**CS301 – Bootcamp: Classes and Structs**

**Case study: pet adoption**

**Structure or “struct”:** C++ construct that allows multiple variables to be grouped together

**Structures** allow us to mix \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ data types and store them within the \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ group. Then, we can treat this group as a single item.

Funnel chart

Description automatically generated with medium confidence

**Example: A structure for pet adoption**

struct PetData

{

string petName; //store pet's name

string species; //store species

double ageInYears; //store age in years

char genderMorF; //store gender as 'M' or 'F'

double weightInPounds; //store weight in pounds

int cageLocation; //store location in shelter

bool isHealthy; //true/false whether the pet is healthy

bool isAvailable; //true/false whether the pet is available

double adoptionFee; //adoption fee in dollars

};

**How to declare and use structs**

Diagram

Description automatically generated

**How to use struct variables:**

Once you declare your struct, you can declare variables of structure type.

structName structVariableName;

Example: We can declare more variables of type PetData using:

PetData pet1;

PetData pet2;

**Important rules for struct**

* Names begin with \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ letters. (\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_helps us tell the struct apart from a normal variable)
* Usually place the struct near the beginning of the \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_, similar to when declaring global constants.
* After it is declared, the struct is available to every function that appears \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ the definition.
* Once you declare it, the \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ can be used just like regular old variables.

**Additional rules**

* The \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_of a struct is the same as any other variable.
* Structs are \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_to the function in which they are declared.
* If declared globally, structs are available to all functions \_\_\_\_\_\_\_\_\_\_\_\_\_\_ the declaration.
* Structs are another data type, similar to an int or double. The difference is that \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ what goes inside a struct.
* You can use structures anywhere your favorite data types are used (including in functions, arrays, vectors and pointers)

**Using structures in programs**

Diagram

Description automatically generated with low confidence

**Key terms to remember:**

**Structure tag** = name of the structure type. First letter is \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_. Otherwise follows usual variable rules.

**Member names** = items \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ inside struct (inside braces).

**More Key terms:**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ value = set of data made using a structure (pet1, pet2)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ value = that structure’s values

**The Dot operator and Using data in structures**

Use the \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_operator to refer to members of struct variables:

structVariableName.memberVariabletoget

Examples: pet1.name = “Fluffy”; //get name from pet1

**Demo: Employee Pay**

We can use structs to store name, hours worked, pay rate, and gross pay for each employee.

#include <iostream>  
#include <string>  
using namespace std;  
  
struct Employee  
{  
 int empID;  
 string name;  
 double hours;  
 double payRate;  
 double grossPay;  
};  
  
Employee getNewHireData();  
*//Precondition: Employee struct is declared  
//Postcondition: return an employee type variable holding data entered by the user*int main()  
{  
 Employee myEmployees[100];  
  
 *//call getNewHireData to enter the data for employee 0* myEmployees[0] = getNewHireData();  
  
  
 *//output data to screen* cout << myEmployees[0].empID << endl;  
 cout << myEmployees[0].name << endl;  
 cout << myEmployees[0].hours << endl;  
 cout << myEmployees[0].payRate << endl;  
 cout << myEmployees[0].grossPay << endl;  
  
 return 0;  
  
}  
  
Employee getNewHireData()  
{  
 Employee newHire;  
 cout << "Enter the new employee's ID: " << endl;  
 cin >> newHire.empID;  
  
 cout << "Enter the new employee's name: " << endl;  
 cin >> newHire.name;  
  
 cout << "Enter the new employee's hours: " << endl;  
 cin >> newHire.hours;  
  
 cout << "Enter the new employee's payRate: " << endl;  
 cin >> newHire.payRate;  
  
 newHire.grossPay = newHire.hours \* newHire.payRate;  
  
 return newHire;  
}

**Example: Employee Pay**

//This program demonstrates use of structures

#include <iostream>

#include <string>

#include <iomanip>

using namespace std;

struct PayRoll

{

int empNumber;//employee ID

string name;//employee Name

double hours;//hours worked

double payRate;//hourly pay rate

double grossPay;//Gross pay

};

int main()

{

PayRoll employee; //declare PayRoll structure called employee

//get the employee ID number

cout << "Enter the employee's number: ";

cin >> employee.empNumber;

//get the employee's name

cout << "Enter the employee's name: ";

cin.ignore(); //ignore the '\n' at the end of the name

getline(cin, employee.name);

//get the hours worked

cout << "How many hours did the employee work? ";

cin >> employee.hours;

//Get the employee's pay rate

cout << "What is the employee's hourly pay rate? ";

cin >> employee.payRate;

//Calculate the employee's gross pay

employee.grossPay = employee.hours \* employee.payRate;

//Display the employee data

cout << "Here is the employee's payroll data: \n";

cout << "Name: " << employee.name << endl;

cout << "Hours worked: " << employee.hours << endl;

cout << "Pay Rate: $" << employee.payRate << " per hour." << endl;

cout << "Gross Pay: $" << employee.grossPay << endl;

return 0;

}

**Additional rules about structs: Duplicate variable names**

* Member variables can be duplicated between different structure types.
* Variables with duplicate names in different structures are treated as \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_variables in \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ locations.

