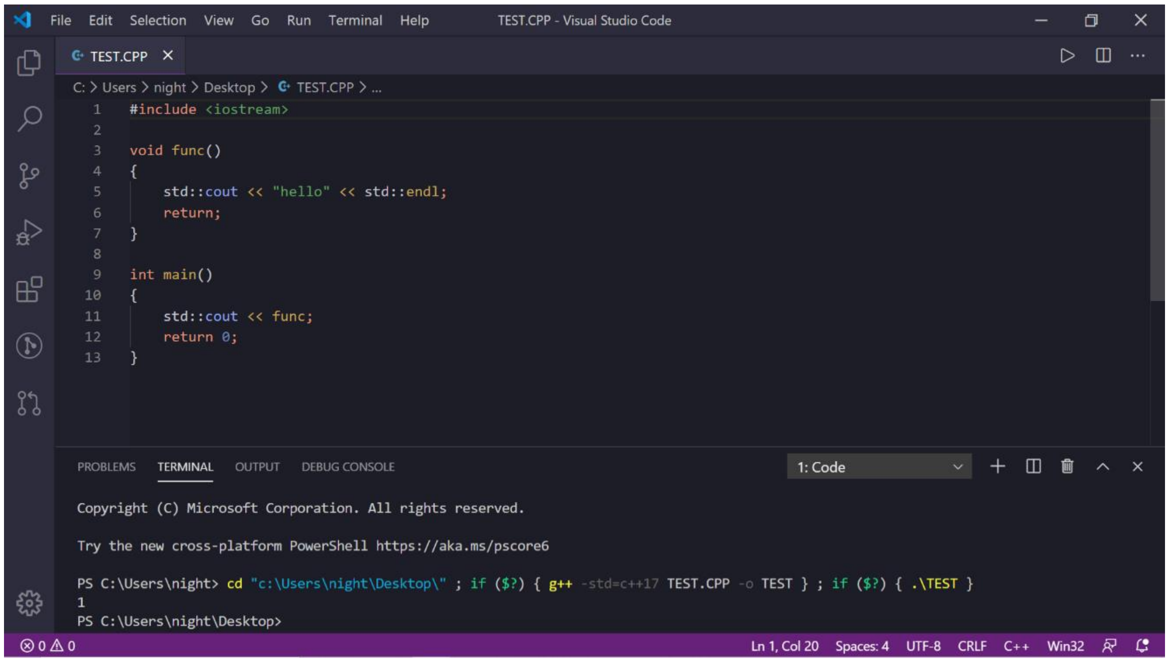
REPORT ABOUT FUNCTION POINTER

If we ever mistakenly output a function instead of calling the function, the output would be 1. This is just a value that modern compilers show when displaying a function address, older compilers would output a memory address.


File Edit Selection View Go Run Terminal Help
TEST.CPP - Visual Studio Code
TEST.CPP
C: > Users > night > Desktop > TEST.CPP > ...
PROBLEMS TERMINAL OUTPUT DEBUG CONSOLE
1: Code
Ln 1, Col 20 Spaces: 4 UTF-8 CRLF C++ Win32

```cpp
#include <iostream>

void func()
{
    std::cout << "hello" << std::endl;
    return;
}

int main()
{
    std::cout << func;
    return 0;
}
```

```
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\night> cd "c:\Users\night\Desktop\" ; if ($?) { g++ -std=c++17 TEST.CPP -o TEST } ; if ($?) { .\TEST }
1
PS C:\Users\night\Desktop>
```

A function is also a memory address, just like an integer, double… also have their addresses when they are defined. A pointer is a type of data that points to the memory address of variables. If we consider a function as a variable, we can use a pointer to points to that function. This is what we call a function pointer.

SYNTAX:

