# Helper Functions

In object-oriented programming, not all instructions that support a class need to be included in the class definition. A well encapsulated class can accept external support in the form of global functions containing additional logic. We call these functions helper functions.

**Free Helpers**

* A free or loosely coupled helper function is a function that **obtains all of its information from the public member functions** of the class that it supports.
* A free helper function **does not require access** to the private members of its class.
* The **coupling** between a free helper function and its class is **minimal**, which is an ideal design solution.

**Comparison**

Consider a helper function that compares two objects of the same class. The function returns **true** if the objects have the same data values and **false** if they differ.

Example

* Let us add three queries (**getStudentNo()**, **getNoGrades()** and **getGrade()**) to our Student class definition and a helper function named areIdentical() as support.
* For conciseness, let us assume that all grades are stored in static memory.

| **// Student.h const int NG = 20; class Student {  int no;  float grade[NG];  int ng; public:  Student();  Student(int);  Student(int, const float\*, int);  void display() const;  int getStudentNo() const { return no; }  int getNoGrades() const { return ng; }  float getGrade(int i) const { return i < ng ? grade[i] : 0.0f; } }; bool areIdentical(const Student&, const Student&);** |
| --- |

The implementation file contains the definition of our helper function.

| **// Student.cpp #include <iostream> using namespace std; #include "Student.h"  Student::Student() {  no = 0;  ng = 0; }  Student::Student(int n) {  \*this = Student(n, nullptr, 0); }  Student::Student(int sn, const float\* g, int ng\_) {  bool valid = sn > 0 && g != nullptr && ng\_ >= 0;  if (valid)  for (int i = 0; i < ng\_ && valid; i++)  valid = g[i] >= 0.0f && g[i] <= 100.0f;   if (valid) {  // accept the client's data  no = sn;  ng = ng\_ < NG ? ng\_ : NG;  for (int i = 0; i < ng; i++)  grade[i] = g[i];  } else {  \*this = Student();  } }  void Student::display() const {  if (no > 0) {  cout << no << ":\n";  cout.setf(ios::fixed);  cout.precision(2);  for (int i = 0; i < ng; i++) {  cout.width(6);  cout << grade[i] << endl;  }  cout.unsetf(ios::fixed);  cout.precision(6);  } else {  cout << "no data available" << endl;  } }  bool areIdentical(const Student& lhs, const Student& rhs) {  bool same = lhs.getStudentNo() == rhs.getStudentNo() &&  lhs.getNoGrades() == rhs.getNoGrades();  for (int i = 0; i < lhs.getNoGrades() && same; i++)  same = lhs.getGrade(i) == rhs.getGrade(i);  return same; }** |
| --- |

The following client code compares the two objects:

| **// Compare Objects // compare.cpp #include <iostream> #include "Student.h" using namespace std;  int main () {  float gh[] = {89.4f, 67.8f, 45.5f};  Student harry(1234, gh, 3), harry\_(1234, gh, 3);  if (areIdentical(harry, harry\_))  cout << "are identical" << endl;  else  cout << "are different" << endl; }** |
| --- |

**The Cost of Friendship**

* A class definition that grants friendship to a helper function allows that f**unction to alter the values of its private data members**.
* Granting friendship pierces encapsulation.
* As a rule, we grant friendship judiciously only to helper functions that require both r**ead and write access** to the private data members.

**Summary**

* A helper function is a global function that supports a class
* A helper function refers to the class that it supports through its explicit parameter(s)
* A helper operator is typically an operator that does not change the value of its operands
* A friend function has direct access to the private members of the class that granted the function friendship
* Friendship is neither reciprocal, transitive, nor exclusive
* Free helper functions reduce coupling at the cost of bloating a class
* Friendly helper functions reduce bloating at the cost of piercing encapsulation