1. Command line / Menu

Estimated time: 2h home work, 3h scheduled lab

You need to understand basic language constructs and input, output with scanf(), printf(), getchar(), putchar() etc.. Build a simple file. It is good if you can create a simple function.

Assignment 1

Create the start of our program for show-jumping competition administration. Do a command line and menu for the program. Your program should:

• Read a command from the user

• Depending on the command given execute different tasks

During the course of the labs you will add functionality to your menu. Thus you will gain from a design that make addition and modification of the program simple. To begin with your program will only recognize three commands that will:

• Print a list of the commands available (a menu)

• Print information about the program (program name, authors, copyright, year, version)

• Exit the program

It is your choice how the menu should look and how advanced it should be. A simple command line that reads the number of the menu item to select is sufficient. (Although you may want to modify such solution later on when you learned more).

If you are a beginner, solve one problem at a time. Start with a program that just print a menu and exit. Add code that asks for user input and print whatever the user entered. Add more code so the program print different things depending on what the user entered. At last, make the program repeat itself instead of exiting immediately.

Assignment 2

Test your menu with as much and as weird input you can figure out. What happen if you enter characters and not digits? What if the user of your program put lot of blanks and/or newlines before or after the input? What happens if the user hit CTRL-Z?

Fix your program so that it seems impossible to crash, hang or otherwise corrupt.

2. Result recording

Estimated time: 4h home work, 6h scheduled lab

Topics

You need further input, output understanding. stringstreams can be useful. How to format output. How to use the Time header file. Including, using and compiling “own” code (the timer) that resides in separate file. It is good if you can create a function with reference parameters. It is good if you can separate your code in different files. Study the complete example program given.

What do the manipulators setprecision and fixed in iomanip do? How does getline () work? How can the separete character be ignored as inputing? You may also find the interesting functions in string header file.

Assignment 1

Write a program that assist time-measurement and result recording during a showjumping competition. (Horse and rider dashing round a track jumping obstacles. Each combination horse and rider is called an equipage.) The competition (of one equipage) goes as follow:

1. The equipage enters the track

2. The referee say “start at will”

3. The equipage passes the start line

4. The equipage possibly pulls down obstacles, or the horse refuse to jump (disobedience). the rider may also fall off etc..

5. The equipage passes the goal line

6. The equipage exit the track

7. The referee tell the result of the ride, how fast the ride was (the track time), and how many penalties the equipage received.

The program should measure the time between point 2. and 3. (the preparation time), and between point 3. and 5. (the track time).

Between point 2. and 3. the rider has a certain preparation time. If it is exceeded the exceeding time is added to the track time.

If the final track time (after adjustments) exceeds a set maximum track time, then one penalty for each started exceeding second is added to the penalties.

After each equipage finish - if it finish (see assignment 2) - the program should print the following to support the referee:

• The total number of penalties

• How many penalties that resulted from exceeding maximum track time

• The final track time

You can study an example run of the program below. The preparation time and

maximum track time is common for all equipages and asked for once at the start of the program.

Assignment 2

At point 4. the program should understand the codes ‘p’ for pulldown (4 penalties), ‘r’ for refusal (4 penalties first time, elimination second time), ‘d’ for “did not finish” and any number should be interpreted as time to be substracted from the total track time (due to some event causing the need for obstacle repair before the ride could continue).

Guidelines

• Do one step at a time. Start with a program that only display the user instructions. Add input handling so the user must press `enter` to continue. Then add timing between the input events, and corresponding calculation and output. Finally, add interpretation of any codes entered in step 4 (use a subroutine for this).

• You are supposed to reuse the Timer class provided, not implement your own.

• You do NOT have to save anything, printing the result to display for referee support is enough.

• You do NOT need any advanced input management that immediately reacts to keys, it is enough the program read an entire input line and process it after the user press `enter`. *In fact, the user should not be required to press more keys than absolutely necessary (and if he does they should be ignored), in most cases just pressing `enter` should do the trick.*

Additional requirements

• All time-measures should be printed with two decimals in fixed point form, as in the example below.

• Use a subroutine to calculate the penalties from the input line.