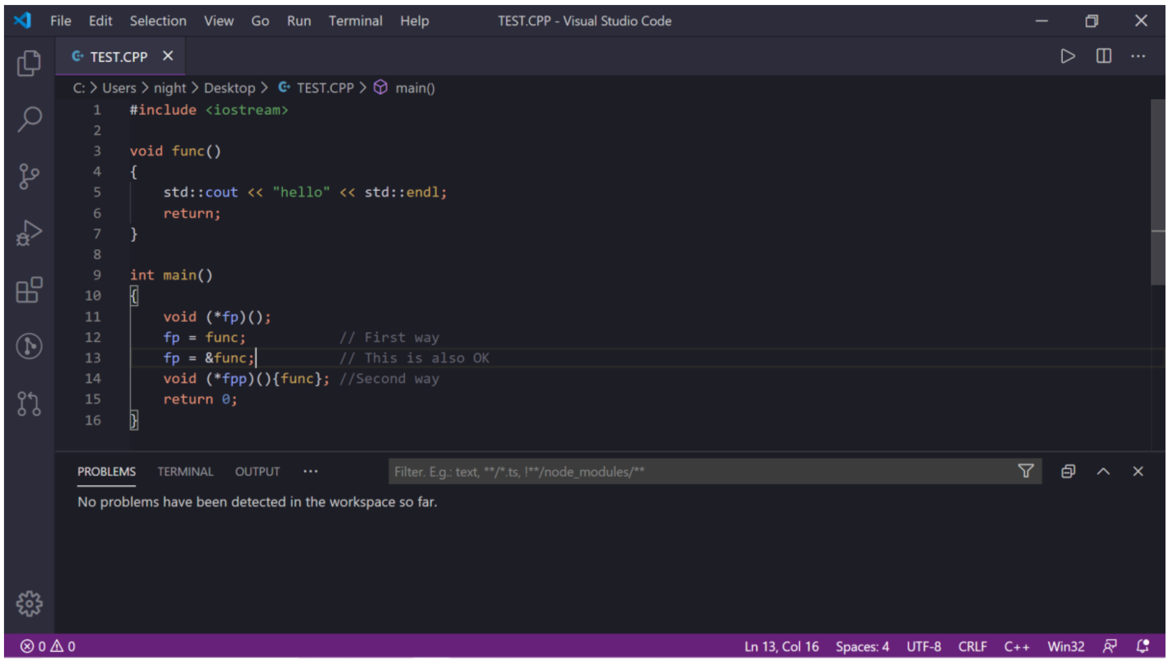
Function pointer has one of the most complicated and ugliest syntaxes in C++. If we were to define a function pointer of type void, then it would look like this:

```cpp
void (*fp)();
```

This is how we read this variable definition: fp is a pointer of a function that takes no argument and returns void(nothing). The name of the variable must be in the parentheses. We declare the arguments of that function in the other parentheses. For example, if the above function fp is a pointer to a function of type void that takes 2 integer arguments, then it would be:

```cpp
void (*fp)(int, int);
```

Assigning a function to a function pointer can be done in two ways:



```cpp
#include <iostream>

void func()
{
    std::cout << "hello" << std::endl;
    return;
}

int main()
{
    void (*fp)();
    fp = func;           // First way
    fp = &func;          // This is also OK
    void (*fpp)(){func}; //Second way
    return 0;
}
```

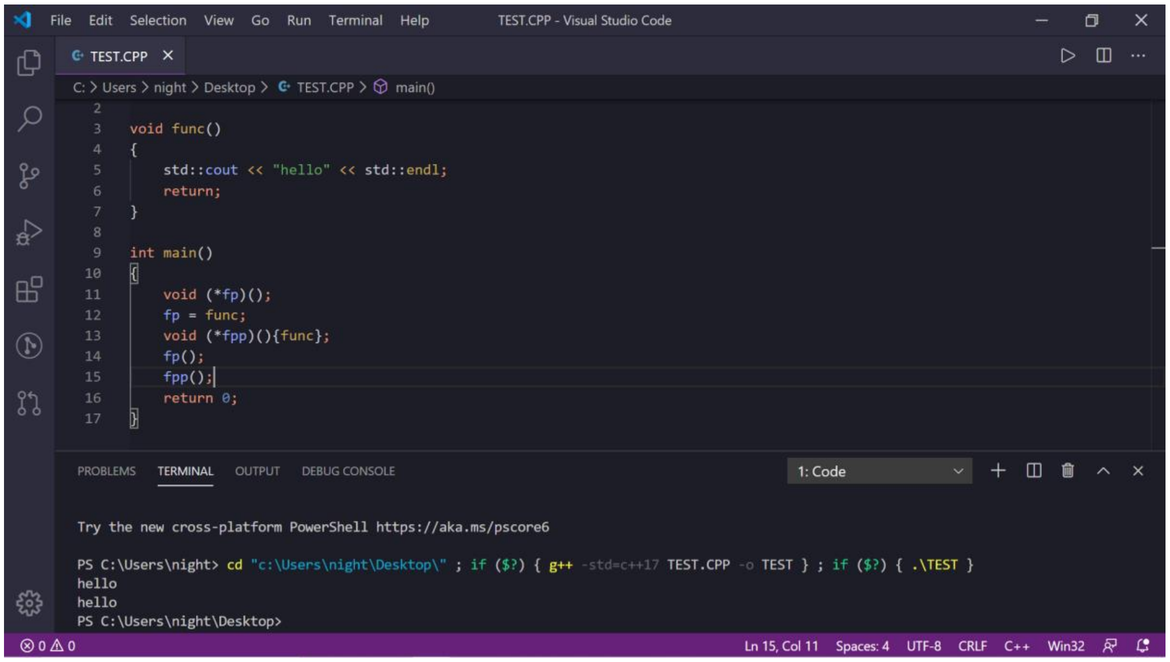
```
No problems have been detected in the workspace so far.
```

