File Management in C

C File Handling:

C files I/O functions handle data on a secondary storage device, such as a hard disk.

C can handle files as Stream-oriented data (Text) files, and System oriented data (Binary) files.

Stream-oriented data files	The data is stored in the same manner as it appears on the screen. The I/O operations like buffering, data conversions, etc. take place automatically.
System-oriented data files	System-oriented data files are more closely associated with the OS and data stored in memory without converting into text format.

C File Operations

Five major operations can be performed on file are:

- Creation of a new file.
- Opening an existing file.
- Reading data from a file.
- Writing data in a file.
- Closing a file.

Steps for Processing a File

- Declare a file pointer variable.
- Open a file using fopen() function.
- Process the file using the suitable function.
- Close the file using fclose() function.
- To handling files in C, file *input/output* functions available in the *stdio* library are:

Function	Uses/Purpose
fopen	Opens a file.
fclose	Closes a file.
getc	Reads a character from a file
putc	Writes a character to a file
getw	Read integer
putw	Write an integer
fprintf	Prints formatted output to a file
fscanf	Reads formatted input from a file
fgets	Read string of characters from a file
fputs	Write string of characters to file
feof	Detects end-of-file marker in a file

C fopen

C fopen is a C library function used to open an existing file or create a new file.

S.A

The basic format of fopen is:

Syntax:

```
FILE *fopen( const char * filePath, const char * mode );
```

Parameters
Return Value

Parameters

- filePath: The first argument is a pointer to a string containing the name of the file to be opened.
- mode: The second argument is an access mode.

C *fopen()* access mode can be one of the following values:

Mode	Description
R	Opens an existing text file.
W	Opens a text file for writing if the file doesn't exist then a new file is created.
A	Opens a text file for appending(writing at the end of existing file) and create the file if it does not exist.
r+	Opens a text file for reading and writing.
w+	Open for reading and writing and create the file if it does not exist. If the file exists then make it blank.
a+	Open for reading and appending and create the file if it does not exist. The reading will start from the beginning, but writing can only be appended.

Return Value

C fopen function returns *NULL* in case of a failure and returns a *FILE stream pointer* on success.

Example:

S.A

```c
#include<stdio.h>

int main()

{

    FILE *fp;

    fp = fopen("fileName.txt","w");

    return 0;

}
```

- The above example will create a file called *fileName.txt*.
- The *w* means that the file is being opened for writing, and if the file does not exist then the new file will be created.

fclose function

fclose() function is C library function and it's used to releases the memory stream, opened by *fopen()* function.

The basic format of fclose is:

Syntax:

```c
int fclose( FILE * stream );
```

Return Value

C fclose returns *EOF* in case of failure and returns *0* on success.

Example:

```c
#include<stdio.h>


int main()

{

    FILE *fp;

    fp = fopen("fileName.txt","w");
```

S.A

```
    fprintf(fp, "%s", "Sample Texts");

    fclose(fp);

    return 0;

}
```

- The above example will create a file called *fileName.txt*.
- The *w* means that the file is being opened for writing, and if the file does not exist then the new file will be created.
- The *fprintf* function writes *Sample Texts* text to the file.
- The *fclose* function closes the file and releases the memory stream.

getc function

getc() function is C library function, and it's used to read a character from a file that has been opened in read mode by *fopen()* function.

Syntax:

```
int getc( FILE * stream );
```

Return Value

- getc() function returns next requested object from the stream on success.
- Character values are returned as an unsigned char cast to an int or EOF on end of file or error.
- The function feof() and ferror() to distinguish between end-of-file and error must be used.

