Design

• How to:
  • break a problem down
  • refine it
Goal – do it right
Want
Input 2 times in the format of

\[ \text{hh:mm:ss} \]

Output the sum of the above 2 times

Continue until 0 is entered for the 1\text{st} time
**Time Machine**

- Input 2 times in the format of **hh:mm:ss**
- Output the sum of the above 2 times
- Continue until 0 is entered for the 1\(^{st}\) time

**Sample run:**

Enter the 1st time: 1:10:15
Enter the 2nd time: 2:10:20
The total time is: 3:20:35
Time Machine

- Input 2 times in the format of `hh:mm:ss`
- Output the sum of the above 2 times
- Continue until 0 is entered for the 1st time

Need to
- get 1st time
- get 2nd time
- add them and print the sum
- continue until a 0 is entered for 1st time
Input 2 times in the format of hh:mm:ss
Output the sum of the above 2 times
Continue until 0 is entered for the 1\textsuperscript{st} time

Need to
\begin{itemize}
  \item get 1\textsuperscript{st} time
  \item get 2\textsuperscript{nd} time
  \item add them and print the sum
  \item continue until a 0 is entered for 1\textsuperscript{st} time
\end{itemize}
Design: input 2 times

- int getData (string prompt)
  - call it twice
  - convert time down to an int
Input 2 times in the format of \texttt{hh:mm:ss}
\textbf{Output the sum} of the above 2 times
Continue until 0 is entered for the 1\textsuperscript{st} time

Need to
\begin{itemize}
\item get 1\textsuperscript{st} time
\item get 2\textsuperscript{nd} time
\item add them and print the sum
\item continue until a 0 is entered for 1\textsuperscript{st} time
\end{itemize}
Design: Output the sum

- void printSum (int time1, int time2)
  - add times together
  - split back apart for the format
  - print
- Input 2 times in the format of `hh:mm:ss`
- Output the sum of the above 2 times
- **Continue until 0 is entered for the 1st time**

- **Need to**
  - get 1st time
  - get 2nd time
  - add them and print the sum
  - continue until a 0 is entered for 1st time
Loop: continue until 0 is entered

While (time1 != 0)
- int getData (string prompt)
  - call it twice
  - convert time down to number of seconds

```cpp
//format  hh:mm:ss
int getData(string prompt)
{
    cout << prompt;
    int hour, min, sec;
    char junk;
    cin >> hour >> junk >> min >> junk >> sec;
    return (hour * 3600 + min * 60 + sec);
}
```
void printSum (int time1, int time2)  
//add times together  
//split back apart for the format  
//print  
{
    int sum = time1 + time2;
    int hr = sum / 3600;
    int leftOverSec = sum % 3600;
    int min = leftOverSec / 60;
    int sec = leftOverSec % 60;
    cout << hr << ‘:’ << min << ‘:’ << sec;
}
Design: continue until 0 is entered

- loop

```c
int time1 = getData("Enter the 1st time: ");
while (time1 != 0)
{
    int time2 = getData("Enter the 2nd time: ");
    printSum(time1, time2);
    time1 = getData("enter the 1st time: ");
}
```
One step at a time

- Start small
- One small step
- Build
- Test
- Add
The Drunkard's Walk

- A drunkard in a grid of streets randomly picks 1 of 4 directions and stumbles to the next intersection.
- then again, randomly picks 1 of 4 directions, ...
- Represent locations as integer pairs \((x,y)\)
- Implement the drunkard's walk over 100 intersections
- Print the beginning and ending location
Break into major 'jobs' that need to be done

- decide initial intersection
Break into major 'jobs' that need to be done

- decide initial intersection \((x, y) = (0, 0)\)
Break into major 'jobs' that need to be done

- decide initial intersection \((x,y) = (0,0)\)
- random pick of direction
Break into major 'jobs' that need to be done

- decide initial intersection \((x, y) = (0, 0)\)
- random pick of direction
  - \texttt{direction = randomPick();}
Break into major 'jobs' that need to be done

- decide initial intersection \((x, y) = (0, 0)\)
- random pick of direction
  - direction = randomPick();
- change intersections
Break into major 'jobs' that need to be done

- decide initial intersection \((x,y) = (0,0)\)
- random pick of direction
  - direction = randomPick();
- change intersections by applying direction
  - applyDirection(x, y, direction);

...
Break into major 'jobs' that need to be done

- decide initial intersection \((x, y) = (0, 0)\)

- run 100 times
  
  ```cpp
  for (int i = 0; i < 100; i++)
  {
    // random pick of direction
    direction = randomPick();
    // change intersections by applying direction
    applyDirection(x, y, direction);
  }
  ```
Break into major 'jobs' that need to be done

- decide initial intersection
- random pick of direction
- change intersections
- ...
- run 100 times
- print beginning and ending locations
int x = 0, y = 0;

for (count = 0; count < 100; count++)
{
    int direction = randomPick();
    applyDirection(x, y, direction);
}
printIntersections(x, y);
int x = 0, y = 0;

for (count = 0; count < 100; count++)
{
    int direction = randomPick();
    applyDirection(x, y, direction);
}
printIntersections(x, y);
int randomPick()

1: up
2: down
3: left
4: right
int randomPick()

//Note: rand()
//returns a random number
// [0, RAND_MAX]
//need it in the range of 1 - 4
int randomPick()

//Note:  rand()
//returns a random number
//  [0, RAND_MAX]
//need it in the range of 1 - 4

{
    int val = rand() % 4 + 1;
    return val;
}
void applyDirection(int & x, int & y, int direction);

// 1:up  2:down  3:left  4:right
{
    if (direction == 1) {
        y ++;
    } else if (direction == 2) {
        y--;
    } else if (direction == 3) {
        x--;
    } else {
        x++;
    }
}
void printResults(int x, int y)
{
    cout << "start at: (0,0)" << endl;
    cout << "end at: (" << x << ", " << y << ")' " << endl;
}
Summary

- Start very general
  - determine "jobs" -- functions
  - determine order, loops, input
- iterate to more specific
- add details later
  - postpone details
  - shove to functions
NEVER - NEVER - NEVER

- start with details
Beginning with details, leads to
Beginning with details, leads to