Structures

- **Problem:**

  A television quiz show has information about a number of people who want to become contestants. The name of each potential contestant and some personal characteristics are on file. The quiz show producer wants to get an answer to a question such as: "Which contestants have blond hair?" or "Which contestants are 21 years old?"

  Write a program to do the following: read in information for a group of contestants and count the number of contestants read in. The information about each contestant consists of:

  - name (last, first)
  - gender (F or M)
  - hair color (red, black, brown, blond, gray, or bald)
  - age
  - job title
  - annual salary (to two decimal places)

  e.g., Smith Mary F brown 27 lawyer 87654.32

  After reading in all the information, print a table containing all the information for all contestants. The table should have appropriate column headings. A typical row of the table would be:

  Mary Smith F brown 27 lawyer $87654.32

  Now print a menu on the screen which allows the user to select a particular trait which is desired in a contestant. The menu contains the names of all possible traits: age, hair color, salary, gender, and job title. In addition, the menu offers the option of quitting the program.

After identifying the trait desired, prompt the user to enter a value that corresponds to that trait (e.g., 17 for age, M for gender, or 50000 for salary). The program prints a list of all contestants who have the selected trait (for salary, we want all contestants whose salary is greater than or equal to the value requested). The program prints their names (first, last). There should also be a heading indicating what question is being answered. For example:

Contestants whose age is 27

Mary Smith
Paul Cooper

At this point, the program presents the menu again to allow the user to make another selection. the program continues to process requests until "quit" is chosen from the menu.

- **Pseudocode for the Main Program:**

```
/* first part */
call readdata() to read in and create the contestant database
call printdata() to print the contestant database
/* second part */
do
    call printmenu() to print a menu of choices
    prompt user to make a selection
    call function to respond to a particular choice
while user wishes to continue
```

Structures

- **Concept of a Structure:**
  - All components of an array are of the same type.

  - The components (fields) of a **structure** can be of different types. The objects declared within a structure are called its **members.**

- **Declaration of a Structure Prototye:**

```
struct struct_name {
    data_type_1 ident_1;        //member variables
    data_type_2 ident_2;        //or data members
    ...
    data_type_n ident_n;
};
```

- **Declaration of Variables whose Data Type is a Structure:**
  struct_name object_1, object_2, ...;

- **Note:**
  Each variable (object) is said to be an **instance** of the structure

- **Example:**

```
struct Address {
    int housenumber;
    string streetname;
};

Address home_address, work_address;
```

- **Accessing the Elements of a Structure:**
  structurename.membername

```
home_address.housenumber = 123;
home_address.streetname =  "Main St.";
cin >> work_address.housenumber;
getline(cin, work_address.streetname);
```

- **Structure Assignment:**
  If two structures are of the same struct type one can assign the value of one structure to the other.

  work_address = **home_address**;

- **Example:**

```
struct Student {
    string name;
    double average;
    char lettergrade;
};

Student freshman, beststudent;
...

if (freshman.average > 3.5)
    freshman.lettergrade = 'A';

if (freshman.average > beststudent.average)
{
    beststudent.name = freshman.name;
    beststudent.average = freshman.average;
    beststudent.lettergrade = freshman.lettergrade;
}

or

if (freshman.average > beststudent.average)
    beststudent = freshman;
```

- **Initializing a Structure:**

```
struct Student {
    string name;
    double average;
    char lettergrade;
};

Student freshman = {"Sam Starter", 2.03, 'C'};
Student sophomore = {"Joe Later", 3.45};
```

Using a Constructor to Initialize Structures

- **Constructor:**
  A constructor is a special function that can be used to set up and initialize (construct) a structure. It looks like a regular function except that its name is the same as the name of the structure tag and it has no return data type.

- **Example: Structure with Constructor**

```
struct Student {
    string name;
    double average;
    char lettergrade;

    Student()                   //Constructor
    {
       name = "";               //note the null string
       average = 3.99;
       lettergrade = 'A';
    }
};

Student freshman, sophomore;

freshman.name = "Sam Starter";
sophomore.name = "Joe Later";

cout << freshman.name;          //prints Sam Starter
cout << freshman.average;       //prints 3.99
cout << freshman.lettergrade    //prints 'A'
cout << spohomore.name;         //prints Joe Later
cout << sophomore.average;      //prints 3.99
cout << sophomore.lettergrade   //prints 'A'
```

Constructors that Accept Arguments

- **Constructor with Arguments:**

```
struct Student {
    string name;
    double average;
    char lettergrade;

    Student(string n, double avg, char grade)   //Constructor
    {
        name = n;
        average = avg;
        lettergrade = grade;
    }
};

Student freshman("Sam Starter", 2.03, 'C');
Student sophomore("Joe Later", 3.45, 'A');

cout << freshman.name;              //prints Sam Starter
cout << freshman.average;           //prints 2.03
cout << freshman.lettergrade        //prints 'C'
cout << spohomore.name;             //prints Joe Later
cout << sophomore.average;          //prints 3.45
cout << sophomore.lettergrade       //prints 'A'
```

