Bridge and torch problem

Four people come to a river in the night. There is a narrow bridge, but it can only hold two people at a time. They have one torch and, because it's night, the torch has to be used when crossing the bridge. Person A can cross the bridge in 1 minute, B in 2 minutes, C in 5 minutes, and D in 10 minutes. When two people cross the bridge together, they must move at the slower person's pace.

Write a program in C++ that determines the fastest time they can cross the bridge. The entry to the program will be a yaml file that describes the speeds of each Person. In this situation there are 4 people but your program should assume any number of people can be passed in.

Analysis 1 (Team of four)

We start with the first part of the problem: Person A can cross the bridge in 1 minute, B in 2 minutes, C in 5 minutes, and D in 10 minutes.

The fastest solution to the puzzle is as follows: (The fastest time is 17 minutes)

Step	Left side	Cross bridge	Right side	Time
1	**A, B, C, D**			
2	**C, D**	**= > A, B**	**A, B**	**2 (minutes)**
3	**A, C, D**	**< = A**	**B**	**1**
4	**A**	**= > C, D**	**B, C, D**	**10**
5	**A, B**	**< = B**	**C, D**	**2**
6		**= > A, B**	**A, B, C, D**	**2**
				Total:17 minutes

The algorithm used here seems straightforward: have people with similar speeds cross the bridge together, so that the faster of the pair “wastes” as little time as possible. In this case, the two fastest people walk together and the two slowest people walk together. Furthermore, we use the fastest available person to move the torch back across the bridge.

To generalize, we use two lists to represent the team members [A, B, C, D] and their times to cross bridge [a, b, c, d] respectively. The table above can be redrawn as follows.

Step	Left side	Cross bridge	Right side	Time
1	A, B, C, D			
2	C, D	= > A, B	A, B	b
3	A, C, D	< = A	B	a
4	A	= > C, D	B, C, D	d
5	A, B	< = B	C, D	b
6		= > A, B	A, B, C, D	b
				Total: a+3b+d

Table 1

However, there are many other options to move all four people to the other side of bridge. For a four-person team, is a+3b+d always the fastest time? Not necessarily. The fastest time also depends on the time it takes for each person to cross the bridge. For example, for a team with time list [1, 4, 5, 10], the fastest time comes from using the following strategy:

Step	Left side	Cross bridge	Right side	Time
1	A, B, C, D			
2	B, C	= > A, D	A, D	10 (d)
3	A, B, C	< = A	D	1 (a)
4	B	= > A, C	A, C, D	5 (c)
5	A, B	< = A	C, D	1 (a)
6		= > A, B	A, B, C, D	4 (b)
				Total: 21(2a+b+c+d)

Table 2

Among all possible options, one of the two strategies listed above will be the fastest possible strategy for a four-person puzzle. But, how do you determine which one to use given a list of crossing times?

Let's compare the time formula for the two: a+3b+d, 2a+c+d. For a time list [a, b, c, d] we will pick the formula which returns the smaller number from the formula: a+b+d + min(2b, a+c). In other words, if 2b>a+c we should use the algorithm in Table 2. Otherwise, we should use the one in Table 1.

Analysis 2 (Team of n>0)

We now have a problem represented by two lists [A, B, X1, X2, X3, …, C, D], [a, b, x1, x2, x3,…, c, d] respectively. The second list is in order. The fastest two people are A and B who take time a and b to cross the bridge, respectively. The slowest two people are C and D who take time c and d to cross the bridge, respectively.

In our algorithm,

(a) We focus on the four people at two ends of the list, A, B, C and D. We will use our criteria, 2b>a+c, to determine which of the two strategies detailed in table 1 or table 2 we will use to move C and D to the other side of bridge. As we determined in Analysis 1, one of these two strategies will result in the fastest time to move C and D across the bridge with the additional criteria that the torch be on the starting side of the bridge.

(b) Repeat the action in (a), to move the next two slowest people across the bridge, until there are four or fewer people left on the starting side of the bridge. This process is illustrated in the following table.

Left side of bridge	**Use Table 1 or 2 (steps 1 to 5) to move the slowest two people across**	**Right side of bridge**
A,B,X1,X2,X3,…,C1,D1,C,D		
A,B,X1,X2,X3,…,C2,D2,C1,D1	**A, B, C, D**	**C, D**
A,B,X1,X2,X3,…,C3,D3,C2,D2,	**A, B, C1, D1**	**C1, D1, C, D**
Continue until there are 4 or fewer people left		

Table 3

(c) If there are four people left, follow step (a), but finish the problem with step 6 of whichever strategy you select. If there are three people left, it will always be the fastest strategy for the fastest and slowest of the remaining three people to cross, the fastest to return, and then the final two people to cross.

