Workshop #4: Constructors, Destructors and Current object

- Version 1.0

In this workshop, you will use Constructors, Destructor and reference of the current object to simulate a Tournament with soccer teams to find out the winner.

Learning Outcomes

Upon successful completion of this workshop, you will have demonstrated the abilities to:

- define default constructor
- define custom constructor with different number of arguments
- define a Destructor to prevent memory leak.
- use reference of the current object
- describe to your instructor what you have learned in completing this workshop

Submission Policy

This workshop is divided into two coding parts and one non-coding part:

- Part 1 (**LAB**): A step-by-step guided workshop, worth 50% of the workshop's total mark
  > Please note that the part 1 section is **not to be started in your first session of the week**. You should start it on your own before the day of your class and join the first session of the week to ask for help and correct your mistakes (if there are any).
- Part 2 (**DIY**): A Do It Yourself type of workshop that is much more open-ended and is worth 50% of the workshop's total mark.
- *reflection*: non-coding part, to be submitted together with *DIY* part. The reflection doesn't have marks associated with it but can incur a **penalty of max 40% of the whole workshop's mark** if your professor deems it insufficient (you make your marks from the code, but you can lose some on the reflection).
- Submissions of part 2 that do not contain the *reflection* (that is the **non-coding part**) are not considered valid submissions and are ignored.

Due Dates

The Due dates depend on your section. Please choose the "-due" option of the submitter program to see the exact due date of your section:

```
~profname.proflastname/submit 2??/wX/pY_sss -due<ENTER>
```

- Replace **??** with your subject code (`00 or 44`)
- Replace **X** with Workshop number: [`1 to 10`]
- Replace **Y** with the part number: [`1 or 2`]
- Replace **sss** with the section: [`naa, nbb, nra, zaa, etc...`]

Late penalties

You are allowed to submit you work up to 2 days after due date with 30% penalty for each day. After that the submission will be closed and the mark will be zero.

Citation

Every file that you submit must contain (as a comment) at the top:
your name, **your Seneca email**, **Seneca Student ID** and the **date** when you completed the work.

For work that is done entirely by you (ONLY YOU)

If the file contains only your work or the work provided to you by your professor, add the following message as a comment at the top of the file:

> I have done all the coding by myself and only copied the code that my professor provided to complete my workshops and assignments.

For work that is done partially by you.

If the file contains work that is not yours (you found it online or somebody provided it to you), **write exactly which part of the assignment is given to you as help, who gave it to you, or which source you received it from.** By doing this you will only lose the mark for the parts you got help for, and the person helping you will be clear of any wrongdoing.

> - Add the citation to the file in which you have the borrowed code
> - In the 'reflect.txt` submission of part 2 (DIY), add exactly what is added to which file and from where (or whom).

> ⚠️ This Submission Policy only applies to the workshops. All other assessments in this subject have their own submission policies.

If you have helped someone with your code

If you have helped someone with your code. Let them know of these regulations and in your 'reflect.txt' of part 2 (DIY), write exactly which part of your code was copied and who was the recipient of this code.
By doing this you will be clear of any wrongdoing if the recipient of the code does not honour these regulations.

Compiling and Testing Your Program

All your code should be compiled using this command on `matrix`:

```
g++ -Wall -std=c++11 -g -o ws file1.cpp file2.cpp ...
```

- `-Wall`: the compiler will report all warnings
- `-std=c++11`: the code will be compiled using the C++11 standard
- `-g`: the executable file will contain debugging symbols, allowing *valgrind* to create better reports
- `-o ws`: the compiled application will be named `ws`

After compiling and testing your code, run your program as follows to check for possible memory leaks (assuming your executable name is `ws`):

```
valgrind --show-error-list=yes --leak-check=full --show-leak-kinds=all --track-origin
```

- `--show-error-list=yes`: show the list of detected errors
- `--leak-check=full`: check for all types of memory problems
- `--show-leak-kinds=all`: show all types of memory leaks identified (enabled by the previous flag)
- `--track-origins=yes`: tracks the origin of uninitialized values (`g++` must use `-g` flag for compilation, so the information displayed here is meaningful).

To check the output, use a program that can compare text files. Search online for such a program for your platform, or use *diff* available on `matrix`.

> Note: All the code written in workshops and the project must be implemented in the **sdds** namespace, unless instructed otherwise.