- **A Structure with Member Element Arrays:**

```
struct Triangle {
   string type;
   int angle[3];
};

Triangle t1;
...

if (t1.angle[0] = = t1.angle[1] &&
      t1.angle[1] = = t1.angle[2])
   t1.type = "equilateral";
else if (t1.angle[0] = = t1.angle[1] ||
      t1.angle[1] = = t1.angle[2] ||
      t1.angle[0] = = t1.angle[2])
   t1.type = "isosceles";
else
   t1.type = "scalene";
```

- **A Structure with Many Elements:**

```
struct Book {
   string lastname;
   string firstname;
   string title;
   string pubname;
   string pubcity;
   string pubstate;
   int yearpub;
   string call_number;
   int numcopies;
};

Book book;
```

Nested Structures

- **Declaration of a Nested Structure:**

```
struct Name {
    string last;
    string first;
};

struct Publisher {
    string name;
    string city;
    string state;
};

struct Book {
    Name author;
    string title;
    Publisher publisher;
    int yearpub;
    string call_number;
    int numcopies;
};

Book book;
```

- **Accessing Members of a Nested Structure:**

```
book.publisher.name = "Prentice-Hall";
book.author.last = "Harrow";
book.yearpub = 2007;          // not nested
cout << book.author.first << " " << book.author.last <<
    endl;
```

An Array of Structures

- **Example:**

```
struct Name {
   string last;
   string first;
};

struct Street_address {
   int housenum;
   string street;
};

struct Address {
   Street_address street_address;
   string city;
   string state;
   string zip;
};

struct Employee {
   Name name;
   string socsecnum;
   Address address;
};

Employee emp[100];
....

cout << emp[0].address.zip << endl << endl;
for (int i = 0; i < 10; i++)
   cout << emp[i].address.street_address.housenum <<
      " " << emp[i].address.street_address.street << endl;
cout << endl << "All employees who live in New Jersey:"
   << endl;
for (int i = 0; i < 100; i++)
   if (emp[i].address.state == "NJ")
      cout << emp[i].name.last << ", " <<
         emp[i].name.first << endl;
```

- **Example of Arrays at Multiple Levels:**

```
struct Name {
    string last;
    string first;
};

struct Classmark {
    int test[5];
    double average;
    char lettergrade;
};

struct Student {
    Name name;
    int numclasses;
    Classmark class[5];
    double overallavg;
};

Student student;

int sum;
double sum_classavg = 0.0;
...

for (int i = 0; i < student.numclasses; i++)
{
    sum = 0;
    for (int j = 0; j < 5; j++)
        sum += student.class[i].test[j];
    student.class[i].average = (double)sum / 5;
    sum_classavg += student.class[i].average;
}
student.overallavg = sum_classavg / student.numclasses;
cout << student.name.last << ", " << student.name.first
    << " " << student.overallavg << endl;
```

Using a Structure in a Function

- **Example:**

```
/* program to print components of employee structure */
#include <iostream>
#include <string>
using namespace std;

struct Name {
    string last;
    string first;
};
struct Street_address {
    int housenum;
    string street;
};
struct Address {
    Street_address street_address;
    string city;
    string state;
    string zip;
};
struct Employee {
    Name name;
    string socsecnum;
    Address address;
};
void print_emp(Employee);              // Function Prototype

int main ()
{
    Employee worker;
    ...                                // read in worker info
    print_emp(worker);
    return 0;
}
void print_emp(Employee worker)        // passed by value
{
    cout << worker.name.last << ", " <<worker.name.first
        << " " << worker.socsecnum << endl;
    cout << worker.address.street_address.housenum <<
        " " << worker.address.street_address.street << endl;
    cout << worker.address.city << " " << worker.address.state
        << " " << worker.address.zip << endl;
    return;
}
```

Changing a Structure in a Function

- **Example:**

```
/* program to read in & print employee structure */
#include <iostream>
#include <string>
using namespace std;
   ...
struct Employee {
   ...
};

/* Function Prototypes */
void read_emp(Employee &);
void print_emp(Employee);

int main ()
{
   Employee worker;

   read_emp(worker);
   print_emp(worker);
   return 0;
}

void print_emp(Employee worker)        // pass by value
{
   ...
}

void read_emp(Employee &worker)        // pass by reference
{
   cin >> worker.name.last;
   cin >> worker.name.first;
   cin >> worker.socsecnum;
   cin >> worker.address.street_address.housenum;
   getline(cin, worker.address.street_address.street);
   ...
   return;
}
```

Using a Constant Reference Parameter

- **Example:**

```
/* program to read in & print employee structure */
#include <iostream>
#include <string>
using namespace std;
   ...
struct Employee {
   ...
};

/* Function Prototypes */
void read_emp(Employee &);
void print_emp(const Employee &);

int main ()
{
   Employee worker;

   read_emp(worker);
   print_emp(worker);
   return 0;
}

void print_emp(const Employee &worker)    // pass by const ref.
{
   cout << worker.name.last << ", " <<worker.name.first
      << " " << worker.socsecnum << endl;
   cout << worker.address.street_address.housenum <<
      " " << worker.address.street_address.street << endl;
   cout << worker.address.city << " " << worker.address.state
      << " " << worker.address.zip << endl;
   return;
   ...
}

void read_emp(Employee &worker)       // pass by reference
{
   ...
}
```