The first way is to simply assign the function pointer to that function name. In this case, fp is pointing to the function func which takes no argument and returns void. The return type and the arguments of the function that the function pointer is pointing to must matched each other. We can omit the reference symbol & since modern compilers automatically recognize this as a memory address.

The second way is to add a pair of square brackets that enclosed the function name.

CALLING FUNCTION POINTER:

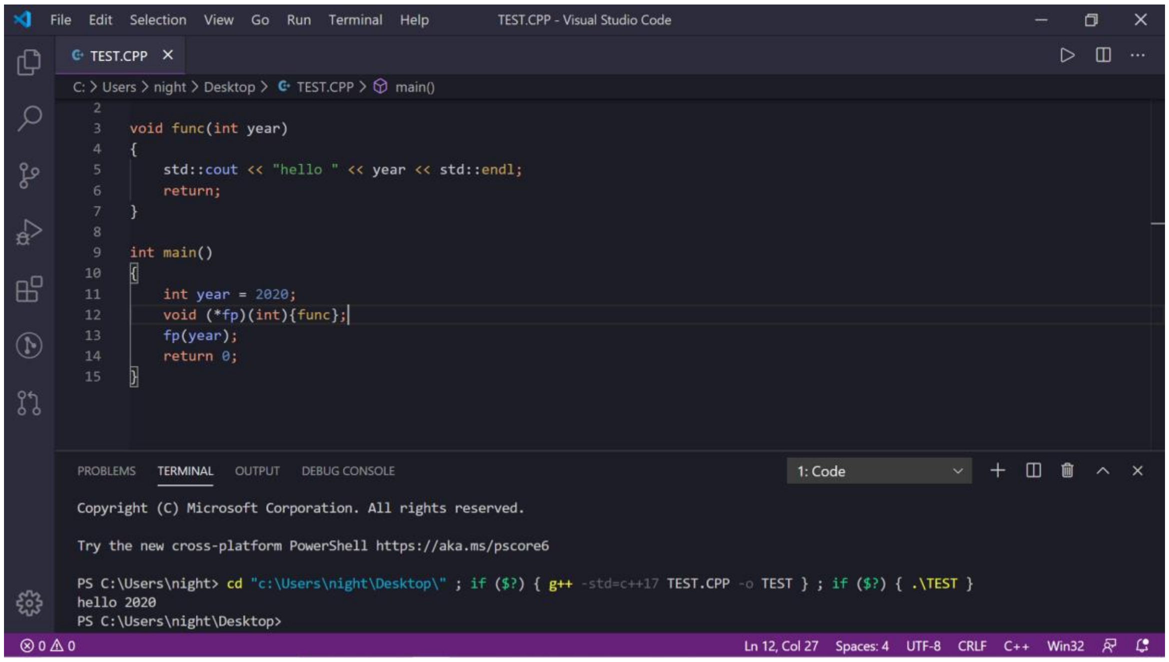
Despite the definition syntax, calling a function pointer is simple. We call it as if we are calling a normal function:



```cpp

void func()
{
    std::cout << "hello" << std::endl;
    return;
}

int main()
{
    void (*fp)();
    fp = func;
    void (*fpp)(){func};
    fp();
    fpp();
    return 0;
}
```

```
Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\night> cd "c:\Users\night\Desktop\" ; if ($?) { g++ -std=c++17 TEST.CPP -o TEST } ; if ($?) { .\TEST }
hello
hello
PS C:\Users\night\Desktop>
```

Passing arguments to the function pointer is also the same:

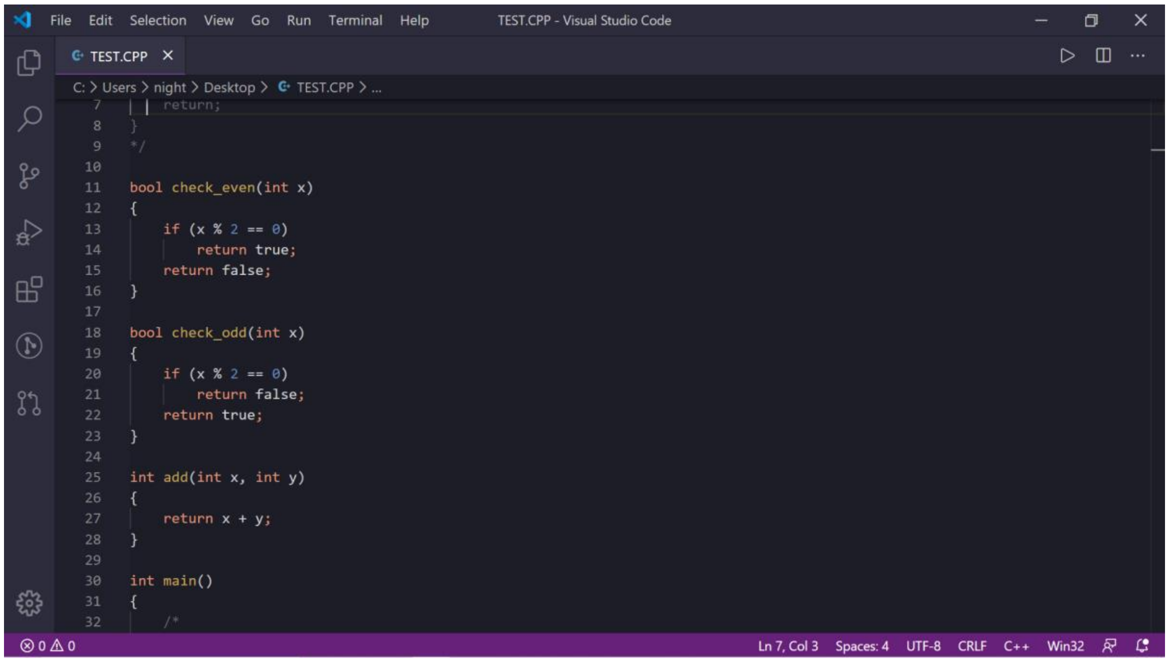


```cpp

void func(int year)
{
    std::cout << "hello " << year << std::endl;
    return;
}

int main()
{
    int year = 2020;
    void (*fp)(int){func};
    fp(year);
    return 0;
}
```

```
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\night> cd "c:\Users\night\Desktop\" ; if ($?) { g++ -std=c++17 TEST.CPP -o TEST } ; if ($?) { .\TEST }
hello 2020
PS C:\Users\night\Desktop>
```

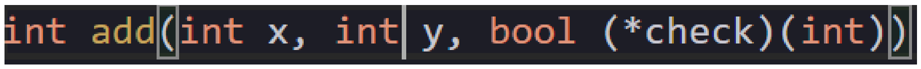
CALLBACK FUNCTIONS:

Function pointer is often used as arguments of some other functions that could control the output of those functions. This kind of function pointers is sometimes called the callback functions. To demonstrate this, consider a simple function that return the sum of two integers if the sum satisfies a condition and two functions used to check if the output is odd or even:



```cpp
    return;
}
*/

bool check_even(int x)
{
    if (x % 2 == 0)
        return true;
    return false;
}

bool check_odd(int x)
{
    if (x % 2 == 0)
        return false;
    return true;
}

int add(int x, int y)
{
    return x + y;
}

int main()
{
    /*
```

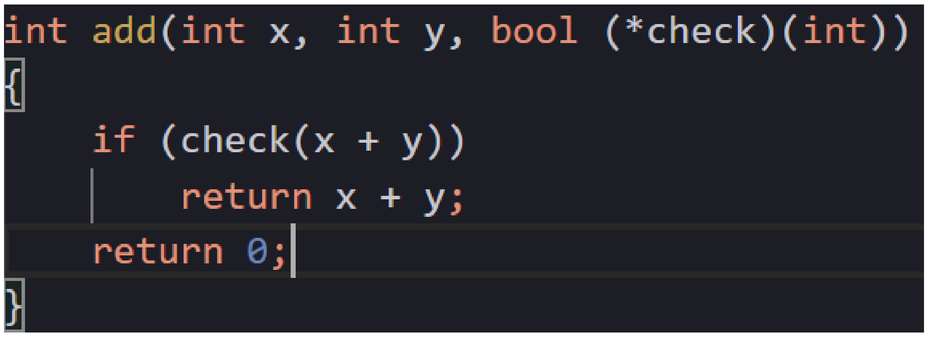
To take a callback function as the argument, we must add a function pointer to the function add:



```cpp
int add(int x, int y, bool (*check)(int))
```

