Fundamental Concepts: The following is a summary of the fundamental concepts of object-oriented programming in C++.

1. Class: A data type used to represent a new structured data type. A class is also regarded as a template through which individual objects are created. Example syntax:

```cpp
class Student
{
public:
    Student();
    string getName();
    void setName();
private:
    string name;
    float gpa;
    int age;
}
```

2. Object: A variable declared with a new class type. An object is an instance of a class type.

```cpp
Student st;
```

Variable st above is an instance or object of class Student. It has a set of attributes and functions declared in class Student.

3. Constructor: A special function defined in a class that is used to set initial values for the member variables of the class. It is invoked or called implicitly when an object is being created. Constructors do not return values and the names of the constructors should be the same as the class name.

4. More about Constructors: a class can have multiple constructors. The constructor without parameters is called the default constructor. The constructors with parameters are called parameterized constructors. Only one constructor with matching parameter list will be called when an object is being created.
class Student
{
public:
    Student(); //default constructor
    Student(string nm, float g, int a); //parameterized constructor
    string getName();
    void setName();
private:
    string name;
    float gpa;
    int age;
};

The definition of constructors:

Student::Student()
{
    name = "John";
    gpa = 3.0;
    age = 18;
}

Student::Student(string nm, float g, int a)
{
    name = nm;
    gpa = g;
    age = a;
}

5. ACCESSIBILITY: There are three levels of accessibility: private, protected, and public. When a member variable is defined as private, it can be accessed only by the member functions of the same class. "protected" is accessible to the functions defined in child classes. "public" is accessible from anywhere.

6. MEMBER VARIABLES: Member functions of a class can directly access and manipulate private member variables of the same class. When a member variable is mentioned in a member function, it is the calling object's member variable. For clarity, sometimes we use "this" to refer to the calling object in member functions by "this->memberVariable." Note that "this" is a pointer to the calling object.

7. MEMBER FUNCTIONS: Member functions can be called from other member functions of the same class. Outside of a class, public member functions must be called by an object of the same type.
8. **CLIENT PROGRAMS:** The programs that use classes are the client or application programs such as a program with main() function. The client programs can access/refer ONLY the public members (variables and functions) of a class.

```cpp
int main()
{
    Student st1;        //The default constructor
    Student st2("Shannon", 3.6, 21); //The non-default constructor
    st1.setName("David");
    cout << st2.getName();

    return 0;
}
```

9. **DECLARATION AND DEFINITION:** The declarations of a class provide a list of member variables and function prototypes. The definitions mean the implementations of the member functions of the class.

10. **INFORMATION Hiding:** Once the member variables and/or functions of a class are declared as private, they cannot be directly accessed or referred from client programs. In other words, they are “hidden” from the client programs. The only way to access or manipulate the private members of a class is through the public functions (interfaces) defined in the class. In the real-world, the creators of classes and the users of the classes are two different group of people or software developers. The users typically don’t care much about how the classes are implemented. As long as they know how the public functions (only prototypes)/interfaces of classes are declared, they should be able to use/call them.

Therefore, any implementation changes made to a class or its member functions don’t affect the use of the class by the users if the prototypes of the public functions remain the same.

11. **Pointers:** A pointer variable stores memory addresses as its value. Sample pointer variable notations:

```cpp
int a = 5;
int * p;
p = &a;       //variable p stores the memory address of a
cout << *p << endl; //asterisk * is called dereference operator. 5
*p = 10; //Indirectly change variable a’s value to 10
p = NULL; //NULL is a special value for pointer variables
```

12. **Reference Variable:** A reference variable is an alternative name (alias) for an object.
int a = 8;
int & p = a;
cout << a << endl;    //8
p = 10;               //indirectly change the value variable a to 10
cout << a << endl;    //10

13. ARRAY AND POINTER: An array can be manipulated through pointers because the fact that the name of an array variable itself is considered as a pointer variable pointing to the memory address of the first array element sometimes called base address of the array. For example.