Sending an Array of Structures as a Parameter

- **Example:**

```
#include <iostream>
#include <string>
using namespace std;

struct Votes {
   string name;
   int numvotes;
};

/* Function Prototype */
void readvotes(Votes [], int &);

int main()
{
   Votes candidate[100];
   int numcands;

   readvotes(candidate,numcands);

   return 0;
}

void readvotes(Votes candidate[], int &numcands)
{
   cin >> numcands;
   for (int i = 0; i < numcands; i++)
   {
      getline(cin, candidate[i].name);
      cin >> candidate[i].numvotes;
      cout << candidate[i].name << endl;
      cout << candidate[i].numvotes << endl;
   }
   return;
}
```

A Function which Returns a Structure

- **Example:**

```
#include <iostream>
#include <string>
using namespace std;

struct Votes {
   string name;
   int numvotes;
};

/* Function Prototype */
Votes whowon(Votes [], int);       // returns a struct
void readvotes(Votes [], int &);

int main()
{
   Votes candidate[100],winner;
   int numcands;

   readvotes(candidate,numcands);
   winner = whowon(candidate,numcands);
   cout << winner.name << " won with " << winner.numvotes
      << " votes" << endl;
   return 0;
}
...
Votes whowon(Votes candidate[], int numcands)
{
   Votes highest;

   highest = candidate[0];
   for (int i = 1; i < numcands; i++)
   {
      if (candidate[i].numvotes > highest.numvotes)
         highest = candidate[i];
   }
   return (highest);
}
```

Return to the Problem

- **Pseudocode for the Main Program:**

```
/* first part */
call readdata() to read in and create the contestant database
call printdata() to print the contestant database
/* second part */
do
    call printmenu() to print a menu of choices
    prompt user to make a selection
    call function to respond to a particular choice
while user wishes to continue
```

- **Declaration for the Contestant Database:**

```
const int MAX_NUM_CONTESTANTS = 50;

struct Name {
    string last;
    string first;
};
struct Job_info {
    string title;
    double salary;
};
struct Personal_info {
    char gender;
    string haircolor;
    int age;
    Job_info job;
};
struct Contestant {
    Name name;
    Personal_info personal;
};

...

Contestant contestant[MAX_NUM_CONTESTANTS];
```

- **The Main Program:**

```
/* Contestant Database */
#include <iostream>
#include <string>
#include <fstream>
using namespace std;

const int MAX_NUM_CONTESTANTS = 50;

/* structure definitions go here */
...

/* Function Prototypes */
void readdata(Contestant [], int &);
void printdata(Contestant [], int);
void printmenu();
```

```
int main()
{
    Contestant contestant[MAX_NUM_CONTESTANTS];
    int num;
    char choice;
    bool not_done = true;

    /* first part */
    /* fill and print database */
    readdata(contestant,num);
    printdata(contestant,num);

    /* second part */
    /* call functions to read and process requests */
    do {
        printmenu();
        cin >> choice;
        switch(choice)
        {
            case 'Q':
                not_done = false;
                break;
            case 'A':
                findage(contestant,num);
                break;
            case 'G':
                findgender(contestant,num);
                break;
            case 'H':
                findhair(contestant,num);
                break;
            case 'T':
                findtitle(contestant,num);
                break;
            case 'S':
                findsalary(contestant,num);
                break;
            default:
                cout << "Invalid choice - try again\n";
                break;
        }
    } while (not_done);

    return 0;
}
```

- **The Function readdata():**

```
   /* Function readdata() */
   void readdata(Contestant contestant[], int &count)
   {
      // open input file
      ifstream cfile("c:\\mypgms\\myinput.txt");
//    ifstream cfile("con");          //un-comment for debugging

      count = 0;                      //initialize count

      while (cfile >> contestant[count].name.last)
      {
         cfile >> contestant[count].name.first;
         cfile >> contestant[count].personal.gender;
         cfile >> contestant[count].personal.haircolor;
         cfile >> contestant[count].personal.age;
         cfile >> contestant[count].personal.job.title;
         cfile >> contestant[count].personal.job.salary;
         count++;
      }

      cfile.close();
      return;
   }
```

- **The Function printdata():**

```
   /* Function printdata: */
   void printdata(Contestant contestant[], int num)
   {
      // open output file
      ofstream dbfile("c:\\mypgms\\myoutput.txt");
//    ofstream dbfile("con");      //un-comment for debugging

      dbfile << "\t\tContestants in the Database\n\n";
      dbfile << "Name\t\tGender\tHair\tAge\tTitle\tSalary\n\n";
      for (int count = 0; count < num; count++)
      {
         dbfile << contestant[count].name.first;
         dbfile << " "  << contestant[count].name.last;
         dbfile << "\t" << contestant[count].personal.gender;
         dbfile << "\t" << contestant[count].personal.haircolor;
         dbfile << "\t" << contestant[count].personal.age;
         dbfile << "\t" << contestant[count].personal.job.title;
         dbfile << "\t" << contestant[count].personal.job.salary;
         dbfile << endl;
      }

      dbfile.close();
      return;
   }
```