• Create a working execute file

Example output

User input is given in bold here to separate it from program output. Also **§** represent the places where the user pressed enter (new line).

```
Example<1> lab0§

Enter preparation time: 45.00§
Enter max time: 34.00§

!!! Start number 1.
<press enter at clearance to start>
[n=stop result recording]§
countdown from 45.00 started

<press enter at start line passage>§
preparation took 46.30 seconds
timing started at 1.30 seconds

<record penalties and press enter to stop timing>
[p=pulldown,r=refusal,d=dnf,(number)=paus time]
prp4.0§
timing stopped at 39.46 seconds

Total time 35.46 seconds (1.46 seconds above max)
Total penalties: 14 (2 due to time)

!!! Start number 2.
<press enter at clearance to start>
[n=stop result recording]§
countdown from 45.00 started

<press enter at start line passage>§
preparation took 16.21 seconds
timing started at 0.00 seconds

<record record penalties and press enter to stop timing>
```

```
[p=pulldown,r=refusal,d=dnf,(number)=paus time]§
timing stopped at 33.49 seconds

Total time 33.49 seconds
Total penalties: 0

!!! Start number 3.
<press enter at clearance to start>
[n=stop result recording]n§
result recording done.
```

3. Start list

Estimated time: 4h home work, 6h scheduled lab

You will need to understand struct, string and how to use a array and iterators. You need to use functions and parameter passing to structure your program. It is good to separate the code in different files. You may use class and member functions.

Assignment 1

When competing in show-jumping, and indeed in many sports, it is common to have a predefined list of competitors defining the order in which to start. For each start number we keep track of some personal data. In show-jumping we need to keep track of each equipage and the club/country they compete for. An equipage consist of a horse and it's rider (the same rider may compete on several horses, but you do not need to consider this yet). You will save the name and license number of each rider, and name, identification number and age of each horse.

Add a menu option to you program that will allow the user to add new equipages. It is preferable to be able to add several equipages in a row, as to not have to enter the menu choice over and over again. It is however to this point sufficient if you can add one equipage at a time.

Assignment 2

Again, make sure your program is safe and do not crash on unexpected input. If the user enter invalid data the program should complain and ask for the correct data until the user gets it right.

Assignment 3

The list of competitors is not particularly useful if you can not view it. Add an option to print the entire list of competitors, both in order and reverse order. Only add one option to your top menu, and add some way to specify the order when selecting that command. The more user friendly the better.

4. Start list management

Estimated time: 4h home work, 3h scheduled lab

You need ability to read and understand reference manuals. Using rand function. Using iterators.

Assignment 1

As mentioned one rider may compete on several horses. It is then important that the rider has sufficient time to change horse between two rounds. Add an option that allow the user to swap two equipages in the startlist, as to be able to change the order.

Assignment 2

It may be an advantage to start first, in the middle or last in different situations. Add a function so that the user easily can randomize the start list. This can be done in many ways, the “harder” method will teach you more about coding, the “easier” more about finding and reading documentation and references. It is your choice. A random number generator can be found in the standard C library <cstdlib>. The functions needed is named `srand` and `rand`. Find some reference manual to learn them.

Assignment 3

Sometimes a competitor gets ill or cannot enter the competition as planned. Add a way to remove a competitor. You should be able to remove by start number, by license and by name. You should also be able to clear the entire list through the same command.

Assignment 4 (Optional)

Add a new option to your “print startlist” command that list only equipages that compete for a given club/country.

Assignment 5 (Optional)

Search a equipage based on name or license.

Assignment 6 (Optional and difficult)

Make sure all riders that enter the competition with more than one horse are separated as far as possible in the startlist (to give them time to prepare the second horse).

5. Using file streams

Estimated time: 4h home work, 3h scheduled lab

You need to understand file streams and how to use them. You may find use of the functions in the <ctype> include library.

Assignment 1

Currently all data disappear as soon as you exit the program. Add an option to store the list to a textfile. You should store one equipage on each line with different fields separated by colon:

```
Ludger Beerbaum:1234:L'espoir:6754:10:Germany:
Beezie Madden:6346:Autentic:2582:12:USA:
```

The fields above is in order rider name, license, horse name, id. number, age and club/country.

