

Part 1: Sample Solution (Phase 2 + check validity of file entries)

activity.h

```

#ifndef ACTIVITY_H
#define ACTIVITY_H

#include <iostream>
using namespace std;

//The Activity Class
class Activity {
public:
    //Constructors
    Activity();
    Activity(int h, int m, int mDur, const char *des);

    // I/O
    friend ostream &operator <<(ostream &outs, const Activity &activity);
    friend istream &operator >>(istream &ins, Activity &activity);

    //Comparison
    friend bool operator >(const Activity &activity1, const Activity &activity2);

    //Searching
    friend bool binary_search(const Activity activities[], int n, int search_hh, int search_mm, int &idFound);

    //Check whether the activity object contains valid values
    bool valid() const;

private:
    //Data members
    int hr, min, minuteDur;
    char description[26];
};

//Sort an array of activities using insertion sort
void insertion_sort(Activity activities[], int n);

//Read activities into an array
void readFileOfActivities(Activity activities[], int &n);

#endif

```

Common mistake:

If a non-member function doesn't use or access private members directly, it should not be a friend.
(eg. readFile, insertion_sort, swap)

Important concerns:

- When should we declare a member function?
 - When should we declare a non-member function?
 - When should we declare a non-member function as friend?
- Read Lec09.pdf (Pages 2) for the above!

activity.cpp

```

#include "activity.h"
#include <iomanip>
#include <fstream>

//Default constructor
Activity::Activity()
{
    hr = 0;
    min = 0;
    minuteDur = 0;
    description[0] = '\0';
}

//Constructor with initialization values
Activity::Activity(int h, int m, int mDur, const char *des)
{
    int iDest, i;
    hr = h;
    min = m;
    minuteDur = mDur;

    for (i=0, iDest=0; des[i] != '\0'; i++, iDest++)
        description[iDest] = des[i];
    description[iDest] = '\0';
}

```

Common mistake:

The aim of a constructor is to initialize **all** data members. You should make sure that whenever you write a constructor, it should achieve this purpose. It is wrong if you leave one or more data member with undefined ("by-luck") value.

```

//Read activities into an array
void readFileOfActivities(Activity activities[], int &n)
{
    ifstream inFile;
    char pathname[200];

    cout << "Type the pathname of the input file: ";
    cin >> pathname;

    inFile.open(pathname);
    if (inFile.fail())
    {
        cout << "Can't open input file.\n";
        exit(1);
    }

    n = 0;
    while (!inFile.eof())
    {
        Activity anActivity;
        inFile >> anActivity;
        if (!inFile.fail())
        {
            if (!anActivity.valid()) //check times
            {
                cout << "Wrong entry: " << anActivity << endl;
            }
            else
            {
                activities[n] = anActivity;
                n++;
            }
        }
    }

    inFile.close();
}

```

An important concern:

Though the file is not ended yet, but `>>` may still fail (because there may be blank line(s) only), special measure has to be taken to check whether `>>` has been executed successfully (**not failed**).

Another shorter version (Correct approach 2):

```

Activity anActivity ;
while (inFile>>anActivity) // do >> and check whether it is sucessfully.
{
    if (!anActivity.valid()) //check times
    ..
}

```

```

//Check whether the activity object contains valid values
bool Activity::valid() const
{
    if (hr<0 || hr>=24)
        return false;
    if (min<0 || min>=60)
        return false;
    if (minuteDur < 0)
        return false;
    return true;
}

```

A common procedure of setting up an object:

Instead of getting values into an object's data members directly, in this sample code it firstly sets up some local variables, and then sets up the object using a constructor according to the values of these local variables. Very often this 2-step procedure allows us apply data checking with higher flexibility and handle data integrity easier.

```

//Overloaded operator >>
istream &operator >>(istream &ins, Activity &activity)
{
    char dummy;
    int h, m, mDur;
    char s[26];
    ins >> h >> dummy >> m >> mDur;
    ins.get(dummy); //the tab before description
    ins.getline(s, 26);
    activity = Activity(h, m, mDur, s);
    return ins;
}

//Overloaded operator >
bool operator >(const Activity &activity1, const Activity &activity2)
{
    if (activity1.hr > activity2.hr)
        return true;
    if (activity1.hr < activity2.hr)
        return false;
    return activity1.min > activity2.min;
}