- **The Function printmenu():**

```
/* Function printmenu() */
void printmenu()
{
   cout << endl << endl";
   cout << "To obtain a list of contestants with a given\n";
   cout << "trait, select a trait from the list and type in\n";
   cout << "the letter corresponding to that trait.\n\n";
   cout << "To quit, select Q.\n\n";
   cout << "\t**************************\n";
   cout << "\t   List of Choices       \n";
   cout << "\t**************************\n";
   cout << "\t    Q -- quit\n";
   cout << "\t    A -- age\n";
   cout << "\t    G -- gender\n";
   cout << "\t    H -- hair color\n";
   cout << "\t    T -- title\n";
   cout << "\t    S -- salary\n";
   cout << endl <<endl << "\tEnter your selection: ";
   return;
}
```

- **The Function findage():**

```
/* Function findage() */
void findage(Contestant contestant[], int num)
{
   int agewanted,found = 0;

   cout << "\n\nEnter the age you want: ";
   cin >> agewanted;
   cout << "\nContestants whose age is " << agewanted << "\n\n";
   for (int count = 0; count < num; count++)
      if (contestant[count].personal.age == agewanted)
      {
         cout << contestant[count].name.first << " " <<
            contestant[count].name.last << endl;
         found++;
      }
   if (!found)
      cout << "No contestants of this age\n\n";
   else
      cout << endl << found << " contestants found\n";

   // give user a chance to look at output
   // before printing menu
   pause();

   return;
}
```

- **The Function pause():**

```
/* Function pause() */
void pause()
{
   system("pause");
   return;
}
```

- **Revised Main Program:**

```
/* Contestant Database */
/* includes go here */
...

const int MAX_NUM_CONTESTANTS = 50;

/* structure definitions go here */
...

/* Function Prototypes */
void readdata(Contestant [], int &);
void printdata(Contestant [], int);
void printmenu();
void findage(Contestant [], int);
void findgender(Contestant [], int);
void findhair(Contestant [], int);
void findtitle(Contsetant [], int);
void findsalary(Contestant [], int);
void pause();

int main()
{
   Contestant contestant[MAX_NUM_CONTESTANTS];
   int num;
   char choice;
   bool not_done = true;

   /* first part */
   /* fill and print database */
   readdata(contestant,num);
   printdata(contestant,num);

   /* second part */
   /* call functions to read and process requests */
   do {
      ...
   } while (not_done);
   return 0;
}
```

Abstract Data Types

- An **abstract data type (ADT)** is a (programmer defined) data type that specifies the legal values the data type can hold and the operations that can be done on them without the details of how the data type is implemented.

- **Abstraction:**
  A general model of something that includes only the general characteristics of an object without the details that characterize a specific instance of the object. (e.g., an abstract triangle is a 3-sided polygon. A specific triangle can be scalene, isosoles, or equilateral.)

- **Data Type:**
  Defines the kind of values an object can have and what operations apply to it. (e.g., a "double" can hold numbers like 100.4, -5.67, etc. and you can perform addition, subtraction, multiplication, and division operations on them, but not the modulus operation.) Note that the implementation is not specified.

- **Data Abstraction:**
  The separation of a data type's logical properties from its implementation is known as **data abstraction.**

Object Oriented Programming

- **Procedural Programming:**
  Focuses on the processes and actions that occur in a program. Centered on creating procedures and functions.

- **Object-Oriented Programming:**
  Centered around objects that encapsulate both data and the functions that operate on them.

- **Class:**
  A **class** is a programmer defined data type consisting of **member variables** and **member functions (methods)**. (It is similar to a structure.) It describes the properties that all instances of the class will have.

- **Object:**
  An **object** in **an instance of a class** (or structure).

- **Object Attributes:**
  The member variables (or member data) of a class.

Classes Declared with class

- By default, unless we explicitly specify otherwise, all the members of a **struct** are **public**. This means all members can be accessed with the **dot operator**

- The keyword **class** can be used in place of struct to declare a class. By default, when **class** is used, all members are **private**. Aside from certain exceptions, all private members are **hidden** and can not be accessed even with the dot operator.

- The keywords **private** and **public** allow us to declare some class members private and others public. All declarations following one of these keywords will be private or public, respectively, until another such keyword is encountered.

- **Access Specifiers:**
  private and **public** are known as **access specifiers**.

```
class Contestant {
   //private member variables
   Name name;
   Personal_info personal;
};

class Contestant {
  public:
   //public member variables
   Name name;
   Personal_info personal;
};

class Contestant {
  public:
   Name name;                  //public member variable
  private:
   Personal_info personal;     //private member variable
};
```

- **Revised Main Program - Using class - Public Variables:**

```
/* Contestant Database */
#include <iostream>
#include <string>
#include <fstream>
using namespace std;

//constant definitions
const int MAX_NUM_CONTESTANTS = 50;

//class definitions
class Name {
  public:
    string last;
    string first;
};

class Job_info {
  public:
    string title;
    double salary;
};

class Personal_info {
  public:
    char gender;
    string haircolor;
    int age;
    Job_info job;
};

class Contestant {
  public:
    Name name;
    Personal_info personal;
};

//function prototypes
...

int main()
...
```

Member Functions

- C++ allows classes to contain functions as members. These **member functions (or methods)** are used to send messages and receive replies from objects declared using a class. Member functions designed to provide a user interface to an object should be declared **public**.