Assignment 2

Add a functionality to read the stored list back from file. It should be possible to save the work, exit the program, and then start administrating a competition where you finished last time.

Assignment 3

Try to store an equipage that competes for a club named "`.: Young Riders :.`", or otherwise contain colons in some field. Exit and load the file. Make sure it works correctly.

Assignment 4

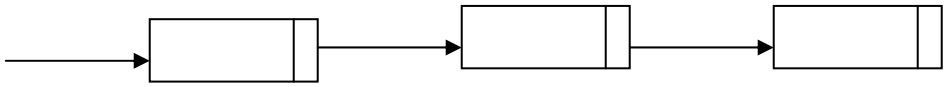
The user may exit the program unintentionally without saving. The program should ask for confirmation where just pressing enter should confirm. Writing Y or Yes in any combination of upper or lower case immediately followed by enter should also confirm. Anything else should be ignored and not exit the program.

6. Event list

Estimated time: 4h home work, 6h scheduled lab

Preparations

A linked list is a data structure with the data stored in nodes, and each node contain a pointer to the next node in the list. A list with three nodes:



What is a pointer? How can you represent this in C?

How do you insert something first? Second? Middle? Last?

How do you allocate and deallocate memory with new and delete?

Assignment 1

In the second lab you used a timer. If you remember it only measured the time elapsed from the last call to the reset function. Recording of several events at different times required the programmer to keep track of the events.

In this lab you should modify the Timer function to relieve the programmer of this task. The new version of the Timer should be able to remember the time at which different events, represented by strings, occur. There should be no set limit on the number of events that can be remembered. You will need a internal datatype to represent an event. The events should be stored as a single linked list in the Timer function. This functionality is required:

• Initiate a timer variable from an existing timer, the entire list of events must be copied to the new variable (and remain in the old). This should be done in the copy sub function.

• Assigning one timer (source) to an other (target), the original list in target must be removed, and the entire list of events must be copied from source to target. This is done in the assignment operator function.

• Reset the timer. This should remove all stored events. All subsequently added events should be relative to this point in time.

• Adding a event, the current time of the timer (as retrieved by the code in operator double) should be associated with the event (represented by a string). All events times should be stored relative to the time of the last call to reset. That is, reset is regarded as time zero. (The timer is currently automatically reset when created.)

• Remove a specified event from the timer

• Print a list or timeline of the stored events and associated time

• Calculate the number of stored events

• Finding a event, it should return the time the searched event occurred on or -1 if it was not found.

All memory dynamically allocated with new must be released with delete. It is part of the task to make sure the timer works as specified above. You do not have to integrate it with previous tasks in this assignment.

Optional Assignment

If you are not interested in this lab assigment, and you can realize a certain task which should be completed by pointer array and some functions. It is better to talk with your TA.

7. Decode (optional extra assignment)

Estimated time: 4h home work, 6h scheduled lab

The program that follow:

```
#include <stdio.h>
int main()
{
  char c;
  int count = 0;
  while (getchar(c))
  {
    if (c == ' ')
      count++;
    else {
      printf("%d", count);
      putchar(c);
      count = 0;
    }
    Putchar('\n');
    return 0;
}
```

Generated this text on a certain input not containing any digits:

```
17.2.15_0_1.10,12
17|0\0/0|1_1.0_0.0.0_0.2.2/2‘0|0_1.0_0.0*1_0_0-0+0-
0.0_1_2_0.1_0_0
17|2|0(0/
0,0[2[2\0_0|2\0_0_0.0[1)0[2|0_0)2|1[1|1)0(0_0]0_0)1
30.0_0|30
0
16.2.14.2.11.3,9
16|0_0_0|1_0.0.0_1.0_3.2|0\1|1_1.4,3\0.0/2_2_0.0.0_0.0
16|2|0(0_0]0[0_0)0[0_0)0\0_0|2|1\0|0(0/0,1\0/0\0/5|2(0/
0,0(0_0]0[2
23|2|2.0_0|
```

Write a program that print the original input on the screen.