Example:

```
#include<stdio.h>


int main()

{
```

```
  FILE *fp = fopen("fileName.txt", "r");
  int ch = getc(fp);
  while (ch != EOF)
  {
    /* To display the contents of the file on the screen */    putchar(ch);
    ch = getc(fp);
  }
  if (feof(fp))
     printf("\n Reached the end of file.");
  else
     printf("\n Something gone wrong.");
  fclose(fp);

  getchar();
  return 0;
}
```

Putc function:

putc() function is C library function, and it's used to write a character to the file. This function is used for writing a single character in a stream along with that it moves forward the indicator's position.

```
int putc( int c, FILE * stream );
```

Example:

```
int main (void)
```

S.A

```
{
  FILE * fileName;
  char ch;
  fileName = fopen("anything.txt","wt");
  for (ch = 'D' ; ch <= 'S' ; ch++) {
    putc (ch , fileName);
    }
  fclose (fileName);
  return 0;
}
```

getw function

C getw function is used to read an integer from a file that has been opened in read mode. It is a file handling function, which is used for reading integer values.

Syntax:

```
int getw( FILE * stream );
```

putw function:

C putw function is used to write an integer to the file.

Syntax:

```
int putw( int c, FILE * stream );
```

Example:

```
int main (void)
{
   FILE *fileName;
```

S.A

```
    int i=2, j=3, k=4, n;
    fileName = fopen ("anything.c","w");
    putw(i, fileName);
    putw(j, fileName);
    putw(k, fileName);
    fclose(fileName);

    fileName = fopen ("test.c","r");
    while(getw(fileName)! = EOF)
    {
       n= getw(fileName);
       printf("Value is %d \t: ", n);
    }
    fclose(fp);
    return 0;
}
```

fprintf function:

C fprintf function pass arguments according to the specified format to the file indicated by the stream. This function is implemented in file related programs for writing formatted data in any file.

Syntax:

```
int fprintf(FILE *stream, const char *format, ...)
```

S.A

Example:

```
int main (void)

{

   FILE *fileName;

   fileName = fopen("anything.txt","r");

   fprintf(fileName, "%s %s %d", "Welcome", "to",  2018);

   fclose(fileName);

   return(0);

}
```

fscanf function:

C fscanf function reads formatted input from a file. This function is implemented in file related programs for reading formatted data from any file that is specified in the program.

Syntax:

```
int fscanf(FILE *stream, const char *format, ...)
```

Its return the number of variables that are assigned values, or EOF if no assignments could be made.

Example:

```
int main()

{

    char str1[10], str2[10];

    int yr;

    FILE* fileName;

    fileName = fopen("anything.txt", "w+");
```

S.A

```
    fputs("Welcome to", fileName);
    rewind(fileName);
    fscanf(fileName, "%s %s %d", str1, str2, &yr);
    printf("---------------------------------------------- \n");
    printf("1st word %s \t", str1);
    printf("2nd word  %s \t", str2);
    printf("Year-Name  %d \t", yr);
    fclose(fileName);
    return (0);
}
```

fgets function:

C fgets function is implemented in file related programs for reading strings from any particular file. It gets the strings 1 line each time.

Syntax:

```
char *fgets(char *str, int n, FILE *stream)
```

Example:

```
void main(void)
{
    FILE* fileName;
    char ch[100];
    fileName = fopen("anything.txt", "r");
```

S.A

```
    printf("%s", fgets(ch, 50, fileName));

    fclose(fileName);

}
```

- On success, the function returns the same str parameter
- C fgets function returns a NULL pointer in case of a failure.

fputs function:

C fputs function is implemented in file related programs for writing string to any particular file.

Syntax:

```
int fputs(const char *str, FILE *stream)
```

Example:

```
#include<stdio.h>

int main()

{

FILE *fp;

fp = fopen("fileName.txt","w");


fputs("This is a sample text file.", fp);

fputs("This file contains some sample text data.", fp);

fclose(fp);

return 0;

}
```

S.A

In this function returns non-negative value, otherwise returns EOF on error.

feof function:

C feof function is used to determine if the end of the file (stream), specified has been reached or not. This function keeps on searching the end of file (eof) in your file program.

Syntax:

```
int feof(FILE *stream)
```

Here is a program showing the use of feof().