```

```
//swap 2 objects
void swap(Activity &a1, Activity &a2)
{
    Activity old_a1 = a1;
    a1 = a2;
    a2 = old_a1;
}
```

Swap function

- **3 lines only - Using = to copy the whole object!!**

Answer: Yes! We can use = in this way (all data members are copied).
See Lec08.pdf (page 9) and Lec11.pdf (page 4 – example 2)

- **What if you don't write this swap function?**

Answer: When the compiler translates the insertion_sort function (below), it needs the swap function. If an appropriate swap function is not available, the compiler will generate an appropriate one based on a given *swap template* (given in standard C++ library).

Small experiment using the debugger: When it runs insertion_sort and calls swap(), use F11 to see where you will be brought to!!

- ❌ **Don't write swap as a friend function**

"Not need to be a friend" because it doesn't access any private member of the class

- ❌ **Don't write swap as a member function**

Ordinary functions (say insertion_sort) will use it to simply swap 2 parameter objects. It is not to handle any any "calling object" (ie. not like "ob.swap(obj1,obj2)"), so the swap function should not be a member function.

- ❌ **Don't write void swap(int &a1, int &a2) ← "int" is WRONG**

```
/* insertion sort -- in ascending order */
void insertion_sort(Activity activities[], int n)
{
    int i;

    for (i = 1; i < n; i++) // round i
    {
        int j = i;

        while (j>0 && activities[j-1]>activities[j]) //this loop runs as long as a[j] needs to swap with a[j-1]
        {
            swap(activities[j-1], activities[j]);
            j--;
        }
    }
}
```

```
//Binary search
bool binary_search(const Activity activities[], int n, int search_hh, int search_mm, int &idFound)
{
    int lower=0, upper=n-1;

    while (lower <= upper)
    {
        int m = (lower+upper)/2;

        if (activities[m].hr==search_hh &&
            activities[m].min==search_mm)
        {
            idFound = m;
            return true; // found and stop!!
        }

        if (activities[m].hr<search_hh ||
            (activities[m].hr==search_hh && activities[m].min==search_mm))
            Lower = m+1;

        else
            upper = m-1;
    }

    return false; //can't find x
}
```

❌ The following is **NOT binary search** (It is "sequential search" – not the required version in this assignment)

```
for (i=0;..i++)
    if (.. == ..)
    {
        id=i;
        found=true;
    }
```

```
//Overloaded operator <<
ostream &operator <<(ostream &outs, const Activity &activity)
```

```

{
    int dueMin = (activity.min+activity.minuteDur) % 60;
    int dueHr = (activity.hr + (activity.min+activity.minuteDur)/60) % 24;
    int durMin = activity.minuteDur % 60;
    int durHr = activity.minuteDur / 60;

    outs << setfill('0');
    outs << setw(2) << activity.hr << ":"<< setw(2) << activity.min << " - "
        << setw(2) << dueHr << ":"<< setw(2) << dueMin << " ";

    outs << activity.description << " (";

    if (durHr == 1)
        outs << "1 hour";
    else if (durHr > 1)
        outs << durHr << " hours";

    if (durHr && durMin)
        outs << " ";

    if (durMin == 1)
        outs << "1 minute";
    else if (durMin > 1)
        outs << durMin << " minutes";
    outs << ")";
    return outs;
}

```

General practices

- (1) You may give **comments** in **both *.h and *.cpp** (in front of all ordinary and member functions, global values and constants)
- (2) Give proper indentations (tabs), spacing, and naming
- (3) Avoid hardcoding (Improve your program by using named constants where appropriate)
- (4) Use call-by-reference or return? Answer: **return** is our **first choice**
- (5) **return**
 - Mind all compilation warnings: "not all control paths retn a value"
 - If a function has a return value, the returned value should be an appropriate result that is really useful to the caller.
- (6) Mind all compilation warnings, eg. "unreferenced local variable", "uninitialized variable is used"
- (7) Unless really necessary, **do not** declare an ordinary function (ie. non-member function) as **friend**.
 Example: swap, readFileOfActivities, and insertion_sort do not need to be friends of the class. They don't access private members directly.