int a[5] = {12, 2, 5, 4, 8};
int * p;
p = a;                //p points to the first element of array a
cout << *a << endl;   //print out the value of the first element of array a
cout << a[0] << endl;  //it is equivalent to the preceding cout
cout << *(a + 3) << endl;  //equivalent to a[3]

14. DYNAMIC DATA: A variable or object can be created “dynamically” through “new” operator. The memory space allocated to dynamic data will be in “heap” or “free store” memory region instead of “stack” region where static data is located. Dynamic data need to be deleted explicitly by programmers to release its member space to the system for reuse when they are no longer used in programs. For example, s2 in the main() below.

class Student {
public:
    Student() { age = 20; }
    Student(int a) { age = a; }
    ~Student() { cout << "Destructor is called" << endl; }
    void print() { cout << age << endl; }
private:
    int age;
};

int main()
{
    Student s;
    s.print();
}
Student * s2 = new Student(22);
s2->print();
delete s2;
return 0;
}

15. DESTRUCTOR: A destructor is used to release the memory of an object when it is deleted. All static variables will be deleted automatically just before the containing functions return. All dynamic data are deleted explicitly by a delete statement.

16. COPY CONSTRUCTOR: The copy constructor is called when an object is being created with another object. A default copy constructor is provided by the system if the user doesn’t provide one. The default one basically copies member variables component by component. In the following sample code, the copy constructor is called when s2 and s3 are being created.

```cpp
int main()
{
    Student s1; // Default constructor will be called
    Student s2(s1); // copy constructor will be called
    Student s3 = s1; // copy constructor will be called

    return 0;
}
```

17. ASSIGNMENT OPERATOR = OVERLOADING: Assignment operator is normally used for arithmetic operations of primitive (built-in) data types. If it is used for objects, the default assignment operator = copies member variables component by component. If the default behavior is not desired, the user needs to re-define the assignment operator = for the classes.

```cpp
#include <iostream>

using namespace std;

class Student
{
public:
    Student(); // Default constructor
};
```
age = a;
cout << "Parameterized constructor is called" << endl;
}

Student(const Student & rhs)
{
age = rhs.age + 10;
cout << "Copy constructor is called" << endl;
}

//Assignment operator = overloading
Student & operator= (const Student & rhs)
{
age = rhs.age + 4;

cout << "Assignment operator = overloading function will be called" << endl;
return * this; //Return a reference to itself
}
~Student()
{
cout << "Destructor is called" << endl;
}
void print()
{
cout << age << endl;
}
private:
int age;
);

int main()
{
Student s1; //Default constructor will be called
//Student s2(s1); //copy constructor will be called
Student s2 = s1; //copy constructor will be called
Student s3(25); //Parameterized constructor will be called
Student * s4 = new Student(28);
s2 = s1; //Assignment operator = overloading function will be called
s1.print();
s2.print();
s3.print();
s4->print();

delete s4; //Need to explicitly delete this dynamic data

return 0;
}

//Output:

//Default constructor is called
//Copy constructor is called
//Parameterized constructor is called
//Parameterized constructor is called
//Assignment operator = overloading function will be called
//20
//24
//25
//28
//Destructor is called
//Destructor is called
//Destructor is called
//Destructor is called
//Press any key to continue . . .

(a) Shallow copy: When an object contains a dynamic data (that is, created through new operator) and is copied from or initialized from another object, default copy constructor copies member variables piece by piece from one object to the other. Since dynamic data are pointed by pointer type variables, the dynamic data were not copied but pointers. As such, a single copy of a dynamic data is pointed or referenced by two objects. When one object deletes the dynamic data, the other object cannot access it any more.

(b) Deep copy: There are two copies of dynamic data, one for each object.

18. MEMORY LEAK: Memory leak occurs when a memory space cannot be accessed after an improper assignment involving pointer variables.