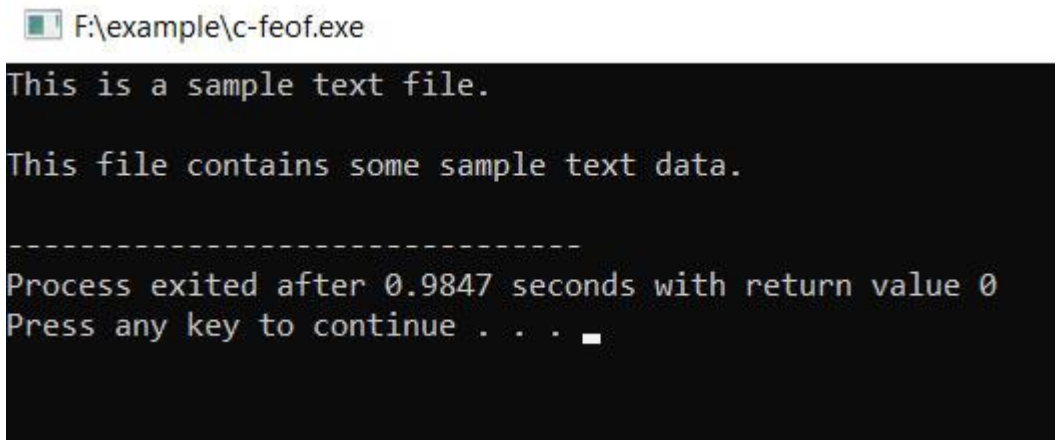
Example:

```
#include<stdio.h>

int main()

{

FILE *filee = NULL;

char buf[50];

filee = fopen("infor.txt","r");

if(filee)

    {

         while(!feof(filee))

         {

             fgets(buf, sizeof(buf), filee);

             puts(buf);

         }

         fclose(filee);
```

S.A

```
    }
     return 0;
}
```

Output:



C feof function returns true in case end of file is reached, otherwise it's return false.
Explanation:

1. It first tries to open a text file infor.txt as read-only mode.
2. Then as the file gets opened successfully to read, it initiates the while loop.
3. The iteration continues until all the statement/lines of your text file get to read as well as displayed. Lastly, you have to close the file.

INTRODUCTION TO C++

What is C++?

C++ is a cross-platform language that can be used to create high-performance applications.

C++ was developed by Bjarne Stroustrup, as an extension to the C language.

C++ gives programmers a high level of control over system resources and memory.

S.A

The language was updated 3 major times in 2011, 2014, and 2017 to C++11, C++14, and C++17.

Why Use C++

C++ is one of the world's most popular programming languages.

C++ can be found in today's operating systems, Graphical User Interfaces, and embedded systems.

C++ is an object-oriented programming language which gives a clear structure to programs and allows code to be reused, lowering development costs.

C++ is portable and can be used to develop applications that can be adapted to multiple platforms.

C++ is fun and easy to learn!

As C++ is close to C# and Java, it makes it easy for programmers to switch to C++ or vice versa

C++ What is OOP?

OOP stands for Object-Oriented Programming.

Procedural programming is about writing procedures or functions that perform operations on the data, while object-oriented programming is about creating objects that contain both data and functions.

Object-oriented programming has several advantages over procedural programming:

- OOP is faster and easier to execute
- OOP provides a clear structure for the programs
- OOP helps to keep the C++ code DRY "Don't Repeat Yourself", and makes the code easier to maintain, modify and debug
- OOP makes it possible to create full reusable applications with less code and shorter development time

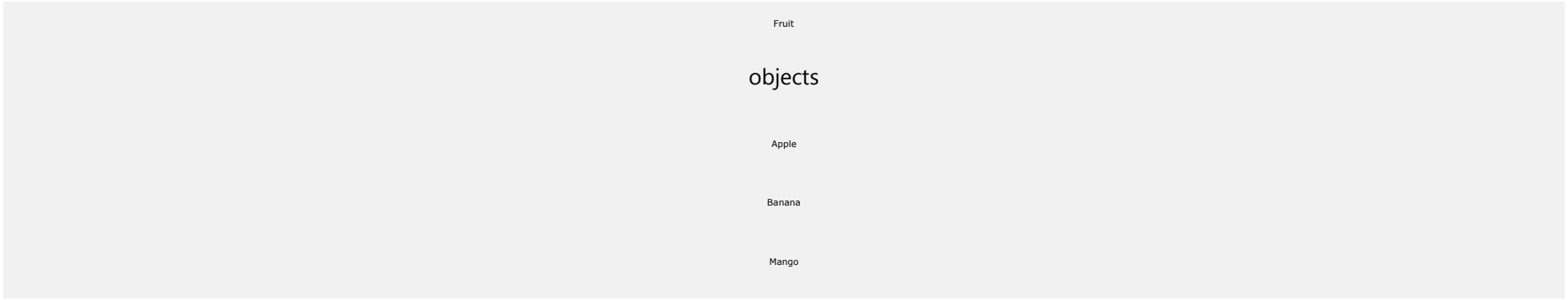
C++ What are Classes and Objects?