Custom code submission

If you have any additional custom code, (i.e. functions, classes etc) that you want to reuse in the workshop save them under a module called Utils (`Utils.cpp and Utils.h`) and submit them with your workshop using the instructions in the "Submitting Utils Module" section.

Part 1 - LAB (50%)

In this workshop you have to Implement two modules (i.e., classes): **Tournament** and **SoccerTeam**

The SoccerTeam module

Develop this module in two files named **soccerTeam.h** and **soccerTeam.cpp**. Create a structure named **soccerTeam** and the structure should have the following member variables (attributes) and member functions (method): First, define the following constant in the **soccerTeam** header file:

```
const int MAX_FOUL = 4; //maximum number of fouls
```

member variables(attributes)

The struct soccerTeam should have the following member variables:

```
    char m_teamName[41];// a statically allocated Cstring with size 41. Remember, nam
                        //maximum 40 characters long and 1 byte is for the null byte.
    int m_noFouls;//number of fouls, it can be zero or more but cannot be a negative
    double m_fines;//it can be equal to and more than zero.
    int m_golas;//can be zero or more
```

member functions(methods)

```
void setTeam(const SoccerTeam& team)
```

It will set the team name, fine and fouls of the team.

```
void setName(const char* tname)
```

If the received name pointer is valid and not null it will copy the received name to the data member **m_teamName**

```
void setFine(double fines, int foul)
```

It will receive the information about fines and no of foul. After checking the validity it will set the values to the appropriate data members. Fine and foul should be grater than and equal to zero. Otherwise, it will set the soccerTeam object to an empty state.

```
void setEmpty();
```

Sets the **soccerTeam** object to an Empty State. Do this by setting the **m_teamName** to an empty Cstring, **m_noFoul** and **m_fines** to a negative number and **m_goals** to **0**.

```
bool isEmpty() const;
```

Returns true if **m_teamName** is not null, **m_fines** and **m_noFouls** is grater than **0**.

```
void calFines();
```

If this function is called it will increases the fine by 20% of the old fine value.

```
int fouls() const ;
```

It will return the m_noFoul

A soccerTeam can be created in two different ways:

```
soccerTeam();
```

- By default a **soccerTeam** is set to the empty state (as in setEmpty())

Three argument constructor

```
SoccerTeam(const char* tname, double fines, int foul);
```

After checking the validity it will set the team`s name,fine and no of fouls to the appropriate data members or else it will set the team to the empty state. (reuse your setter fucntions)

```
std::ostream& display()const;
```

1. If a soccerTeam object is valid

   a) prints **m_teamName** with width 30, left justified and fill with empty spaces.

   b) prints **m_fines** with width 6 and after the decimal point 2 digits.

   c) prints **m_noFoul** with width 6

   d) prints **m_fines** with width 6

   e) prints **m_golas** with width 10. If number of goals are greater than zero then it will print "w" besides the number.

2. otherwise prints, "Invalid Team".

3. At the end return the reference of the ostream object.

The Tournament module

Develop this module in two files named **Tournament.h** and **Tournament.cpp**. Place your class definition in **Tournament.h** and your function definitions in **Tournament.cpp**. Create a class named **Tournament** and the class should have the following member variables (attributes) and member functions (method):

Private member variables(attributes)

The class should have the following private member variables:

```
    char* m_name;//points to a dynamically allocated Cstring
    int m_num;//size of the dynamically allocated array of soccer team. It should be
              zero.
    SoccerTeam* m_soccer=nullptr;//pointer to the dynamically allocated array of socc
```

Public member functions(methods)

```
void setTournament(const char* name, int noOfteam,const SoccerTeam* soccer);
```

- First, it will check the validity of all the received arguments. Tournament name should be valid and non-empty Cstring. Number of teams should be greater than zero.

- It will set **m_num** to the corresponding argument value.
- Then it will dynamically allocate memory for tournament name in the tournament name pointer attribute and then copies the content of the tournament name argument into the newly allocated memory.
- It will dynamically allocate an array of soccer team pointed by **m_soccer** member variable. The length of this array will be **m_num**.
- Lastly, add all the teams to the dynamically allocated array of **m_soccer**.
- If any of the arguments are not valid it will set the Tournament to an empty state.

```
    void setEmpty();
```

Sets the **Tournamnet** object to an Empty State. Do this by setting the **m_name** and **m_soccer** to null and **m_num** to **0**.

```
bool isEmpty() const;
```

Returns true if **m_name** and **m_soccer** is not null and **m_num** is grater than **0**.

```
Tournament& match(const SoccerTeam* ls);
```

This function will find out the winner between 2 soccer teams by having matches. It will loop through all the teams.