Architecture

(1) The solver class SolverPuzzle is listed below:

```
class SolvePuzzle
{
public:
	SolvePuzzle();
	pair_vec ReadYAMLFile(string filepath);
	int Solve(string filePath);
	int Solve2(string filePath);
	~SolvePuzzle();

	int GetMinTime() { return m_MinTime; }
	void DisplayMembers() { pCB->DisplayMembers(); }
	void DisplayActions() { pCB->DisplayActions(); }
private:
	CrossBridgeBase *pCB;
	CrossBridgeBase *pCB2;
	int m_MinTime;
};
```

There are three parts in the class.

(a) Exposed functions for input data. The function `ReadYAMLFile` is used to read input data and store data in a map (I didn't find a good YAML file parser. This is just some simple code to read mapping data in a YAML file. It can be re-implemented once a YAML file parser dll is added to the solution.)

At the same time, the data in the map is filled to a `vector<pair<string, int>>` which is used for the construction of core solver engine(s). In the vector, the data is sorted based on the time required to cross the bridge. We need sorted data based on the algorithm described in table 3 of Analysis 2. In addition, in the vector, the type of element is `pair<string, int>`. In a STL container, such as a vector, a lot of element copying is performed during calculation. If an element's size is too big, performance could be an issue. In that case we may choose a pointer (or a smart pointer) to an object as the element of the vector. In the current problem, the size of `pair<string, int>` isn't too big, so we still store objects directly in the vector.

(b) The core problem solver engines are `CrossBridgeN` and `CrossBridgeN2` which are derived from a base pure abstract class `CrossBridgeBase`. Only pointers of `CrossBridgeBase` are members in `SolvePuzzle`. The implementation of the engines is hidden.

(c) Functions for the output of results are listed below:

```
int GetMinTime();
void DisplayMembers();
void DisplayActions();
```

(2) The first core solver engine `CrossBridgeN` is derived from the pure abstract class:

```
class CrossBridgeBase
{
public:
	CrossBridgeBase();
	virtual int TimetoTake() = 0;
	virtual ~CrossBridgeBase();
	void DisplayMembers();
	void DisplayActions();

protected:
	void RecordActions(bool direction, pair<string, int> p1, pair<string, int> p2);
	void RecordActions(bool direction, pair<string, int> p1);
	pair_vec m_mem_vec;
	int m_n;
	vector<string> m_act_vec;
};

class CrossBridgeN :
	public CrossBridgeBase
{
public:
	CrossBridgeN(pair_vec &vec, int n);
	int TimetoTake() override;
	~CrossBridgeN();
};
```

(a) We consider the base class `CrossBridgeBase` as a blueprint. If needed, it is ready to serve as a dynamic polymorphism for the program (used in the second solver engine). As long as we keep the protocol of functions, when we change any internal implementation or implement a new solver engine later, we will not have to change much to the out layer classes.

(b) The function `TimetoTake` is implemented based on the earlier discussion in Analysis 1 and Analysis 2.

(c) With the following functions we can output the result minTime, people and the corresponding time for bridge crossing and the specific steps to achieve the minTime.

```
CrossBridgeN::TimetoTake()

CrossBridgeN::DisplayMembers()

CrossBridgeN::DisplayActions()
```

(3) The second core solver engine `CrossBridgeN2` is derived from the pure abstract class `CrossBridgeBase` as well.

```
class CrossBridgeN2 :
	public CrossBridgeBase
{
public:
	CrossBridgeN2(pair_vec &vec, int n);
	int TimetoTake() override;
	~CrossBridgeN2() = default;
};
```

(a) Like the first solver, this solver uses the same algorithm in the implementation. Technically the first solver is using a recursive function to loop through, whereas a while loop is used in the second solver with polymorphism.

(b) The class `CrossBridgeN2` uses `CrossBridge1`, `CrossBridge2`, `CrossBridge3` and `CrossBridge4` which are all subclasses of `CrossBridgeBase`. The class `CrossBridge1` is used to deal with a one-person team. The class `CrossBridge2` is for a team of two, etc. The class `CrossBridge4` is special. It can perform all six steps in Tabel 1 or Table 2 or only the first five steps. A function `SetFiveSteps` controls this. The first 5 steps only move the two slowest people to the other side of bridge given a team of four people. The purpose is discussed in Table 3.

Run the program

The main () function of the program is in the file CrossBridge.cpp. Edit the first line for the location of your input file.

The YAML file contains a mapping. Here is a sample:

person:

person_1: 1

person_2: 2

person_3: 5

person_4: 10

The file name and path would also be in the YAML file or a separate configuration file.

Thank you for reading my work.