, operations and development of *object-oriented toolkit*
- Introduce classes and overloading in C++

Outline

- 1.1 The heart of file structure design
- 1.2 History of file structure design
- 1.3 Conceptual toolkit : file structure literacy
- 1.4 Object-Oriented Toolkit
  : make file structure usable
- 1.5 Using objects in C++

What are File Structures?

- Definition
  - A ***File Structure*** is a combination of ***representations*** for data in files and of ***operations*** for accessing the data.
  - Functionalities of a file structures
    - allows applications to ***read***, ***write*** and ***modify*** data.
    - **finds** the data that matches some search criteria
    - or ***reads through*** the data in some particular order

Files

- **Sequence of bytes… nothing more, nothing less**
- **File System** resides on secondary storage (disks)

History of file structure design (1/2)

- Sequential file
  - look at records in order
- AVL tree (Balanced Binary Tree)
  - self-adjusting binary tree
  - guarantee good access time for data in RAM
- B-tree
  - balanced tree structure
  - provide fast access to data in file

History of file structure design (2/2)

- B^+-tree
  - variation on B-tree structure
  - provide both sequential access and fast-indexed access
- Hashing
  - transform search key into storage address
  - provide fast access to stored data
- Extendible hashing
  - approach to hashing that works well with files undergoing many changes in size over time

Conceptual Toolkit : File Structure Literacy

- Objective of Conceptual toolkit
  - Fundamental file concepts
  - Generic file operations

- Conceptual tools in this book
  - basic tools
    - Chapter 2 ~ 6
  - evolution of basic tools
    - Chapter 7 ~ 12
    - **B-trees, B+trees, hashed indexes, and extensible dynamic hashed files**

Object-Oriented Toolkit: Making File Structures Usable

- Object-Oriented Toolkit
  - making file structures usable requires turning *conceptual toolkit* into *collections* :*(classes)* of *data types* and *operations*

- Major problem
  - complicated and progressive
    - often modified and extended from other classes and details of classes become more complex

Using objects in C++ (1/2)

- Features of object in C++
  - class definition
    - data members(attributes) + methods
  - constructor
    - provide a guarantee for initialization of every object & called in creation time of object
  - public, private & protected sections
    - public label specifies that any users can freely access
    - private & protected label are restricted access

Using objects in C++ (2/2)

- Features of object in C++ (cont.)
  - operator overloading
    - allows a particular symbol to have more than one meaning
  - other features
    - inheritance, virtual function, and templates
    - explained in later chapters

Let us briefly review c++

- How to define and use c++ classes
  - Class Person
    - Separate c++ compile in Linux
    - Makefile
  - Class String
    - Learn intermediate c++ concepts by examples

Person.h / Person.cpp

```
#include <iostream>

using namespace std;

class Person
{
    public:
        // data members
        char LastName[11], FirstName[11], Address[16];
        char City[16], State[3], ZipCode[10];

        // method
        Person();   // default constructor
};
```

```
#include "Person.h"

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;
}
```

PersonTest.cpp

- PesrsonTest.cpp

```
#include "Person.h"

using namespace std;

int main()
{
    cout<<"<<<<< Testing Person class >>>>>>>>"<<endl;

    Person p;
    Person *p_ptr = new Person;

    cout<<"Print Person class"<< endl;
    cout<<"FirstName: "<< p_ptr->FirstName;
    cout<<"LastName: "<< p_ptr->LastName;
    cout<<endl;
}
```

Compile C++ program in Linux

1. Assemble

```
$ g++ -o Person.o  –c Person.cpp
$ g++ –o PersonTest.o –c Person.cpp
```

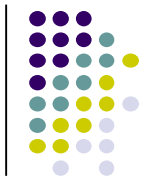
2. Link

```
$ g++ –o ptest Person.o PersonTest.o
```

3. Execute

```
$ ./ptest
```

Makefile for c++ program



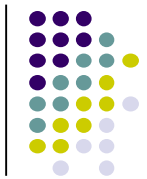
```
CFLAGS= -Wall
OBJS = Person.o PersonTest.o
all: ptest
%.o: %.cpp
    g++ -c -o $@ $(CFLAGS) $<
ptest: $(OBJS)
    g++ -o ptest $(OBJS)
clean:
    rm ptest $(OBJS)
```

String class definition

- At strclass.h
  - This class will be used at ch5.3, ch7 and ch8

```
class String
{
public:
    String();                                   // default constructor
    ~String();                                  // destructor
    String (const char*);                       // create from C String
    String (const String&);                     // copy constructor
    String (const char*);                       // create from C String
    String & operator = (const String& );       // assignment
    int operator ==  (const String& ) const;    // equality
    operator char*();                           //
    char * str() const;                         //  converstion to char*

    friend ostream & operator << (ostream& stream, String& str);

private:
    char * string;                              // represent value as C string
    int MaxLength;
};
```

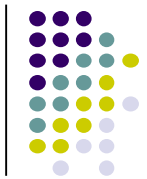
String class (1/4)