- **Example:**

```
class Contestant {
    // private member variables
    Name name;
    Personal_info personal;
  public:
    // public member functions
    bool readData(ifstream &);
    void printData(ofstream &);
    bool compareAge(int);
};
```

- The member functions (methods) readData(), printData(), and compareAge() can be used to manipulate objects of type Contestant.

```
Contestant contestant[MAX_NUM_CONTESTANTS];

count = 0;
while (contestant[count].readData(cfile))
    count++;

for (int count = 0; count < num; count++)
    contestant[count].printData(dbfile);

found = 0;
for (int count = 0; count < num; count++)
    if (contestant[count].compareAge(agewanted))
        found++;
```

- **Member Function readData():**

```
/* Contestant member function readData():
 * Input:
 *     cfile - a reference to the input file
 * Process:
 *     read the input file and load the object's data members
 * Output:
 *     return true if file is read and object fields loaded
 *     else return false if EOF reached.
 */
bool Contestant::readData(ifstream &cfile)
{
   if (cfile > > name.last)
   {
      cfile > > name.first;
      cfile > > personal.gender;
      cfile > > personal.haircolor;
      cfile > > personal.age;
      cfile > > personal.job.title;
      cfile > > personal.job.salary;
      return true;
   }
   return false;
}
```

- **Mutators:**
  Member functions that change the value of a member variable are known as **mutators** (or **setter functions**).

- **Member Function printData():**

```
/* Contestant member function printData():
 * Input:
 *    dbfile - a reference to the output file
 * Process:
 *    write the object's data members to the output file
 * Output:
 *    the written data members of the object
 */
void Contestant::printData(ofstream &dbfile)
{
   dbfile.setf(ios::fixed,ios::floatfield);
   dbfile.precision(2);                    //set decimal precision

   dbfile << name.first;
   dbfile << " "  << name.last;
   dbfile << "\t" << personal.gender;
   dbfile << "\t" << personal.haircolor;
   dbfile << "\t" << personal.age;
   dbfile << "\t" << personal.job.title;
   dbfile << "\t";
   dbfile.width(9);
   dbfile << personal.job.salary;
   dbfile << endl;

   return;
}
```

- **Accessors:**
  Member functions that use values from a member variable but do not change its value are **accessors** (or **getter functions**).

- **Member Function compareAge():**

```
/* Contestant member function compareAge():
 * Input:
 *     agewanted - age wanted
 * Process:
 *     compare agewanted to the object's age
 * Output:
 *     if ages are the same, print the object's name
 *          and return true
 *     else return false
 */
bool Contestant::compareAge(int agewanted)
{
   if (personal.age == agewanted)
   {
     cout << name.first << " " << name.last << endl;
     return true;
   }
   return false;
}
```

- **Revised Main Program - Using class - Private Variables:**

```
/* Contestant Database - with member functions */
#include <iostream>
#include <string>
#include <fstream>
using namespace std;

//constant definitions
const int MAX_NUM_CONTESTANTS = 50;

//class definitions
class Name {
  public:
    string last;
    string first;
};

class Job_info {
  public:
    string title;
    double salary;
};

class Personal_info {
  public:
    char gender;
    string haircolor;
    int age;
    Job_info job;
};

class Contestant {
    // member variables                    //private variables
    Name name;
    Personal_info personal;
  public:
    // member functions                    //public functions
    bool readData(ifstream &cfile);
    void printData(ofstream &dbfile);
    bool compareAge(int agewanted);
};
```

```
// Function Prototypes
void readdata(Contestant [], int &);
void printdata(Contestant [], int);
void printmenu(void);
void findage(Contestant [], int);
void findgender(Contestant [], int);
void findhair(Contestant [], int);
void findtitle(Contestant [], int);
void findsalary(Contestant [], int);
void pause(void);

int main()
{
   Contestant contestant[MAX_NUM_CONTESTANTS];
   int num;
   char choice;
   bool not_done = true;

   /* first part */
   /* fill and print database */
   readdata(contestant,num);
   printdata(contestant,num);

   /* second part */
   /* call functions to read and process requests */
   do {
      printmenu();
      cin >> choice;
      switch (choice) {
         ...
      };
   } while (not_done);

   return 0;
}
```

- **Revised Function readdata():**

```
void readdata(Contestant contestant[], int &count)
{
   // open input file
   ifstream cfile("c:\\mypgms\\myinput.txt");
//   ifstream cfile("con");          //un-comment for debugging

   count = 0;                 //initialize count

   while (contestant[count].readData(cfile))
      count++;

   cfile.close();
   return;
}
```

- **Revised Function printdata():**

```
void printdata(Contestant contestant[], int num)
{
   // open output file
   ofstream dbfile("c:\\mypgms\\myoutput.txt");
//   ofstream dbfile("con");        //un-comment for debugging

   dbfile << "\t\tContestants in the Database\n\n";
   dbfile << "Name\t\tGender\tHair\tAge\tTitle\tSalary\n\n";

   for (int count = 0; count < num; count++)
      contestant[count].printData(dbfile);

   dbfile.close();
   return;
}
```

- **Revised Function findage():**

```
void findage(Contestant contestant[], int num)
{
   int agewanted,found = 0;

   cout << "\n\nEnter the age you want: ";
   cin >> agewanted;
   cout << "\nContestants whose age is " << agewanted << "\n\n";
   for (int count = 0; count < num; count++)
      if (contestant[count].compareAge(agewanted))
         found++;
   if (!found)
      cout << "No contestants of this age\n\n";
   else
      cout << endl << found << " contestants found\n";

   // give user a chance to look at output
   // before printing menu
   pause();

   return;
}
```

Separating Class Specification from Implementation

- **Class Specification File:**
  Class declarations are stored in their own header files called class specification files. The name of the header file is usually the same name as the class, with a .h extension.
  (e.g., Contestant.h)

- Use the **#ifndef, #define, and #endif directives** to act as an **include guard** to prevent the header file from being included more than once.