Classes and objects are the two main aspects of object-oriented programming.

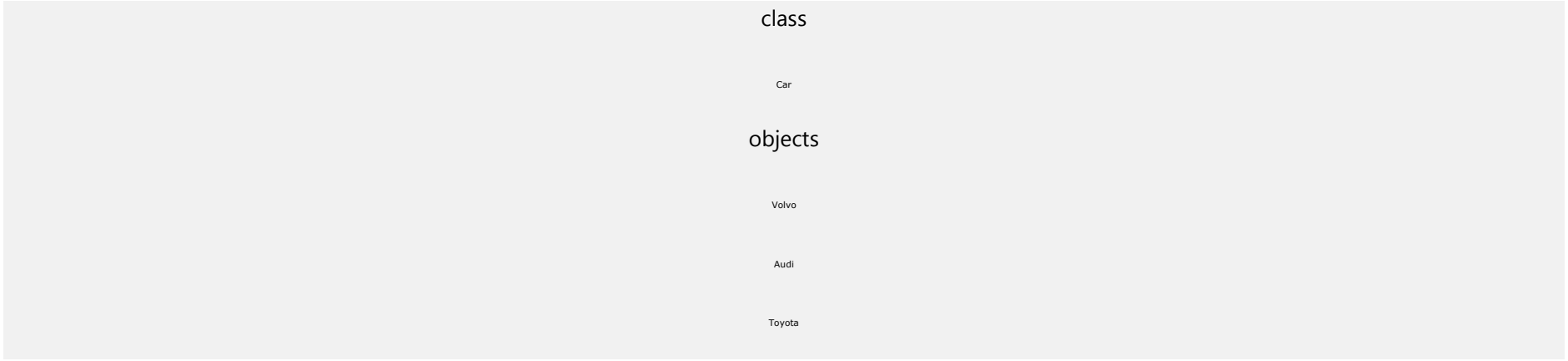
Look at the following illustration to see the difference between class and objects:

class

S.A



Another example:



So, a class is a template for objects, and an object is an instance of a class.

When the individual objects are created, they inherit all the variables and functions from the class.

C++ Classes/Objects

C++ is an object-oriented programming language.

Everything in C++ is associated with classes and objects, along with its attributes and methods. For example: in real life, a car is an **object**. The car has **attributes**, such as weight and color, and **methods**, such as drive and brake.

Attributes and methods are basically **variables** and **functions** that belongs to the class. These are often referred to as "class members".

S.A

A class is a user-defined data type that we can use in our program, and it works as an object constructor, or a "blueprint" for creating objects.

Create a Class

To create a class, use the `class` keyword:

Example

Create a class called "`MyClass`":

```
class MyClass {       // The class
  public:             // Access specifier
    int myNum;        // Attribute (int variable)
    string myString;  // Attribute (string variable)
};
```

Example explained

- The `class` keyword is used to create a class called `MyClass`.
- The `public` keyword is an **access specifier**, which specifies that members (attributes and methods) of the class are accessible from outside the class. You will learn more about access specifiers later.
- Inside the class, there is an integer variable `myNum` and a string variable `myString`. When variables are declared within a class, they are called **attributes**.
- At last, end the class definition with a semicolon `;`.

Create an Object

In C++, an object is created from a class. We have already created the class named `MyClass`, so now we can use this to create objects.

