User input in C++:

In C++ user input is taken using “cin” object. The cin object in C++ is an object of class iostream. It is used to accept the input from the standard input device i.e. keyboard. It is associated with the standard C input stream stdin. The extraction operator(>>) is used along with the object cin for reading inputs. The extraction operator extracts the data from the object cin which is entered using the keyboard.

Program:

```
#include<iostream>

using namespace std;

int main(){

        int marks;

        cout<<"Enter your Marks: ";

        cin>>marks;

        cout<<"your Marks : "<<marks<<endl;

}
```

Output:

```
        PS E:\C++ PROGRAMMES> g++ inputincpp.cpp

        PS E:\C++ PROGRAMMES> ./a.exe

        Enter your Marks: 99

        your Marks : 99
```

Note: If we read a Boolean value as true or false it does not consider as Boolean values. We have to give the input as 0 or 1.

Program:

```
#include<iostream>

using namespace std;

int main(){

   bool isCorrect;

   cout<<"sun rises on the east: ";

   cin>>isCorrect;

   cout<<"sun rises on the east is : "<<isCorrect<<endl;

}
```

Output:

```
PS E:\C++ PROGRAMMES> g++ boolinput.cpp
PS E:\C++ PROGRAMMES> ./a.exe
sun rises on the east: 1
sun rises on the east is : 1
PS E:\C++ PROGRAMMES> ./a.exe
sun rises on the east: 0
sun rises on the east is : 0
PS E:\C++ PROGRAMMES> ./a.exe
sun rises on the east: true
sun rises on the east is : 0
```

Home Work Questions:

1. **cin.ignore():**

In C++, cin.ignore() is a function used to discard characters from the input buffer. It is often used in conjunction with the cin (standard input stream) to clear any unwanted characters, such as newline characters, left in the input buffer after previous input operations.

The general syntax for cin.ignore() is as follows:

```
#include <iostream>
int main() {
        // Some code...
        std::cin.ignore(); // Ignores one character from the input buffer
        // Some more code...
        return 0;
}
```

By default, cin.ignore() ignores a single character. However, you can provide additional arguments to customize its behavior. For example:

```
#include <iostream>
int main() {
   // Some code...
  std::cin.ignore(100, '\n'); // Ignores up to 100 characters or until a newline is encountered
```

```
    // Some more code...

    return 0;
}
```

In this example, cin.ignore(100, '\n') will ignore up to 100 characters or stop if a newline character ('\n') is encountered, effectively discarding the rest of the current line.

Using cin.ignore() is helpful when you want to clear the input buffer before reading user input, especially when switching between different types of input (e.g., from numeric input to string input). It helps avoid unexpected behavior caused by lingering newline characters or other unwanted input.

For More Reference: https://cplusplus.com/forum/beginner/182907/

2. cin.fail()

In C++, `cin.fail()` is a function that checks whether the last input operation on the `cin` (standard input stream) encountered an error. The `cin.fail()` function returns `true` if an error occurred, indicating that the expected input type was not successfully read.

Here's an example program that demonstrates the use of `cin.fail()`:

```
#include <iostream>

int main() {
    int num;

    // Prompt the user for an integer
    std::cout << "Enter an integer: ";
    std::cin >> num;

    // Check if the input operation was successful
    if (std::cin.fail()) {
        std::cout << "Invalid input. Please enter an integer." << std::endl;

        // Clear the error flag
        std::cin.clear();

        // Ignore the rest of the input buffer up to a newline character
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
    } else {
        std::cout << "You entered: " << num << std::endl;
    }

    return 0;
}
```

In this example:

1. The program prompts the user to enter an integer.

2. After the attempt to read an integer (`std::cin >> num`), `std::cin.fail()` is used to check if the operation was successful.

3. If `std::cin.fail()` returns `true`, it means that the input was not an integer, and the program handles the error by printing a message, clearing the error flag (`std::cin.clear()`), and ignoring the remaining characters in the input buffer up to the next newline character.

This approach helps handle invalid input and prevents the program from getting stuck in a loop of failed input attempts. The `std::numeric_limits<std::streamsize>::max()` is used as the second argument to `std::cin.ignore()` to ignore the maximum number of characters in the input buffer until a newline is encountered.

3. **getline():**

   It seems like there might be a misunderstanding in your usage of `getline`. The `getline` function in C++ is used to read a line of text from an input stream and store it in a string. The syntax is typically:

```
#include <iostream>
#include <string>
int main() {
    std::string input;

    std::cout << "Enter a line of text: ";
    std::getline(std::cin, input);

    std::cout << "You entered: " << input << std::endl;

    return 0;
}
```

   In this example, `std::getline(std::cin, input)` reads a line of text from the standard input (`std::cin`) and stores it in the `input` string. The second argument (`input`) is the variable where the line of text will be stored.

If you specifically want to read a maximum of 10 characters using `getline`, you can modify the example as follows:

```
#include <iostream>
#include <string>
int main() {
    std::string input;

    std::cout << "Enter up to 10 characters: ";
    std::getline(std::cin, input, '\n');  // Specify newline as the delimiter

    // Keep only the first 10 characters
    input = input.substr(0, 10);

    std::cout << "You entered: " << input << std::endl;

    return 0;
}
```

In this modified example, `std::getline(std::cin, input, '\n')` reads a line of text until a newline character (`'\n'`) is encountered or up to the specified maximum size. The subsequent `input.substr(0, 10)` ensures that only the first 10 characters are kept.