

HiLCoE School of Computer Science & Technology
Computer Programming II
Chapters 2 and 3
Structure and File Management

Struct

- A (structured) data type
- A collection of data items, called fields, describing the same entity. Struct is mostly used to describe a non primitive entities describe using several attributes.

Struct type definition

Syntax

```
struct < idf > {  
    < list of field declarations >  
};
```

Description

idf is a (struct) type identifier. Note that <idf> is not a var-idf but rather a type-idf. This is the fourth declaration statement we have covered so far in this course (after constant idf, variable idf and function idf declaration statements).

Examples:

Eg1 struct course {
 char code[10];
 char title[50];
 int creditHr;
};

Eg2 struct date
{
 int day, month, year;
};

Eg3 struct country {
 char name[20];
 char capitalCity[20];
 char continentalLocation[15];
 long population;
 float gdp;
};

Eg4 struct book {
 char author[25];
 char title[40];
 char publisher[15];
 char year[5];
};

```

        char bookId[12];
        short nbPages;
    };

```

Eg5 struct member { // of a library
 char name[20]; // member name
 char mid[10]; // member id
 date dom; // date of membership; note that this is a
struct type
 char telephone[10]; // telephone number of the
member
 };

Struct variable declaration

Syntax

```
<struct type>            <list of idfs>;
```

Desc

The list of idfs are var-idfs of a struct type.

Examples: (Assuming the above type declarations)

Eg1 **book** x, y;
 x and y are variable idfs of the **book** type.
 How big is the memory space reserved for each variable?

Eg2 **country** yourc;
 yourc is a variable idf of the **country** type.
 How big is the memory space reserved for *yourc* variable?

Field Manipulation

As can be observed from the struct type declaration syntax, the fields can be of any type; it can even be another struct type (= hierarchical struct). These fields can be manipulated just like any other data elements that we have been manipulating up to now. In the struct case, we need only know how to access the individual fields constituting the struct.

Accessing fields

Syntax

```
<struct-var-idf> .<field>
```

Examples: (Assuming the above type and variable declarations)

Eg1
 strcpy(x.title, "Crime and punishment");
 strcpy(x.author, "Fyodor Dostoevsky");

```
strcpy(x.publisher, "Mega Publishing");
strcpy(x.year, "1998");
strcpy(x.bookId, "HiL/2458/99");
x.nbPages = 462;
```

Eg2

```
cout << "Enter the title of the book > ";
cin.getline(y.title, 39);
cout << "Enter the number of pages of the book > ";
cin >> y.nbPages;
```

Eg3 member mb;

```
strcpy(mb.name, "Wako Hadi");
```

Exercise: if the membership date of Wako is 14/8/96, how would you insert this in the variable *mb*?

Struct Manipulation

Field manipulation is not something new in relation to what we have been doing up to last chapter; struct manipulation, however, is quite new. What we can do with the whole struct variable is the question here.

1. Struct Constants: **Reading Assignment**

2. Assignment on struct

- It is possible to assign to a struct variable

Eg. (Assuming the struct variable x and y as above)

```
x = y; will assign values of y on a field by field basis to x
```

3. Struct in expressions

- No expression involving struct variables

4. Struct in I/O statements

- I/O statements work only on "Simple type" variables and strings; not on a struct type.

5. Struct and arrays

a. Arrays within struct is possible

Eg. Struct book {

```
    char author[5][20]; // multiple authors
```

```
    ....
```

```
    ....
```

```
};
```

b. Arrays of struct is also possible

Eg. book bk[50]; // bk is an array of books

6. Struct and pointer

7. Struct within a function

- a. Struct can be used as a common variable to communicating functions

- b. Struct as a return value is quite possible

Eg. `book readBook( )` // the struct book is the return type

```
{ book bk;
    cout << "Enter Author >";
    cin.getline(bk.author, sizeof(bk.author));
    cout << "Enter Title >";
    cin.getline(bk.title, sizeof(bk.title));
    ...
    ...
    return bk;
}
```

- c. Struct as a parameter is also possible. Refer to the parameter passing table discussed in class.

