Previous lecture

We discussed about using some of C++ basic constructs:

- Namespace, `std`, `using`
- Variables, types, and type safety
- Comments, strings
- Performing input/output (IO)
- Program organization

C++ functions

Today:

The keyword `const`

Functions:

- call by value, by reference
- default arguments
- overloading

Input from text files

const keyword

`const` is a keyword stating that the value of a variable/argument **may not** be changed.

```
int main(){
   int const ival = 3;     // a non-modifiable variable
   ival = 4;                //ERROR
}
```

Some uses of `const`:

a) to specify the size of an array:

```
  int const SIZE = 20;
   char buf[SIZE];          // 20 chars big
```

b) To declare pointers, (e.g., in pointer-arguments):

```
  char const *buf;
```

Functions

C++ functions are based on C functions.
They can have 0 or more parameters and they can return **0** or **one** value.

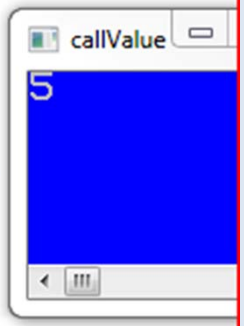
Every function must have its **prototype** declared before the function is used.
So that the compiler recognizes its **signature.**

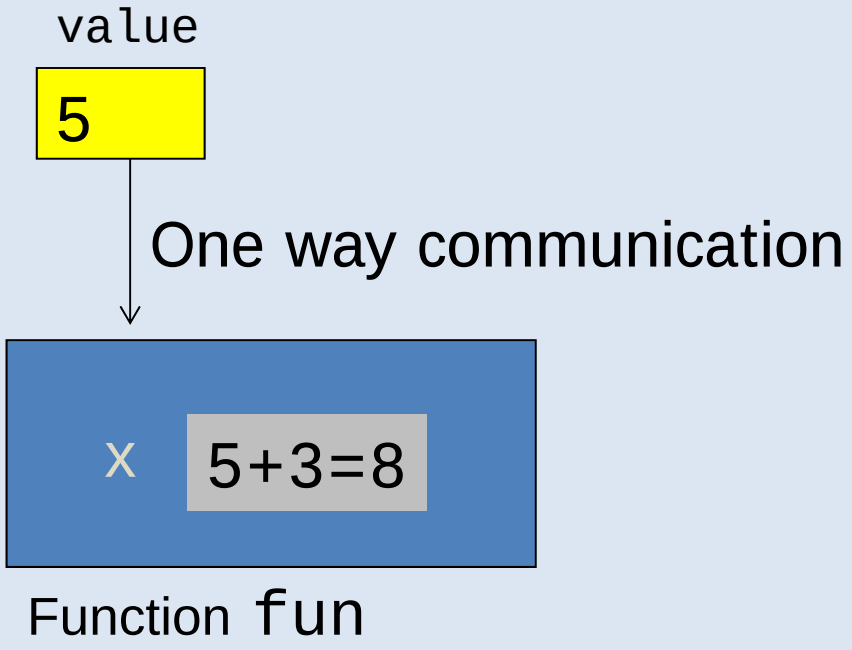
In C++ the use of `void` is optional in a function that has no parameters:

```
void displayInfo() {
   cout<<"159.234, A1 sol, S1-3015";
   cout<<"Author: Gates Bill, ID 10189";
}
```

Functions-call by value

```
1  //demo for call by value
2
3  #include<iostream>
4  using namespace std;
5
6  //adds 3 to parameter
7  void fun(int);
8
9  int main(){
10     int value=5;
11     fun(value);
12     cout << value;
13     return 0;
14 }
15
16 void fun(int x){
17     x += 3;
18     return;
19 }
20
```





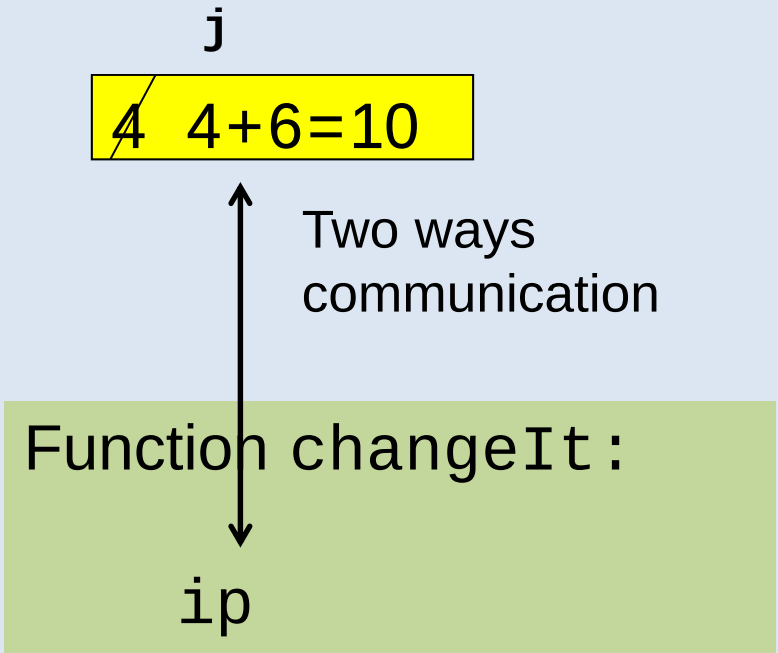
Call by pointer reference

If a **C function** wants to alter an argument, then **a pointer** to the variable must be used

```
int main() {
   int j = 4;

   changeIt(&j);
   ...
}
```

```
void changeIt(int *ip) {
   (*ip) += 6;
}
```