- (8) Adopt a member function where appropriate.

For example, it is proper to design

`bool Activity::valid() const;` ← a better version
 rather than `bool valid(const Activity &r);` ← not so good

Reason: The above function is to apply on one object only.

It is natural to call it like `activity.valid()` and `activities[i].valid()`

- (9) Apply **const** to **member functions** where appropriate (ie. in case the calling object is not changed). Example:

```

class Activity {
    ..
    bool valid() const;
    ..
};

```

- (10) Apply **const** to **parameters** where appropriate (**arrays and objects**). (Review last page of Lec09.pdf)

Example:

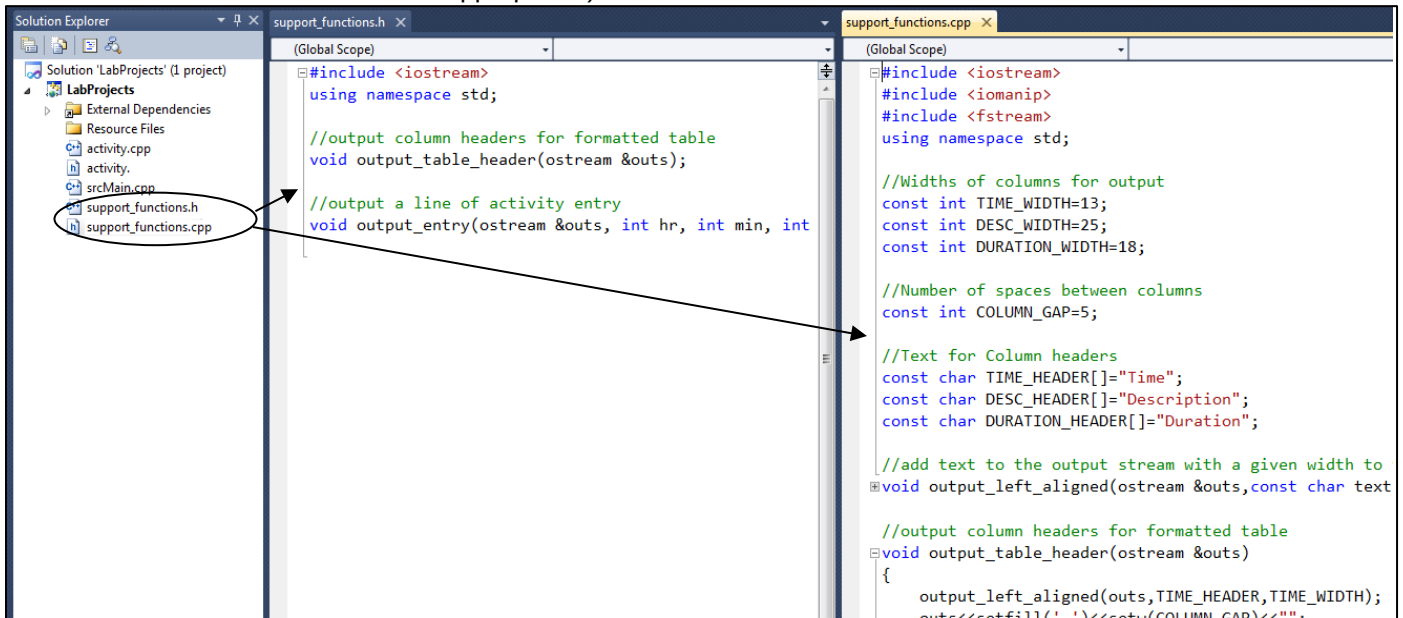
```

class Activity {
    ..
    Activity(int h, int m, int mDur, const char *des);
    ..
    friend bool operator >(const Activity &a1, const Activity &a2);
    friend bool binary_search(const Activity activities[], int n, int search_hh, int search_mm, int &idFound);
};

```

Part 2: Extensions for “Creative Design” -- Formatted columns and Different sort keys

- The Activity.cpp and h should contain core members and friends of the class only.
 - "Application specific" functions should be implemented elsewhere (not within this class).
 - Example: we may think that the code that *outputs formatted columns (user interface design) for this assignment* will probably be irrelevant when we want to re-use this Activity class to write other programs later.
 - Friends can be converted to non-friend easily if we provide additional access functions.
 - If you've designed some friend functions that *are specific to the application* (eg. this assignment's user interface) or the friend functions do not really serve solely for the Activity objects, then it is better to change them to non-friend and move them elsewhere (not in activity.cpp and .h)
- In the implementation below, support_functions.cpp and .h are added. (Currently only two functions are kept inside. But we can add more whenever appropriate.)