`void readBook(book * bk)` // struct type acts like simple types

```
{
    cout << "Enter Author >";
    cin.getline(bk->author, sizeof(bk->author));
    // or cin.getline((*bk).author,
sizeof((*bk).author);
    cout << "Enter Title >";
    cin.getline(bk->title, sizeof(bk->title));
    ...
    ...
}
```

Exercise: Provide the alias solution

File Management

File is a sequence of data elements stored in a secondary (= permanent) storage. Examples of permanent storage are floppy and hard disks, flash memory, CD and DVD. Unlike the data stored in the RAM, data stored in the secondary storage are saved for later use even after the computer is switched off. File Management deals with the creation, modification and closure of files.

Sequential and Random Access files

Sequential (access) file: A file is said to be sequential if the elements of the file can only be accessed sequentially one after another starting from the first element.

Random Access file: A file is said to be Random when it is possible to access any elements of the file directly (without necessarily passing through the “previous” elements).

Text and Binary files

Text file: A sequence of data elements considered as (ASCII) characters. In other words, the stream of bits constituting a text file are interpreted as a sequence of 8 bit character elements.

Binary file: A sequence of data elements of same type ($\neq$ char). In other words, the stream of bits constituting a binary file are interpreted as a sequence of, say, two byte “integers”, or a four byte “real” and so on depending on the type of the data elements of the file.

File Manipulation (in C++)

File manipulation (in C++) is a three step activity.

Step 1: File Opening

Step 2: File Processing

Step 3: File Closing

Step 1: File Opening

Opening a file means making the file ready for processing. A file must be opened before it can be read, write into or before any other operation can be made on it.

Syntax

```
<file-var-idf>.open(<filename>, <access mode>);
```

Desc

Open the file <filename>

Note:

- the file need not necessarily exist at the time of opening
- file-var-idf is a variable declared as one of the file stream (“types”) as discussed below
- access mode refers to one of the predefined access rights that may be attached to the file to be opened (see below)

File Streams (“types”)

- fstream: used for input and output
- ifstream : used for input only (Read Only)
- ofstream: used for output only (Write Only)

Access Modes

Access Mode	When the file exists	When the file does not exist	Remark
ios::in	Open for RO	Error	File Pointer (FP) is at <BOF>
ios::out	Open for WO; discard content	Create & Open for WO	<BOF> = <EOF>
ios::app	Open for WO	Create & Open for WO	FP at <EOF>; write (add) only at the end
ios::ate	Open for WO	Create & Open for WO	FP at <BOF>; write anywhere
ios::trunc	Discard content	Error	Used with app and ate
ios::nocreate		Error	Used with out , app and ate
ios::noreplace	Error		Used with out
ios::binary	Left as an exercise		

Note: Access modes may be combined using bit operators, like | (OR).

Examples:

```
fstream x;  
ifstream y;  
ofstream z;
```

```
x.open ("numb.dat", ios::in | ios::out);
```

```
y.open("student.bin", ios::in);
```

```
z.open("a:\\finance\\payrol\\october.rec", ios::out);
```

Step 2: File processing

File processing basically refers to file I/O (ie. writing into and reading from the file) just like writing and reading into/from a memory. However, there is one other set of activities related to what is called the File Pointer (see below).

File I/O

Text file I/O

Syntax for input
<file-var-idf> >> <var-idf>;

Description

Read a data element from a file designated by the <file-var-idf> and assign it to the <var-idf>.

Syntax for output
`<file-var-idf> << <exp>;`

Description
Write the evaluated value of the `<exp>` into a file designated by the `<file-var-idf>`.

Binary file I/O

Syntax for input
`<file-var-idf>.read ((char *) &<var-idf>, <size>;`

Description
Read a stream of bits of the given `<size>` from the designated file `<file-var-idf>` and assign it to the `<var-idf>`.

Syntax for output
`<file-var-idf>.write ((char *) &<var-idf>, <size>;`

Description
Write the stream of bits of the given `<size>` stored in the `<var-idf>` into the designated file `<file-var-idf>`.

File Pointer

It is a "pointer" indicating the next element of the file to be processed (ie. to read/write). File pointer plays a role almost similar to the cursor on the screen.