- Any function that uses the class should #include its header file. (e.g., **#include "Contestant.h"** - Note: Double quotes indicates that the file is in the current project directory)

- **Class Implementation File:**
  The member function definitions are stored in a separate .cpp file called the class implementation file. The file usually has the same name as the class with a .cpp extension.
  (e.g., Contestant.cpp)

- The class implementation file should be **compiled** (possibly separately) and **linked** to the main program to create an executable file. This process can be automated with a **project** or **make** utility.

- Integrated development environments also provide the means to create multi-file projects.

- **Class Declaration File: Contestant.h**

```
#ifndef CONTESTANT_H
#define CONTESTANT_H

#include <string>
using namespace std;

//class definitions
class Name {
  public:
    string last;
    string first;
};

class Job_info {
  public:
    string title;
    double salary;
};

class Personal_info {
  public:
    char gender;
    string haircolor;
    int age;
    Job_info job;
};

class Contestant {
    // member variables                    //private variables
    Name name;
    Personal_info personal;
  public:
    // member functions                    //public functions
    bool readData(ifstream &cfile);
    void printData(ofstream &dbfile);
    bool compareAge(int agewanted);
};

#endif
```

- **Class Implementation File: Contestant.cpp**

```
#include <iostream>
#include <string>
#include <fstream>

#include "Contestant.h"

using namespace std;


/* Contestant member function readData(): */

bool Contestant::readData(ifstream &cfile)
{
   ...
}


/* Contestant member function printData(): */

void Contestant::printData(ofstream &dbfile)
{
   ...
}


/* Contestant member function compareAge(): */

bool Contestant::compareAge(int agewanted)
{
   ...
}
```

- **Revised Main Program:**

```
#include <iostream>
#include <string>
#include <fstream>

#include "Contestant.h"

using namespace std;

// Function Prototypes
void readdata(Contestant [], int &);
void printdata(Contestant [], int);
void printmenu(void);
void findage(Contestant [], int);
void findgender(Contestant [], int);
void findhair(Contestant [], int);
void findtitle(Contestant [], int);
void findsalary(Contestant [], int);
void pause(void);

//constant definitions
const int MAX_NUM_CONTESTANTS = 50;

int main()
{
   ...
}

//function implementations
   ...
```

Performing I/O in a Class Object

- In general, it is considered good design to avoid having class member functions do I/O. This is so that anyone that uses a class is not locked into the way the class performs I/O. Classes should provide member functions for setting and getting values of member variables without doing I/O. (The exception is when a class is designed to specifically handle a program's I/O.)

- **Revised Example:**

```
class Contestant {
   // private member variables
   Name name;
   Personal_info personal;
 public:
   // public member functions prototypes

   // mutators (or setter member functions)
   void setLastname(string);
   void setFirstname(string);
   void setGender(char);
   void setHaircolor(string);
   void setAge(int);
   void setJobtitle(string);
   void setJobsalary(double);

   // accessors (or getter member functions)
   string getLastname();
   string getFirstname();
   char getGender();
   string getHaircolor();
   int getAge();
   string getJobtitle();
   double getJobsalary();

   bool compareAge(int);
};
```

- **Mutators (or Setter Functions):**

```
/* Contestant member function setLastname:
 * Input:
 *    lastname - contestant's last name
 * Process:
 *    sets the value of member variable name.last
 * Output:
 *    none
 */
void Contestant::setLastname(string lastname)
{
   name.last = lastname;
   return;
}

/* Contestant member function setFirstname:
 * Input:
 *    firstname - contestant's first name
 * Process:
 *    sets the value of member variable name.first
 * Output:
 *    none
 */
void Contestant::setFirstname(string firstname)
{
   name.first = firstname;
   return;
}

...

void Contestant::setGender(char gender)
{
   personal.gender = gender;
   return;
}
```

```
void Contestant::setHaircolor(string haircolor)
{
   personal.haircolor = haircolor;
   return;
}

void Contestant::setAge(int age)
{
   personal.age = age;
   return;
}

void Contestant::setJobtitle(string title)
{
   personal.job.title = title;
   return;
}

void Contestant::setJobsalary(double salary)
{
   personal.job.salary = salary;
   return;
}
```