New run-down:

[A] Input activities from file
=====

Type the pathname of the input file: c:\t1.txt

7 activities are extracted.

[B] Display list of activities
=====

Time	Description	Duration
09:00 - 09:30	Go to laundry shop	30 minutes
15:00 - 16:00	Swim	1 hour
20:00 - 21:30	Movie	1 hour 30 minutes
23:30 - 00:15	Long-distance call	45 minutes
08:20 - 08:50	Breakfast with Maggie	30 minutes
12:00 - 12:50	Lunch	50 minutes
19:00 - 19:45	Dinner at home	45 minutes

[C] Sort activities
=====

By [S]Start time, [T]Duration or [D]Description? Your choice: D

Time	Description	Duration
08:20 - 08:50	Breakfast with Maggie	30 minutes
19:00 - 19:45	Dinner at home	45 minutes
09:00 - 09:30	Go to laundry shop	30 minutes
23:30 - 00:15	Long-distance call	45 minutes
12:00 - 12:50	Lunch	50 minutes
20:00 - 21:30	Movie	1 hour 30 minutes
15:00 - 16:00	Swim	1 hour

[D] Search activities
=====

Type a start time to search: 20:00

20:00 - 21:30	Movie	1 hour 30 minutes
---------------	-------	-------------------

Thank you for using this program.
Press any key to continue . . .

Sample Output

- The main function:

```
#include "activity.h"
#include "support_functions.h"

#include <iostream>
using namespace std;

int main()
{
    //The activities array and count of activities (n)
    Activity activities[20];
    int n;

    char sort_choice; // choice of sorting

    //For part D
    int id_found;
    int search_hh, search_mm;
    char dummy; //colon between search_hh and search_mm

    cout << "[A] Input activities from file\n";
    cout << "=====\n";
    readFileOfActivities(activities, n);
    cout << endl;
    cout << n << " activities are extracted.\n\n";

    cout << "[B] Display list of activities\n";
    cout << "=====\n";
    output_table_header(cout);
    for (int i=0; i<n; i++)
        cout << activities[i] << endl;
    cout << endl << endl;

    cout << "[C] Sort activities\n";
    cout << "=====\n";
    cout << "By [S]Start time, [T]Duration or [D]Description? Your choice: ";
    cin >> sort_choice;
    setSortMode(sort_choice);
    insertion_sort(activities,n);

    output_table_header(cout);
    for (int i=0; i<n; i++)
        cout << activities[i] << endl;
    cout << endl << endl;

    cout << "[D] Search activities\n";
    cout << "=====\n";
    cout << "Type a start time to search: ";
    cin >> search_hh >> dummy >> search_mm;

    if (getSortMode()!='S')
    {
        setSortMode('S');
        insertion_sort(activities,n); // <==== sorting before search
    }

    if (binary_search(activities, n, search_hh, search_mm, id_found))
    {
        cout << "\n";
        cout << activities[id_found] << "\n\n";
    }
    else
        cout << "\nNo activity is found.\n\n";

    cout << "Thank you for using this program.\n\n";

    return 0;
}
```

- Modified activity.h and cpp, and added support_functions.h and cpp: see next page

New/Updated code in activity.h

```
..  
//The Activity Class  
class Activity {  
public:  
  
};  
..  
  
// Set/get sort mode ('S' for start time, 'T' for  
void setSortMode(char mode);  
char getSortMode();  
#endif
```

New functions added

These functions can be called by outsiders (eg. main()) to control the value of Sort_mode (a global variable defined in activity.cpp)
Then the `>` operator will return true/false according to Sort_mode. (The original insertion_sort function can still be used without any change.)