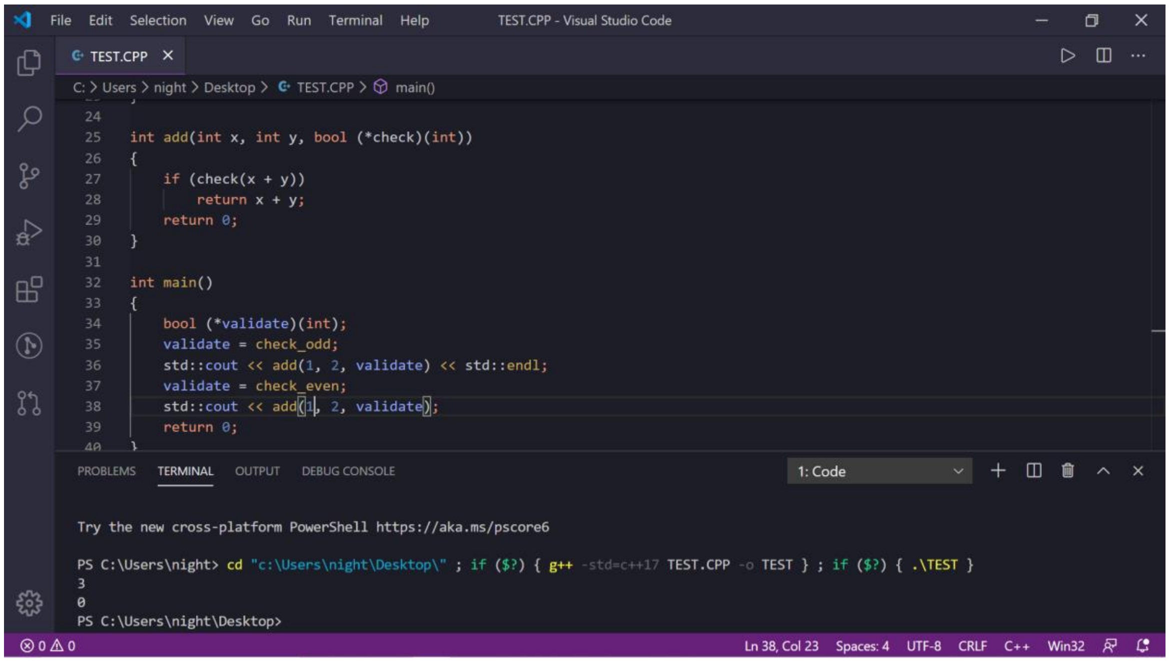
The function will now take a function pointer that takes an integer as argument and return a Boolean, this matches with the two check_even and check_odd functions that we are going to pass into.

The add function will become:



```cpp
int add(int x, int y, bool (*check)(int))
{
    if (check(x + y))
        return x + y;
    return 0;
}
```

We will define validate, which is a function pointer that takes an integer as argument and returns Boolean to store the check_even and check_odd function. Then we will try to call the function add with x = 1, y = 2, the first time using the check_odd as the callback function and the second time using check_even as the callback function:


File Edit Selection View Go Run Terminal Help
TEST.CPP - Visual Studio Code
TEST.CPP
C: > Users > night > Desktop > TEST.CPP > main()
PROBLEMS TERMINAL OUTPUT DEBUG CONSOLE
1: Code
⊗ 0 ⚠ 0
Ln 38, Col 23 Spaces: 4 UTF-8 CRLF C++ Win32