- **Accessors (or Getter Functions):**

```
/* Contestant member function getLastname:
 * Input:
 *    none
 * Process:
 *    retrieves the value of member variable name.last
 * Output:
 *    returns the value of name.last
 */
string Contestant::getLastname()
{
   return (name.last);
}

/* Contestant member function getFirstname:
 * Input:
 *    none
 * Process:
 *    retrieves the value of member variable name.first
 * Output:
 *    returns the value of name.first
 */
string Contestant::getFirstname()
{
   return (name.first);
}

...

char Contestant::getGender()
{
   return (personal.gender);
}
```

```
string Contestant::getHaircolor()
{
   return (personal.haircolor);
}

int Contestant::getAge()
{
   return (personal.age);
}

string Contestant::getJobtitle()
{
   return (personal.job.title);
}

double Contestant::getJobsalary()
{
   return (personal.job.salary);
}
```

- **Member Function compareAge():**

```
/* Contestant member function compareAge():
 * Input:
 *    agewanted - age wanted
 * Process:
 *    compare agewanted to the object's age
 * Output:
 *    if ages are the same return true
 *    else return false
 */
bool Contestant::compareAge(int agewanted)
{
   if (personal.age == agewanted)
      return true;
   else
      return false;
}
```

- **Revised Function readdata():**

```
   /* Function readdata() */
   void readdata(Contestant contestant[], int &count)
   {
      string lastname,firstname,haircolor,jobtitle;
      char gender;
      int age;
      double salary;

      // open input file
      ifstream cfile("c:\\mypgms\\myinput.txt");
//    ifstream cfile("con");          //un-comment for debugging

      count = 0;                      //initialize count

      while (cfile >> lastname) {
         contestant[count].setLastname(lastname);
         cfile >> firstname;
         contestant[count].setFirstname(firstname);
         cfile >> gender;
         contestant[count].setGender(gender);
         cfile >> haircolor;
         contestant[count].setHaircolor(haircolor);;
         cfile >> age;
         contestant[count].setAge(age);
         cfile >> jobtitle;
         contestant[count].setJobtitle(jobtitle);
         cfile >> salary;
         contestant[count].setJobsalary(salary);
         count++;
      }

      cfile.close();
      return;
   }
```

- **Revised Function printdata():**

```
   /* Function printdata: */
   void printdata(Contestant contestant[], int num)
   {
      // open output file
      ofstream dbfile("c:\\mypgms\\myoutput.txt");
//    ofstream dbfile("con");      //un-comment for debugging

      dbfile << "\t\tContestants in the Database\n\n";
      dbfile << "Name\t\tGender\tHair\tAge\tTitle\tSalary\n\n";
      for (int count = 0; count < num; count++)
      {
         dbfile << contestant[count].getFirstname();
         dbfile << " "  << contestant[count].getLastname();
         dbfile << "\t" << contestant[count].getGender();
         dbfile << "\t" << contestant[count].getHaircolor();
         dbfile << "\t" << contestant[count].getAge();
         dbfile << "\t" << contestant[count].getJobtitle();
         dbfile << "\t" << contestant[count].getJobsalary();
         dbfile << endl;
      }

      dbfile.close();
      return;
   }
```

- **Revised Function findage():**

```
/* Function findage() */
void findage(Contestant contestant[], int num)
{
   int agewanted,found = 0;

   cout << "\n\nEnter the age you want: ";
   cin >> agewanted;
   cout << "\nContestants whose age is " <<agewanted<< "\n\n";
   for (int count = 0; count < num; count++)
      if (contestant[count].compareAge(agewanted))
      {
         cout << contestant[count].getFirstname()
            << " " << contestant[count].getLastname()
            << endl;
         found++;
      }
   if (!found)
      cout << "No contestants of this age\n\n";
   else
      cout << endl << found << " contestants found\n";

   // give user a chance to look at output
   // before printing menu
   pause();

   return;
}
```

Using a Constructor with a Class

```
// This program demonstrates when a constructor executes.
#include <iostream>
using namespace std;

class Demo
{
 public:
   Demo();                          // Constructor prototype
};

Demo::Demo()                        // Constructor function definition
{
   cout << "Now the default constructor is running.\n";
}

int main()
{
   cout << "This is displayed before the object is created.\n";

   Demo demoObj;                    // Define the Demo object.

   cout << "This is displayed after the object is created.\n";

   return 0;
}
```

Overloading Constructors

```
//class definitions
class Contestant{
    Name name;
    Personal_info personal;

  public:
    // Constructors
    Contestant()
    {
       name.first = "John";
       name.last = "Doe";
       personal.gender = 'M';
       personal.haircolor = "brown";
       personal.age = 35;
       personal.job.title = "Hobo";
       personal.job.salary = 0.00;
    }

    Contestant(string firstname, string lastname, char gender)
    {
       name.first = firstname;
       name.last = lastname;
       personal.gender = gender;
       personal.haircolor = "brown";
       personal.age = 35;
       personal.job.title = "Hobo";
       personal.job.salary = 0.00;
    }

    Contestant(string firstname, string lastname, char gender,
        string haircolor, int age, string title, double salary)
    {
       name.first = firstname;
       name.last = lastname;
       personal.gender = gender;
       personal.haircolor = haircolor;
       personal.age = age;
       personal.job.title = title;
       personal.job.salary = salary;
    }
```

```
    // mutators (or setter member functions)
    void setLastname(string);
    void setFirstname(string);
    void setGender(char);
    void setHaircolor(string);
    void setAge(int);
    void setJobtitle(string);
    void setJobsalary(double);

    // accessors (or getter member functions)
    string getLastname();
    string getFirstname();
    char getGender();
    string getHaircolor();
    int getAge();
    string getJobtitle();
    double getJobsalary();

    bool compareAge(int);
};
```