New/Updated code in activity.cpp

```
#include "activity.h"  
#include "support_functions.h"  
#include <fstream>  
  
char Sort_mode; //Mode of sorting ('S' for start time, 'T' for duration, 'D' for description)  
  
//Change the mode of sorting (see the comments for the global variable Sort_mode)  
void setSortMode(char mode)  
{  
    Sort_mode = mode;  
}  
  
//Return the mode of sorting (see the comments for the global variable Sort_mode)  
char getSortMode()  
{  
    return Sort_mode;  
}  
  
bool operator >(const Activity &activity1, const Activity &activity2)  
{  
    if (Sort_mode == 'S')  
    {  
        if (activity1.hr > activity2.hr)  
            return true;  
        if (activity1.hr < activity2.hr)  
            return false;  
        return activity1.min > activity2.min;  
    }  
    else if (Sort_mode == 'T')  
        return activity1.minuteDur > activity2.minuteDur;  
    else // Sort_mode is 'D'  
        return (strcmp(activity1.description, activity2.description) == 1);  
}
```

```
//Overloaded operator <<  
ostream &operator <<(ostream &outs, const Activity &act)  
{  
    output_entry(outs, act.hr, act.min, act.minuteDur, act.description);  
  
    return outs;  
}
```

`output_entry` is defined in support_functions.cpp

support_functions.h

```
#include <iostream>
using namespace std;

//output column headers for formatted table
void output_table_header(ostream &outs);

//output a line of activity entry
void output_entry(ostream &outs, int hr, int min, int minuteDur, const char *description);
```

support_functions.cpp

```
#include <iostream>
#include <iomanip>
#include <fstream>
using namespace std;

//Widths of columns for output
const int TIME_WIDTH = 13;
const int DESC_WIDTH = 25;
const int DURATION_WIDTH = 18;

//Number of spaces between columns
const int COLUMN_GAP = 5;

//Text for Column headers
const char TIME_HEADER[] = "Time";
const char DESC_HEADER[] = "Description";
const char DURATION_HEADER[] = "Duration";

//add text to the output stream with a given width to fill up
void output_left_aligned(ostream &outs, const char text[], int n)
{
    int len;
    outs << text;
    for (len=0; text[len]!='\0'; len++)
        ;
    outs << setfill(' ') << setw(n-len) << "";
}

//output column headers for formatted table
void output_table_header(ostream &outs)
{
    output_left_aligned(outs, TIME_HEADER, TIME_WIDTH);
    outs << setfill(' ') << setw(COLUMN_GAP) << "";
    output_left_aligned(outs, DESC_HEADER, DESC_WIDTH);
    outs << setfill(' ') << setw(COLUMN_GAP) << "";
    output_left_aligned(outs, DURATION_HEADER, DURATION_WIDTH);
    outs << endl;

    outs << setfill('-') << setw(TIME_WIDTH) << "";
    outs << setfill(' ') << setw(COLUMN_GAP) << "";
    outs << setfill('-') << setw(DESC_WIDTH) << "";
    outs << setfill(' ') << setw(COLUMN_GAP) << "";
    outs << setfill('-') << setw(DURATION_WIDTH) << "";
    outs << endl;
}

//output a line of activity entry
void output_entry(ostream &outs, int hr, int min, int minuteDur, const char *description)
{
    int dueMin = (min+minuteDur) % 60;
    int dueHr = (hr+(min+minuteDur)/60) % 24;
    int durMin = minuteDur % 60;
    int durHr = minuteDur / 60;

    outs << setfill('0');
    outs << setw(2) << hr << ":" << setw(2) << min << " - " << setw(2) << dueHr << ":" << setw(2) << dueMin;

    outs << setfill(' ') << setw(COLUMN_GAP) << "";
    output_left_aligned(outs, description, DESC_WIDTH);
    outs << setfill(' ') << setw(COLUMN_GAP) << "";

    if (durHr == 1)
        outs << "1 hour";
    else if (durHr > 1)
        outs << durHr << " hours";

    if (durHr && durMin)
        outs << " ";

    outs << durMin << " minute";
    if (durMin > 1)
        outs << "s";
}
```

Not listed in .h

As this function is not used by outsiders (like main()), so it can be "hidden" in .cpp and is not listed in .h.

Call this function