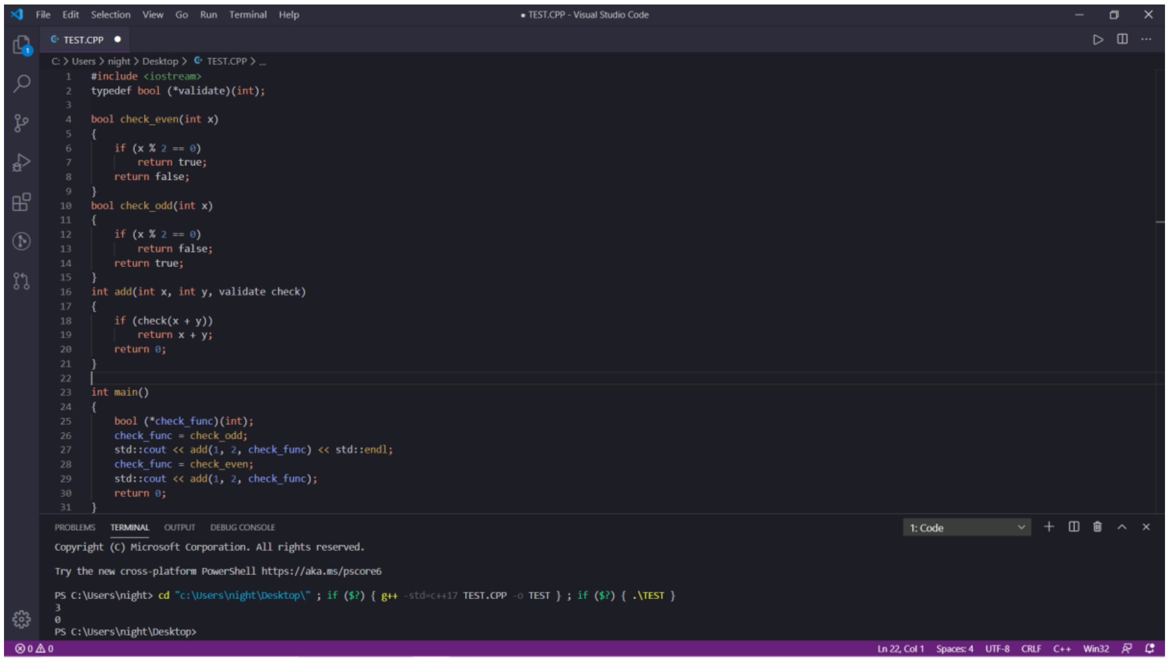
```cpp
int add(int x, int y, bool (*check)(int))
{
    if (check(x + y))
        return x + y;
    return 0;
}

int main()
{
    bool (*validate)(int);
    validate = check_odd;
    std::cout << add(1, 2, validate) << std::endl;
    validate = check_even;
    std::cout << add(1, 2, validate);
    return 0;
}
```

```
Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\night> cd "c:\Users\night\Desktop\" ; if ($?) { g++ -std=c++17 TEST.CPP -o TEST } ; if ($?) { .\TEST }
3
0
PS C:\Users\night\Desktop>
```

Since 1 + 2 = 3 is odd so the first calling successfully returned 3, but when validate is changed to check_even, the output is not even so by default, the function returned 0.

By passing function pointers to another function, we can control how the function will work. The application of the callback function is to let the user to control how a particular function works. For example, consider the sort function, by using function pointer, we can let the user choose whether the sort function will sort in ascending or descending.

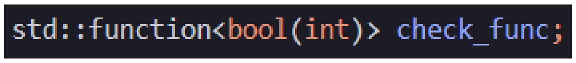
We also can make the function looks cleaner by using the typedef keyword:



```cpp
#include <iostream>
typedef bool (*validate)(int);

bool check_even(int x)
{
    if (x % 2 == 0)
        return true;
    return false;
}
bool check_odd(int x)
{
    if (x % 2 == 0)
        return false;
    return true;
}
int add(int x, int y, validate check)
{
    if (check(x + y))
        return x + y;
    return 0;
}

int main()
{
    bool (*check_func)(int);
    check_func = check_odd;
    std::cout << add(1, 2, check_func) << std::endl;
    check_func = check_even;
    std::cout << add(1, 2, check_func);
    return 0;
}
```

```
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\night> cd "c:\Users\night\Desktop\" ; if ($?) { g++ -std=c++17 TEST.CPP -o TEST } ; if ($?) { .\TEST }
3
0
PS C:\Users\night\Desktop>
```

In this case, we define globally validate as a function pointer of a function that takes an integer as argument and return Boolean, and also define check_func as a function pointer with the same argument and return type as validate. The function argument in the add function now looks much cleaner and the program output the same result. std::function in the functional library is another way to define a clean syntax for function pointers. Consider the above check_func function pointer, using std::function, it might look like this:



```cpp
std::function<bool(int)> check_func;
```

The return type and the arguments are put together, hence it is more readable. Both the typedef keyword and std::function would ignore almost all the * symbols that often confused us.

CONCLUSION:

- Function pointer is a pointer that points to a function's memory address.
- Function pointer main use are callback functions and array of function pointers.
- Function pointer's syntax is confusing, but it makes functions work flexibly.