- First it will check which team is having more no of fouls.
- If first team is having less fouls then second team then, second team`s no of foul will increase by double and fine will be increased by 20%
- First team`s goal will be increased by 1.
- If seconds team`s no of foul exceed the MAX_FOUL then this team will become a invalid team. You can do this by setting the number to foul to an invalid value.
- At the end it will return the reference of the current object.

```
    ostream& display() const;
```

1. If Tournament object is valid

   a) prints "Tournament name: " then tournament name

   b) prints "list of the teams" after that a newline

   c) prints "Fines" with width 45 and fill with empty spaces.

d) prints "Fouls" with width 10 and fill with empty spaces.

e) prints "Goals" with width 10 and fill with empty spaces.

e) prints all the soccer teams information. For details see the sample output.

2. otherwise prints, "Invalid Tournament".
3. At the end return the reference of the ostream object.

Sample output

```
Tournament name : Soccer Tournament
list of the teams
                                       Fines     Fouls     Goals
Team[1] : Scarborough Soccer Team       0.00     0         1w
Team[2] : North York Soccer Team      120.00     2         0
```

construction and destruction

A **Tournament** can be created in two different ways:

```
    Tournament();
```

- By default a **Tournament** is initiated by setting all the member variable values to default values. You can do this by setting **m_name**, **m_soccer** to nullptr and **m_num** to a value like **0**.

Three argument constructor

```
    Tournamnet(const char* name, int noOfteam,const SoccerTeam* soccer);
```

It works exactly like the setTournamnet()

```
   ~Tournament();
```

Deallocate the memory allocated by **m_name** and **m_soccer**.

Tester Program

main.cpp

Execution Sample

correct_output.txt

PART 1 Submission (lab)

Files to submit:

```
SoccerTeam.cpp
SoccerTeam.h
Tournament.cpp
Tournament.h
main.cpp
```

Custom code submission

If you have any additional custom code, (i.e. functions, classes etc) that you want to reuse in this workshop save them under a module called Utils (`Utils.cpp and Utils.h`) and submit them with your workshop using the instructions in the "Submitting Utils Module" section.

Submission Process:

Upload the files listed above to your `matrix` account. Compile and run your code using the `g++` compiler as shown in Compiling and Testing Your Program and make sure that everything works properly.

Then, run the following command from your matrix account

```
~profname.proflastname/submit 2??/wX/pY_sss  <ENTER>
```

- Replace **??** with your subject code (`00 or 44`)
- Replace **X** with Workshop number: [`1 to 10`]
- Replace **Y** with the part number: [`1 or 2`]
- Replace **sss** with the section: [`naa, nbb, nra, zaa, etc...`]

and follow the instructions.

Submitting Utils Module

To have your custom Utils module compiled with your workshop and submitted, add a **u** to the part number of your workshop (i.e **u**p1 for part one and **u**p2 for part two) and issue the following submission command instead of the above:

```
~profname.proflastname/submit 2??/w#/upX
```

See Custom Code Submission section for more detail

> ⚠**Important:** Please note that a successful submission does not guarantee full credit for this workshop. If the professor is not satisfied with your implementation, your professor may ask you to resubmit. Re-submissions will attract a penalty

DIY (50%)

> Please note that you can (and probably should) add more member functions to make the DIY part work.

Create a Module for a NameTag to create/read information for a name tag and print it.

The class name must be **NameTag**.

It should be able to hold two pieces of information; A name and a 5 digit extension number. The name should be kept dynamically; however if the name is longer than 40 characters, only 40 characters should be printed in the name tag.

Also if the name is shorter than 20 characters, it should be printed in 20 spaces. The extension number is optional and if not provided, it should be printed as **N/A**.

Print the name tag by drawing a box around it. The box must have five interior lines and the name should have exactly one space distance with the line of the box at left and minimum of one space with the line at right. The name and the extension are printed on lines 2 and 4 and the rest of the name tag is filled with spaces.