Operations on File Pointers

End of file

Syntax
`<file-var-idf>.eof( );`

Description
It returns 0 if the FP is not at `<EOF>`; a non-zero value otherwise.

Where is the FP ?

Syntax
`<file-var-idf>.tellg ( )`
`<file-var-idf>.tellp ( )`

Description
Returns the position of the file pointer in bytes.

Moving the FP

Syntax
`<file-var-idf>.seekg (<absolute posn>;` or `<file-var-idf>.seekg (<relative posn>, <reference>;`

`<file-var-idf>.seekp (<absolute posn>);` or `<file-var-idf>.seekp (<relative posn>, <reference>);`

Desc

Moves the file pointer to the designated position.

where

- `~seekg` is used when the file-var-idf is declared as `ifstream` or as `fstream`
- `~seekp` is used when the file-var-idf is declared as `ofstream` or as `fstream`
- `<absolute posn>` refers to the position (in bytes) where the FP should be placed counting from the first byte. In C++, the first byte is Byte # 0.
- `<relative posn>` is just like `<posn>` except that the reference now is indicated in the parameter `<reference>`
- `<reference>` are one of the following:
 - o `ios::beg` beginning of file
 - o `ios::cur` current position of the FP
 - o `ios::end` end of file

Step3: File Closing

Closing a file means making the file unavailable for processing. A file should be closed when it is no more needed for the mode of access it is opened for.

Syntax

`<file-var-idf>.close( );`

Examples

Example1: Text file input

```
int main( )
{
    int y;
    ifstream myf;
    myf.open("number.txt", ios::in);
    myf >> y;
    while (!myf.eof( ))
    {
        cout << y << endl;
        myf >> y;
    }
}
```

```

        myf.close( );
return 0;
}

```

Example2: Text file output

```

int main( )
{
    int n, i, y;
    ofstream df;
    df.open("numbers.dat", ios::out);
    cout << "Enter the number of data :> ";
    cin >> n;
    for (i=1; i<= n; i++)
    {
        cout << "Enter a number > ";
        cin >> y;
        df << y;
    }
    df.close( );
return 0;
}

```

Example3: Binary file input

```

int main( )
{
    int y;
    ifstream myf;
    myf.open("number.bin", ios::in);
    myf.read((char *) &y, sizeof(y));
    while (!myf.eof( ))
    { cout << y<< endl;
      myf.read((char *) &y, sizeof(y));
    }
    myf.close( );
return 0;
}

```

Example4: Binary file output

```

int main( )
{
    int n, i, y;
    ofstream df;
    df.open("numbers.bin", ios::out);
    cout << "Enter the number of data :> ";
    cin >> n;
    cout << " Enter the data one by one \n";
    for (i=1; i<= n; i++)
    {
        cout << "Enter a number > ";
        cin >> y;
    }
}

```

```

        df.write((char *)&y, sizeof(y));;
    }
    df.close( );
return 0;
}

```

Example5

File *course.txt* is a text file consisting of course code, title and credit hours separated by TAB as shown below

```
***** course.txt *****
```

```

CS221 Computer Programming I 5
CS302 Computer Organization 5
CS446 Object Oriented Software Engineering 4

```

```
/*
```

Program 1: a program that reads the above text file ("course.txt") and translates the content into a binary file of courses; the binary file is saved under the name "course.bin". A course is declared as a struct consisting of code, title and credit hours fields.

```
*/
```

```

#include<fstream>
#include<conio.h>

```

```

struct course {
    char code[7];
    char title[40];
    unsigned char creditHr;
};

```

```

int main()
{
    ifstream sourcef;
    ofstream targetf;
    course c;
    clrscr();
    sourcef.open("d:/tarex/cplus/course.txt", ios::in);
    targetf.open("d:/tarex/cplus/course.bin", ios::out);
    if (!sourcef.fail()){
        while (sourcef.peek() != EOF)
        {
            sourcef.getline(c.code, sizeof(c.code)-1, '\t');
            cout << c.code << endl;
            sourcef.ignore();
            sourcef.getline(c.title, sizeof(c.title)-1, '\t');
            cout << c.title << endl;
            sourcef >> c.creditHr;
            cout << c.creditHr << endl;
            sourcef.ignore();
            getch();
            targetf.write((char *) &c, sizeof(c));
        }
        sourcef.close();
    }
}

