Functions Overloading

delete function

```
#include <iostream>

void printInt(int x)
{
    std::cout << x << '\n';
}

void printInt(char) = delete; // calls to this function will halt
compilation
void printInt(bool) = delete; // calls to this function will halt
compilation

int main()
{
    printInt(97);   // okay

    printInt('a');  // compile error: function deleted
    printInt(true); // compile error: function deleted

    printInt(5.0);  // compile error: ambiguous match

    return 0;
}
```

and an example with a template

```
#include <iostream>

// This function will take precedence for arguments of type int
void printInt(int x)
{
    std::cout << x << '\n';
}

// This function template will take precedence for arguments of other types
// Since this function template is deleted, calls to it will halt
compilation

```

```
template <typename T>
void printInt(T x) = delete;

int main()
{
    printInt(97);   // okay
    printInt('a');  // compile error
    printInt(true); // compile error

    return 0;
}
```

default arguments

```
#include <iostream>

void print(int x, int y=4) // 4 is the default argument
{
    std::cout << "x: " << x << '\n';
    std::cout << "y: " << y << '\n';
}

int main()
{
    print(1, 2); // y will use user-supplied argument 2
    print(3); // y will use default argument 4, as if we had called print(3,
4)

    return 0;
}
```

Default arguments can not be redeclared, and must be declared before use

The best practice is to declare the default argument in the forward declaration and not in the function definition, as the forward declaration is more likely to be seen by other files and included before use (particularly if it’s in a header file).

in foo.h:

```
#ifndef FOO_H
#define FOO_H

```

```
void print(int x, int y=4);
#endif
```

```
#include "foo.h"
#include <iostream>

void print(int x, int y)
{
    std::cout << "x: " << x << '\n';
    std::cout << "y: " << y << '\n';
}

int main()
{
    print(5);

    return 0;
}
```