To create an object of `MyClass`, specify the class name, followed by the object name.

To access the class attributes (`myNum` and `myString`), use the dot syntax (`.`) on the object:

Example

Create an object called "`myObj`" and access the attributes:

```
class MyClass {       // The class
  public:             // Access specifier
    int myNum;        // Attribute (int variable)
    string myString;  // Attribute (string variable)
};

int main() {
  MyClass myObj;  // Create an object of MyClass

  // Access attributes and set values
  myObj.myNum = 15;
  myObj.myString = "Some text";

  // Print attribute values
  cout << myObj.myNum << "\n";
  cout << myObj.myString;
  return 0;
```

S.A

```
}
```

Multiple Objects

You can create multiple objects of one class:

Example

```
// Create a Car class with some attributes
class Car {
  public:
    string brand;
    string model;
    int year;
};

int main() {
  // Create an object of Car
  Car carObj1;
s  carObj1.brand = "BMW";
  carObj1.model = "X5";
  carObj1.year = 1999;

  // Create another object of Car
  Car carObj2;
  carObj2.brand = "Ford";
  carObj2.model = "Mustang";
  carObj2.year = 1969;

  // Print attribute values
  cout << carObj1.brand << " " << carObj1.model << " " << carObj1.year << "\n";
  cout << carObj2.brand << " " << carObj2.model << " " << carObj2.year << "\n";
  return 0;
}
```

C++ Constructors

Constructors

A constructor in C++ is a **special method** that is automatically called when an object of a class is created.

To create a constructor, use the same name as the class, followed by parentheses `()`:

Example

```
class MyClass {     // The class
  public:           // Access specifier
    MyClass() {     // Constructor
      cout << "Hello World!";
    }
};

int main() {
  MyClass myObj;    // Create an object of MyClass (this will call the constructor)
  return 0;
}
```

Note: The constructor has the same name as the class, it is always `public`, and it does not have any return value.

S.A

Constructor Parameters

Constructors can also take parameters (just like regular functions), which can be useful for setting initial values for attributes.

The following class have `brand`, `model` and `year` attributes, and a constructor with different parameters. Inside the constructor we set the attributes equal to the constructor parameters (`brand=x`, etc). When we call the constructor (by creating an object of the class), we pass parameters to the constructor, which will set the value of the corresponding attributes to the same:

Example

```
class Car {        // The class
  public:          // Access specifier
    string brand;  // Attribute
    string model;  // Attribute
    int year;      // Attribute
    Car(string x, string y, int z) { // Constructor with parameters
      brand = x;
      model = y;
      year = z;
    }
};

int main() {
  // Create Car objects and call the constructor with different values
  Car carObj1("BMW", "X5", 1999);
  Car carObj2("Ford", "Mustang", 1969);

  // Print values
  cout << carObj1.brand << " " << carObj1.model << " " << carObj1.year << "\n";
  cout << carObj2.brand << " " << carObj2.model << " " << carObj2.year << "\n";
  return 0;
}
```

Just like functions, constructors can also be defined outside the class. First, declare the constructor inside the class, and then define it outside of the class by specifying the name of the class, followed by the scope resolution `::` operator, followed by the name of the constructor (which is the same as the class):

Example

```
class Car {        // The class
  public:          // Access specifier
    string brand;  // Attribute
    string model;  // Attribute
    int year;      // Attribute
    Car(string x, string y, int z); // Constructor declaration
};

// Constructor definition outside the class
Car::Car(string x, string y, int z) {
  brand = x;
  model = y;
  year = z;
}

int main() {
  // Create Car objects and call the constructor with different values
  Car carObj1("BMW", "X5", 1999);
  Car carObj2("Ford", "Mustang", 1969);

  // Print values
  cout << carObj1.brand << " " << carObj1.model << " " << carObj1.year << "\n";
  cout << carObj2.brand << " " << carObj2.model << " " << carObj2.year << "\n";
  return 0;  }
}
```

S.A

C++ Access Specifiers

Access Specifiers

By now, you are quite familiar with the `public` keyword that appears in all of our class examples:

Example

```
class MyClass {  // The class
  public:        // Access specifier
    // class members goes here
};
```

The `public` keyword is an **access specifier.** Access specifiers define how the members (attributes and methods) of a class can be accessed. In the example above, the members are `public` - which means that they can be accessed and modified from outside the code.