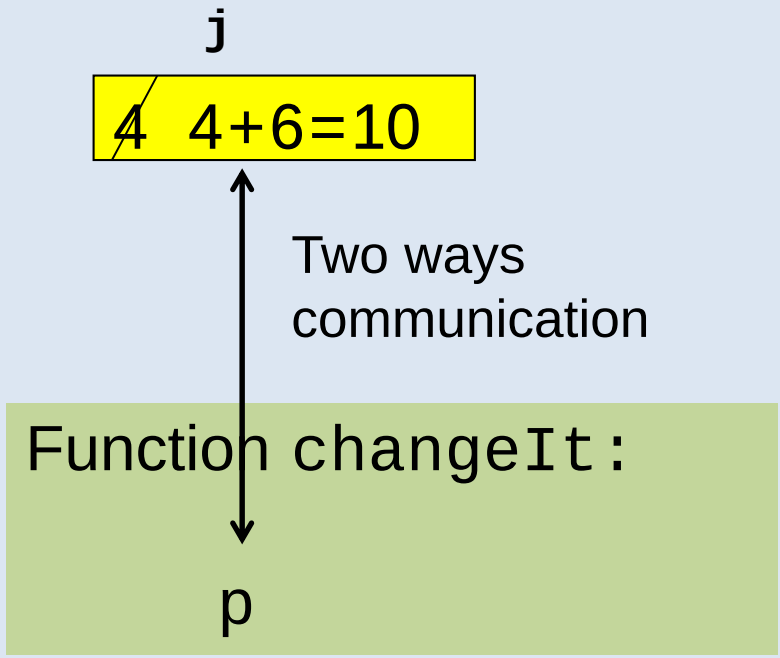


Call by reference

In C++, functions can use **call-by-reference**.

```
int main() {
   int j = 4;

   changeIt(j);
   ...
}
```




```
void changeIt(int &p) {
   p += 6;
}
```

Passing parameters

- How parameters can be passed to functions:
  - by value:

```
void myFunc(string s);
```

  - by reference:

```
void myFunc(string &s);
```

    if an arguments is not modified by the function use `const`:

```
void myFunc(const string &s);
```

  - by pointer reference/const-pointer-reference

Passing (big types) parameters by value is less efficient than passing them by reference however **efficiency should in most cases be sacrificed for clarity.**

Default function arguments

In C++ it is possible to provide `default arguments' when defining a function. These arguments are supplied by the compiler when they are not specified by the programmer.

```
void showString(string str = "Hello World!\n");

   int main() {
       showString("Here's an explicit argument.\n");

       showString();   // in fact this says:
                       // showString("Hello World!\n");
   }
```

Default function arguments

Functions may be defined with more than one default argument:

```
 // function prototype:
void twoInts(int a = 1, int b = 4);

int main(){}

// function definition
void twoInts(int a, int b){
    ...
}
```

Default function arguments

Default arguments must be known at compile-time since at that moment arguments are supplied to functions. Therefore, the default arguments must be mentioned at the function's declaration, rather than at its implementation.

```
// function prototype:
void twoInts(int a, int b);
```

← **May be in a header file**

May be in another file

```
int main(){}

// function definition
void twoInts(int a=1, int b=4){
    ...
}
```

Error

It is an error to supply default arguments in function definitions.

Default function arguments

Important:

1. The default arguments must be mentioned in the function's declaration.
2. The order of actual values maters.
3. Arguments can be defaulted from last to first.

```
void twoInts(int a = 1, int b = 4);

int main(){
   twoInts();              // actual values:  1, 4
   twoInts(20);            // actual values: 20, 4
   twoInts(20, 5);         // actual values: 20, 5
   twoInts( , 5);          // Illegal
}
```

Function overloading

In C++ it is possible to define functions having identical names but performing different actions. The functions must differ in their parameter lists.

```
void show(int val) {cout<<"Integer: "<< val; }
void show(string val) {cout<<"String: "<< val; }
void show(int val, float num){cout<<"Two numbers: "
                                  << val<<", "<<num;}
```

```
int main() {
   show(12, 3.1415);
   show(12);
   show("Hello World!\n");
}
```

- Do not use function overloading for functions doing conceptually different tasks.
- C++ does not allow identically named functions to differ only in their return values.