```

```

        targetf.close();
    }
    else cout << "Unable to open the file ";
    cout << "\n\n\t Binary file successfully created \n\n" <<
    endl;
    cout << " Press any key to terminate ... ";
    getch();
    return 0;
}

/*
Program 2: a program that reads the courses from the binary file
"course.bin" and dumps its content on the screen. A course is
declared as a struct consisting of code, title and credit hours fields.
*/

```

```

#include<fstream>
#include<conio.h>

struct course {
    char code[7];
    char title[40];
    unsigned char creditHr;
};

int main()
{
    ifstream sourcef;
    course c;
    clrscr();
    sourcef.open("d:/tarex/cplus/course.bin", ios::in);
    if (!sourcef.fail()){
        while (sourcef.peek() != EOF)
        {
            sourcef.read((char *) &c, sizeof(c));
            cout << c.code << endl;
            cout << c.title << endl;
            cout << c.creditHr << endl;
        }
        sourcef.close();
    }
    else cout << "Unable to open the file ";
    getch();
    return 0;
}

```

Exercise

Assuming that you have a binary file of **course** (as defined above), write a menu driven program that lists, modifies, inserts and deletes courses into/from the binary file.

Example6

```

/* This program reads a text file (in this case a C++ program) and then
incorporates line numbers
*/

```

```

#include<fstream>
#include<conio.h>
#include<cstring>

int main()
{
    const MAXLINELENGTH = 100; // this is just an assumption
    ifstream sourcef;
    ofstream targetf;
    int i = 0;
    char x[100];
    sourcef.open("d:/tarex/cplus/average.cpp", ios::in);
    targetf.open("d:/tarex/cplus/averagel.cpp", ios::out);
    if (!sourcef.fail()){
        while (sourcef.peek() != EOF)
        {
            i++;
            sourcef.getline(x, MAXLINELENGTH);
            cout << i << ": " << x << endl;
            targetf << i << ": " << x << endl;
        }
        sourcef.close();
        cout << "\n\n\t Number of lines = " << i;
    }
    else cout << "Unable to open the file ";
    fin:
    getch();
    return 0;
}

```

Exercise

Modify the program so that

- it accepts the file name from the user
- it tells whether the source file (assumed to be a C++ program) has a balanced curly brackets or not
- counts the C++ statements rather than just physical lines (!!!)

Example7: Files in modular programming

```

/* This program lists the contents of a binary file consisting of
books; it also allows to add new books.
*/

```

```

#include<fstream>
#include<conio.h>

struct book {
    char bid[10];
    char author[20];
    char title[40];
};

void addNewBooks(fstream & outf)
{
    short nbooks;
    book bk;
    outf.open("d:/tarex/cplus/books.bin", ios::app);
}

```

```

        cout << "How many new books you want to add > ";
        cin >> nbooks;
        cin.ignore();
        for (int i=1; i<=nbooks; i++)
        {
            cout << "\n\n\t\t\t Book # " << i << "\n\n";
            cout << "\t Enter Book Id > ";
            cin.getline(bk.bid, sizeof(bk.bid)-1, '\n');
            cout << "\t Enter Author > ";
            cin.getline(bk.author, sizeof(bk.author)-1, '\n');
            cout << "\t Enter title > ";
            cin.getline(bk.title, sizeof(bk.title)-1, '\n');
            outf.write((char *) &bk, sizeof(bk));
        }
        outf.close();
    }

void listAllBooks(fstream & inf)
{
    short nbooks;
    book bk;
    inf.open("d:/tarex/cplus/books.bin", ios::in);
    inf.seekg(0); // File pointer at beginning
    while (inf.peek() != EOF)
    {
        inf.read((char *)&bk, sizeof(bk));
        cout << bk.bid << '\t';
        cout << bk.author << '\t';
        cout << bk.title << endl;
    }
    inf.close();
}

int main()
{
    fstream bookfile;
    clrscr();
    listAllBooks(bookfile);
    addNewBooks(bookfile);
    listAllBooks(bookfile);
    getch();
    return 0;
}

```