However, what if we want members to be private and hidden from the outside world?

In C++, there are three access specifiers:

- `public` - members are accessible from outside the class
- `private` - members cannot be accessed (or viewed) from outside the class
- `protected` - members cannot be accessed from outside the class, however, they can be accessed in inherited classes. You will learn more about Inheritance later.

In the following example, we demonstrate the differences between `public` and `private` members:

Example

```
class MyClass {
  public:    // Public access specifier
    int x;   // Public attribute
  private:   // Private access specifier
    int y;   // Private attribute
};

int main() {
  MyClass myObj;
  myObj.x = 25;  // Allowed (public)
  myObj.y = 50;  // Not allowed (private)
  return 0;
}
```

If you try to access a private member, an error occurs:

```
error: y is private
```

Note: It is possible to access private members of a class using a public method inside the same class. See the next chapter (Encapsulation) on how to do this.

Tip: It is considered good practice to declare your class attributes as private (as often as you can). This will reduce the possibility of yourself (or others) to mess up the code. This is also the main ingredient of the Encapsulation concept, which you will learn more about in the next chapter.

Note: By default, all members of a class are `private` if you don't specify an access specifier

S.A

Example

```
class MyClass {
  int x;   // Private attribute
  int y;   // Private attribute
};
```

C++ Functions

A function is a block of code which only runs when it is called.

You can pass data, known as parameters, into a function.

Functions are used to perform certain actions, and they are important for reusing code: Define the code once, and use it many times.

Create a Function

C++ provides some pre-defined functions, such as `main()`, which is used to execute code. But you can also create your own functions to perform certain actions.

To create (often referred to as *declare*) a function, specify the name of the function, followed by parentheses **()**:

Syntax

```
void myFunction() {
  // code to be executed
}
```

Example Explained

- `myFunction()` is the name of the function
- `void` means that the function does not have a return value. You will learn more about return values later in the next chapter
- inside the function (the body), add code that defines what the function should do

Call a Function

Declared functions are not executed immediately. They are "saved for later use", and will be executed later, when they are called.

To call a function, write the function's name followed by two parentheses `()` and a semicolon `;`

In the following example, `myFunction()` is used to print a text (the action), when it is called:

S.A

Example

Inside `main`, call `myFunction()`:

```
// Create a function
void myFunction() {
  cout << "I just got executed!";
}

int main() {
  myFunction(); // call the function
  return 0;
}

// Outputs "I just got executed!"
```

A function can be called multiple times:

Example

```
void myFunction() {
  cout << "I just got executed!\n";
}

int main() {
  myFunction();
  myFunction();
  myFunction();
  return 0;
}

// I just got executed!
// I just got executed!
// I just got executed!
```

Function Declaration and Definition

A C++ function consist of two parts:

- **Declaration:** the function's name, return type, and parameters (if any)
- **Definition:** the body of the function (code to be executed)

```
void myFunction() { // declaration
  // the body of the function (definition)
}
```

Note: If a user-defined function, such as `myFunction()` is declared after the `main()` function, **an error will occur**. It is because C++ works from top to bottom; which means that if the function is not declared above `main()`, the program is unaware of it:

Example

```
int main() {
  myFunction();
  return 0;
}

void myFunction() {
  cout << "I just got executed!";
}
```

S.A

```
// Error
```

However, it is possible to separate the declaration and the definition of the function - for code optimization.

You will often see C++ programs that have function declaration above `main()`, and function definition below `main()`. This will make the code better organized and easier to read:

Example

```
// Function declaration
void myFunction();

// The main method
int main() {
  myFunction();  // call the function
  return 0;
}

// Function definition
void myFunction() {
  cout << "I just got executed!";
}
```

C++ Function Parameters

Parameters and Arguments

Information can be passed to functions as a parameter. Parameters act as variables inside the function.