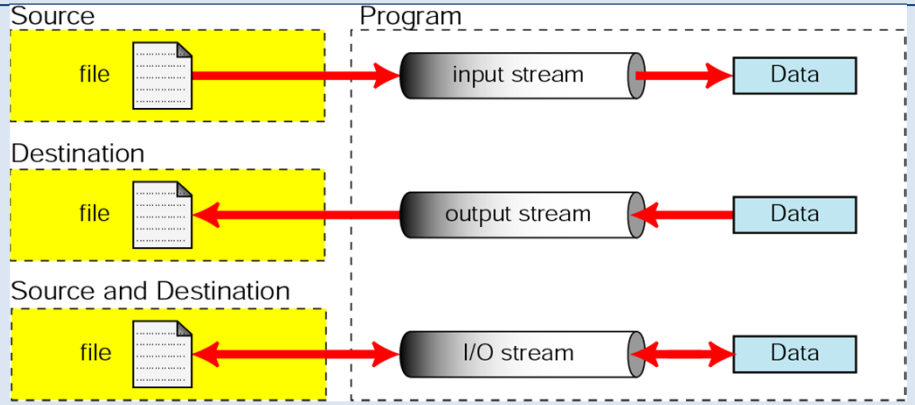
File streams

- Files are used to store information on a permanent basis
- To do file input/output **<fstream>** needs to be included
- An input or output stream needs to be declared:

```
ifstream inF; // input file stream
ofstream outF; // output file stream
```

- Before C++ program can read from or write to a file stream, the stream needs to **be connected** to a file. This is referred to as *opening* the file:

```
inF.open("myfile.dat"); // opening file
```



Reading from files

C-way

```
#include <stdio.h>
FILE *f;
f = fopen(“data.txt”, “r”);
fscanf(f,"%d %d",&n1,&n2);
fclose(f);
```

C++ way

```
#include <fstream>
..
ifstream inFile;
inFile.open(“data.txt”);
//to check file was found-to be added here
inFile >> n1 >> n2;
inFile.close();
```

Writing data on files

C-way

```
#include <stdio.h>
FILE *f;
f = fopen(“result.txt”, “w”);
//..
fprintf(f,"%d %d",n1,n2);
fclose(f);
```

C++ way

```
#include <fstream>
using namespace std;
ofstream outFile;
outFile.open(“result.txt”);
//check the file was created
outFile << n1 << n2;
outFile.close();
```

Caution!

Always check for the existence of the file before using it!

```
if( ! myFile ){ //no file to work with
   cout << “Failed to find/create file” << endl;
   return 1;    //something was wrong
}
```

When the file is in a different directory, e.g

```
“A:\mystuff\input.txt”
```

we should use:

```
ifstream myFile(“A:\\mystuff\\input.txt”);
```

Open a file- the short way

```
#include<iostream>
#include<string>
#include<fstream>
#include<cstdlib>
using namespace std;
int main(){
    ifstream inFile("data.txt");
   if(!inFile){
      cout<<"Open error\n";
      return EXIT_FAILURE;
   }
    string word;
    do{
      inFile >> word;
      cout <<word<<endl;
    }while(!inFile.eof());
    return EXIT_SUCCESS;
}
```

Streams as parameters

- Streams (input, output) can be passed as parameters
- **Streams are always called by reference**

```
struct Pair{int first; int sec;};

Pair func(istream &input);
int main(){
   Pair val= func(cin);
   cout <<val.first <<" and "<<val.sec<<endl;
   ifstream inF("data.txt");
   if (inF){
     val=func(inF);
     cout <<val.first <<" and "<<val.sec<<endl;
   }
}
Pair func(istream &input){
  Pair nums;
  cout<<"Enter two integers:  ";
  input >> nums.first >> nums.sec;
  return nums;
}
```

Input data

cin **#include <iostream>**	**myF (input file)** **#include<ifstream>**	**Doing:**
`cin >>ch`	`myF >> ch`	Skip white spaces
`cin.get(ch)`	`myF.get(ch)`	Reads every char into `ch`

Output data

cout **#include<iostream>**	**myF (output file)** **#include<ofstream>**	**Doing:**
`cout <<value`	`myF<<value`	Write out `value (char, int,` C/C++ strings …)
`cout.put(ch)`	`myF.put(ch)`	Write out `char ch`

Be careful with strings

cin **#include <iostream>**	**myF (input file)** **#include<fstream>**	**Doing:**
`cin >> ch`	`myF >> ch`	Skip white spaces, ch can be char or a single word
`cin.get(ch)`	`myF.get(ch)`	Reads every char into `ch`
`getline(cin, str)`	`getline(myF,str)`	Reads a line of text (removes '\n') into **C++ string** `str`
`cin.getline(str,60,'#)` `cin.getline(str,60)`	`myF.getline(str,60,'#)` `myF.getline(str,60)`	**Read C-strings** (removes terminal. ch. from stream), appends '\0'.

Summary

- Functions
  - can have default parameters,
  - can use call by value or call by reference
  - can be overloaded
- Working with text files
- Streams as parameters
- `const` keyword

Next: Classes