Here are couple of examples:
(Name: "Fred Soley", extension 12345):

```
+----------------------+
|                      |
| Fred Soley           |
|                      |
| Extension: 12345     |
|                     |
+---------------------+
```

(Name: "David Wright Mason Gilmour Waters Rogers Nick", extension not available)

```
+----------------------------------------+
|                                        |
| David Wright Mason Gilmour Waters Rogers |
|                                        |
| Extension: N/A                         |
|                                        |
+----------------------------------------+
```

If the data provided are invalid, (name being null or extensions being more or less than 5 digits) the printout will be:

```
EMPTY NAMETAG!
```

Public and mandatory functions, constructors and destructor

constructors

- default constructor (for an empty tag)
- constructor with one argument to set the name without extension number
- constructor with two arguments to set the name and the extension

destructor

- have a destructor to avoid memory leaks

Functions:

print

a function called **print** to print the name tag as shown above

read

a function called **read** that receives the name and then the extension number and then returns the reference of the current object (NameTag&).

Read function should read the name up to the maximum of 40 characters and ignore the rest and then set the name of the **NameTag** to the entered name.
(This may overwrite an already existing name in the **NameTag**)
Then it should give an option to the user to enter the extension number or not. If the user chooses to enter the extension it should enforce the user to enter a 5 digit integer and if user does not comply, it should print one of the following messages based on the type of the error:

```
Bad integer value, try again:
```

```
Invalid value [10000<=value<=99999]:
```

and get the new value and repeat until the user enters a correct value.

A sample for execution is provided below; Note that the **read()** execution is demonstrated right before the name tag printouts.(The exact tester output is in Execution Sample Section)

```
Please enter the name: Hubert Blaine Wolfeschlegelsteinhausenbergerdorff
Would you like to enter an extension? (Y)es/(N)o: x
Only (Y) or (N) are acceptable, try agin: n
+------------------------------------------+
|                                          |
| Hubert Blaine Wolfeschlegelsteinhausenbe |
|                                          |
| Extension: N/A                           |
|                                          |
+------------------------------------------+
Please enter the name: Lisa Simpson
Would you like to enter an extension? (Y)es/(N)o: Y
Please enter a 5 digit phone extension: five
Bad integer value, try again: 9999
Invalid value [10000<=value<=99999]: 100000
Invalid value [10000<=value<=99999]: 12345
+----------------------+
|                      |
| Lisa Simpson         |
|                      |
| Extension: 12345     |
|                      |
+----------------------+
```

Tester program:

main.cpp

Sumbmission Execution Sample

Here is the execution sample for the tester program (submission execution output will be different)

correct_output.txt

> Modify the tester program to test are the different circumstances/cases of the application if desired and note that the professor's tester may have many more samples than the tester program here.

Reflection

Study your final solutions for each deliverable of the workshop, reread the related parts of the course notes, and make sure that you have understood the concepts covered by this workshop. **This should take no less than 30 minutes of your time and the result is suggested to be at least 150 words in length.**

Create a file named `reflect.txt` that contains your detailed description of the topics that you have learned in completing this workshop and mention any issues that caused you difficulty.

You may be asked to talk about your reflection (as a presentation) in class.

Part 2 Submission (DIY)

Files to submit:

```
NameTag.cpp
NameTag.h
main.cpp
reflect.txt
```

Data Entry

Follow instruction in professor's tester program

Submission Process:

Upload the files listed above to your `matrix` account. Compile and run your code using the `g++` compiler as shown in Compiling and Testing Your Program and make sure that everything works properly.

Then, run the following command from your matrix account

```
~profname.proflastname/submit 2??/wX/pY_sss  <ENTER>
```

- Replace **??** with your subject code (`00 or 44`)
- Replace **X** with Workshop number: [`1 to 10`]
- Replace **Y** with the part number: [`1 or 2`]
- Replace **sss** with the section: [`naa, nbb, nra, zaa, etc...`]

and follow the instructions.

Submitting Utils Module

To have your custom Utils module compiled with your workshop and submitted, add a **u** to the part number of your workshop (i.e **u**p1 for part one and **u**p2 for part two) and issue the following submission command instead of the above:

```
~profname.proflastname/submit 2??/w#/upX
```

See Custom Code Submission section for more detail

> ⚠**Important:** Please note that a successful submission does not guarantee full credit for this workshop. If the professor is not satisfied with your implementation, your professor may ask you to resubmit. Re-submissions will attract a penalty