Parameters are specified after the function name, inside the parentheses. You can add as many parameters as you want, just separate them with a comma:

Syntax

```
void functionName(parameter1, parameter2, parameter3) {
  // code to be executed
}
```

The following example has a function that takes a `string` called **fname** as parameter. When the function is called, we pass along a first name, which is used inside the function to print the full name:

Example

```
void myFunction(string fname) {
  cout << fname << " Refsnes\n";
}

int main() {
  myFunction("Liam");
  myFunction("Jenny");
  myFunction("Anja");
  return 0;
}

// Liam Refsnes
// Jenny Refsnes
// Anja Refsnes
```

S.A

C++ Function Overloading

Function Overloading

With **function overloading**, multiple functions can have the same name with different parameters:

Example

```
int myFunction(int x)
float myFunction(float x)
double myFunction(double x, double y)
```

Consider the following example, which have two functions that add numbers of different type:

Example

```
int plusFuncInt(int x, int y) {
  return x + y;
}

double plusFuncDouble(double x, double y) {
  return x + y;
}

int main() {
  int myNum1 = plusFuncInt(8, 5);
  double myNum2 = plusFuncDouble(4.3, 6.26);
  cout << "Int: " << myNum1 << "\n";
  cout << "Double: " << myNum2;
  return 0;
}
```

Instead of defining two functions that should do the same thing, it is better to overload one.

In the example below, we overload the `plusFunc` function to work for both `int` and `double`:

Example

```
int plusFunc(int x, int y) {
  return x + y;
}

double plusFunc(double x, double y) {
  return x + y;
}

int main() {
  int myNum1 = plusFunc(8, 5);
  double myNum2 = plusFunc(4.3, 6.26);
  cout << "Int: " << myNum1 << "\n";
  cout << "Double: " << myNum2;
  return 0;
}
```

Note: Multiple functions can have the same name as long as the number and/or type of parameters are different.

S.A

C++ Variables

Variables are containers for storing data values.

In C++, there are different **types** of variables (defined with different keywords), for example:

- `int` - stores integers (whole numbers), without decimals, such as 123 or -123
- `double` - stores floating point numbers, with decimals, such as 19.99 or -19.99
- `char` - stores single characters, such as 'a' or 'B'. Char values are surrounded by single quotes
- `string` - stores text, such as "Hello World". String values are surrounded by double quotes
- `bool` - stores values with two states: true or false

Declaring (Creating) Variables

To create a variable, you must specify the type and assign it a value:

Syntax

```
type variable = value;
```

Where *type* is one of C++ types (such as `int`), and *variable* is the name of the variable (such as **x** or **myName**). The **equal sign** is used to assign values to the variable.

To create a variable that should store a number, look at the following example:

Example

Create a variable called **myNum** of type `int` and assign it the value **15**:

```
int myNum = 15;
cout << myNum;
```

You can also declare a variable without assigning the value, and assign the value later:

Example

```
int myNum;
myNum = 15;
cout << myNum;
```

Note that if you assign a new value to an existing variable, it will overwrite the previous value:

S.A

Example

```
int myNum = 15;  // myNum is 15
myNum = 10;  // Now myNum is 10
cout << myNum;  // Outputs 10
```

Other Types

A demonstration of other data types:

Example

```
int myNum = 5;               // Integer (whole number without decimals)
double myFloatNum = 5.99;    // Floating point number (with decimals)
char myLetter = 'D';         // Character
string myText = "Hello";     // String (text)
bool myBoolean = true;       // Boolean (true or false)
```

Display Variables

The `cout` object is used together with the `<<` operator to display variables.

To combine both text and a variable, separate them with the `<<` operator:

Example

```
int myAge = 35;
cout << "I am " << myAge << " years old.";
```

output:

```
I am 35 years old.
```

Add Variables Together

To add a variable to another variable, you can use the `+` operator:

Example

```
int x = 5;
int y = 6;
int sum = x + y;
cout << sum;
```

output:

```
11
```

S.A