- Constructor/Destructor

```
#include "Strclass.h"

String::String()
{
    string =0;
    MaxLength = 0;
}

// create from C String
String::String(const char* str)
{
    MaxLength = strlen(str);
    string = new char[MaxLength];
    strcpy(string, str);
}

String::~String()
{
    if (string != 0)    delete string;
    MaxLength = 0;
    string = 0;
}
```

String class (2/4)



- Operator overloading

```
ostream & operator<< (ostream& stream, String &str);
inline ostream & operator<< (ostream& stream, String &str)
{
    stream << str.string;
    return stream;
}

int String::operator== (const String& str) const
{
    return strcmp (string, str.string) == 0;
}
```

String class (3/4)

- Copy constructor and assignment

```
// copy constructor
String::String (const String& str)
{
    string = strdup( str.string);
    MaxLength = strlen(string);
}

// assignment
String& String::operator= (const String & str)
{
    if (strlen (str.string) >= MaxLength)
    {
        delete string;
        string = strdup(str.string);
        MaxLength = strlen(string);
    }
    strcpy (string, str.string);
    return *this;
}
```

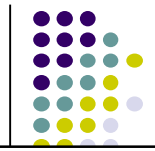
String class (4/4)

- Operator char* and char* str()
  - What is difference?

```
String::operator char*()
{
    return strdup(string);
}

// return a copy of the string
char* String::str() const
{
    return strdup(string);
}
```

Declares an overloaded cast operator function for creating a char*out of a class

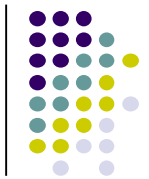
Test for String class



- strMain.cpp

```
int main()
{
    cout<<"=======String Class Test============"<<endl;
    String s1("abcdefg");    // Q1: uses which fuction?
    char str[10];
    strcpy (str, s1);       //Q2: uses which funciton?

    String s2;
    s2 = s1;        // Q3: uses which function?

    cout << "s1:\t"<< s1 << endl;
    cout << "s2:\t"<< s2 << endl;
    cout << "After setString" << endl;
    s1.setString("newStr");
    cout << "s1:\t"<< s1 << endl;
    cout << "s2:\t"<< s2 << endl;
    int c1=strcmp(s1, s2);     // Q4: why this code works?
    cout<<"comp: " << c1 << endl;

    String s3(s1);   // Q5: uses which function?
    String* s4 = new String;
    *s4 = s3;       // Q6: uses which function?

    cout << "s3:\t"<< s3 << endl;
    cout << "s4:\t"<< *s4 << endl;        // Q7: why use *s4?
    return 0;
}
```

Makefile



```
CFLAGS= -Wall
OBJS = Strclass.o StrMain.o
all: strTest
%.o: %.cpp
    g++ -c -o $@ $(CFLAGS) $<
strTest: $(OBJS)
    g++ -o strTest $(OBJS)
clean:
    rm strTest $(OBJS)
```

Getting source code from github

- A github link
  - https://classroom.github.com/a/9q5Hys8d
  - This link will provide source codes for the class

- Please see the file
  - 01.B.Git

Demo

- Compile and build

```
$ make
g++ -c -o Strclass.o -Wall -g Strclass.cpp
…
g++ -c -o StrMain.o -Wall -g StrMain.cpp
g++ -o strTest Strclass.o StrMain.o
```

- Execute

```
$ ./strTest
=======String Class Test===========
s1:     abcdefg
s2:     abcdefg
After setString
s1:     newStr
s2:     abcdefg
comp: 13
s3:     newStr
s4:     newStr
```

Answers (1/2)

- `String s1("abcdefg");  // Q1: uses which fuction?`
  - Constructor
- `strcpy (str, s1);  //Q2: uses which funciton?`
  - Operator char*()
- `s2 = s1;  // Q3: uses which function?`
  - Overloaded assignment
- `int c1=strcmp(s1, s2);  // Q4: why this code works?`
  - Operator char*()
- `String s3(s1);  // Q5: uses which function?`
  - Copy constructor

Answers (2/2)

- `*s4 = s3;` `// Q6: uses which function?`
  - Overloaded assignment

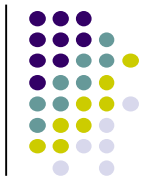
- `cout << "s4:\t"<< *s4 << endl;`
  `// Q7: why use *s4?`
  - Reference and pointer

Answers

- What we will need for the following code?

```
int main()
{

    cout<<"=======String Class Test==========="<<endl;

    String s1("Test string1");;
    String* s2 = new String;
    s2->setString("This is the second string");



    cout << s1 << endl;
    cout << *s2 << endl;

    return 0;
}
```

Conclusion



- 1.1 The heart of file structure design
- 1.2 History of file structure design
- 1.3 Conceptual toolkit : file structure literacy
- 1.4 Object-Oriented Toolkit
  : make file structure usable
- 1.5 Using objects in C++

Q&A

```
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
        operator char*();                           //
        char * str() const;                         //  converstion to char*

    private:
        char * string;                              // represent value as C string
        int MaxLength;

    friend ostream & operator << (ostream& stream, String& str);
};
```