- **Example main() using Constructors:**

```
#include <iostream>
#include <string>
#include <fstream>
#include "Contestant.h"
using namespace std;

// Function Prototypes
void printdata(Contestant);

int main()
{
   Contestant contestant1;
   Contestant contestant2("Jane","Smith",'F');
   Contestant contestant3("Jack","Smith",'M', "Black", 27,
                          "Lawyer", 98765.43);
   printdata(contestant1);
   printdata(contestant2);
   printdata(contestant3);

   system("pause");
   return 0;
}

/* Function printdata: */
void printdata(Contestant contestant)
{
   // open output file
//    ofstream dbfile("c:\\mypgms\\myoutput.txt");
   ofstream dbfile("con");      //un-comment for debugging

   dbfile<< "Name\t\tGender\tHair\tAge\tTitle\tSalary" << endl;
   dbfile << contestant.getFirstname();
   dbfile << " "  << contestant.getLastname();
   dbfile << "\t" << contestant.getGender();
   dbfile << "\t" << contestant.getHaircolor();
   dbfile << "\t" << contestant.getAge();
   dbfile << "\t" << contestant.getJobtitle();
   dbfile << "\t" << contestant.getJobsalary();
   dbfile << endl << endl << endl;
   dbfile.close();
   return;
}
```

Destructors

- A destructor is a public member function that is automatically called when an object is destroyed.

- Destructors have the same name as the class and are preceded by a tilde (~).

- Destructors take no arguments and they can not be overloaded.

- **Example:**

```
// This program demonstrates a destructor.
#include <iostream>
using namespace std;

class Demo
{
  public:
       Demo();                      // Constructor prototype
       ~Demo();                     // Destructor prototype
};

Demo::Demo()                        // Constructor function definition
{
   cout << "An object has just been defined, so the"
       << " constructor is running." << endl;
}

Demo::~Demo()                       // Destructor function definition
{
   cout << "Now the destructor is running." << endl;
}

int main()
{
   Demo demoObj;                    // Declare a Demo object;

   cout << "The object now exists, but is about to be"
          << " destroyed." << endl;

  return 0;
}
```

Input Validation Objects

- Classes can be designed to validate user input.

- File CharRange.h - CharRange class specification file

```
#ifndef CHRANGE_H
#define CHRANGE_H

class CharRange
{
 private:
    char input;                 // User input
    char lower;                 // Lowest valid character
    char upper;                 // Highest valid character
 public:
    CharRange(char, char);
    char getChar();
};
#endif
```

- File CharRange.cpp - CharRange class function implementation file

```
#include <iostream>
#include <cctype>              // Needed to use toupper
#include "CharRange.h"
using namespace std;

//*********************************************************
// CharRange default constructor
//*********************************************************
CharRange::CharRange(char l, char u)
{
  lower = toupper(l);
  upper = toupper(u);
}

//*********************************************************
// CharRange member function getChar
// Inputs a character, validates it is in the correct
// range, and then returns the valid character.
//*********************************************************
char CharRange::getChar()
{
  cin.get(input);                 // Get a character
  cin.ignore();                   // Ignore the '\n' in the input buffer
  input = toupper(input);         // Uppercase the character

  // Ensure character is in the correct range
  while (input < lower || input > upper)
  {
    cout << "That is not a valid character.\n";
    cout << "Enter a value from " << lower;
    cout << " to " << upper << ".\n";
    cin.get(input);
    cin.ignore();
    input = toupper(input);
  }
  return input;
}
```

- **Sample Program:**

```
// This program uses the CharRange class and demonstrates its
// capabilities. This file should be combined into a project
// along with the CharRange.h and CharRange.cpp files.
#include <iostream>
#include "CharRange.h"
using namespace std;

int main()
{
   char ch;                        // Holds user input

   // Create a CharRange object named input
   // It will check for characters in the range J - N
   CharRange input('J', 'N');

   // Prompt the user to enter a letter
   cout << "Enter any of the characters J, K, L, M, or N.\n";
   cout << "Entering N will stop this program.\n";
   ch = input.getChar();

   // Continue allowing letters to be entered until a 'N' is input
   while (ch != 'N')
   {
      cout << "You entered " << ch << endl;
      ch = input.getChar();
   }
   return 0;
}
```

Private Member Functions

- Private member functions are for carrying out internal class activities. They may only be called from a member function of the same class.

- Sample Program:
  see textbook

Object Oriented Analysis and Design

- **Object-Oriented Analysis:**
  That phase of program development when the program functionality is determined from the requirements

1. Identify the Objects and Classes:
Determine the major data elements and decide what procedures and operations are required on these elements.

2. Define Each Class's Attributes:
A class's attributes are the data elements used to describe an object instantiated from the class. They are all the values needed for the object to function properly in the program.

3. Define Each Class's Behaviors:
Identify the activities or behaviors each class must be capable of performing.

4. Define the Relationships Between Classes:
Define the relationships that exist between and among the classes in a program.
- Access ("uses-a")
- Ownership/Composition ("has-a")
- Inheritance ("is-a")