**Additional rules: Structs containing more Structs**

* Structs can contain member variables which are also \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Additional Rules: Alternate Method for Initializing Structs**

* A structure can be \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ when declared

Date birthday = {1, 31, 2001};

**Additional rules: using a struct in a struct**

* Structs can contain other structs

struct RGBpixel

{

int redValue;

int greenValue;

int blueValue;

};

struct PPMimage

{

string header;

int width;

int height;

int maxValue;

vector <RGBpixel> pixels;

};

**Additional Rules: using struct in arrays, vectors, functions**

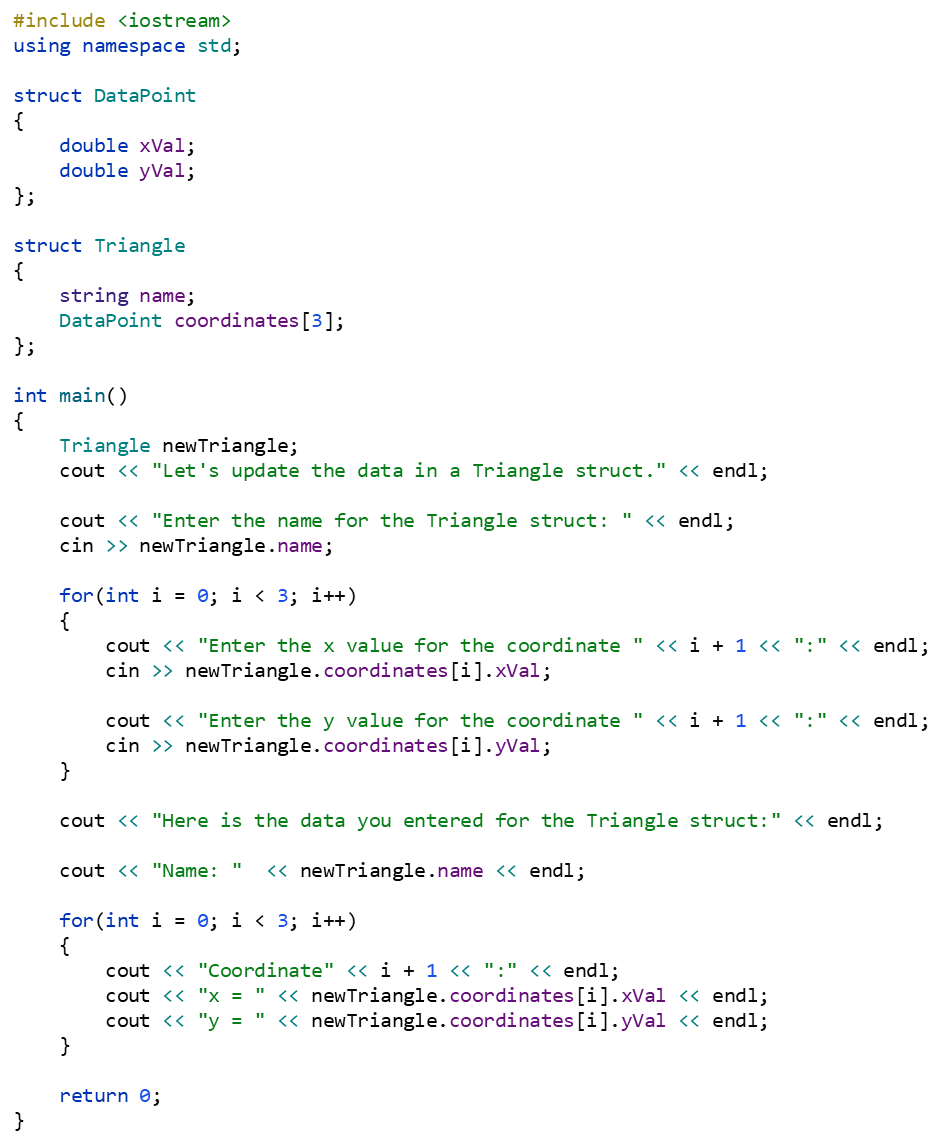
PetData adoptionData[NUM\_PETS];

//makes an array called adoptionData containing //[NUM\_PETS] sets of PetData

vector <PetData> adoptionData;

//makes a vector containing sets of PetData

**Example: Using DataPoint and Triangle structs**

****

**Additional rules: = with Structures**

* The \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ operator works\* with struct!
* Setting one struct \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_to another struct will assign all\* member variables the corresponding values of the struct it is set equal to.
* **\*CAUTION:** Be very careful if your struct contains \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ or \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ as members. The = will copy the \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ address and each struct will share the same memory address.

**Recap so far: Structures**

Declaring the structure or struct data type is very useful!Convenient to use when you have variables which you want to keep as a \_\_\_\_\_\_\_\_\_\_\_\_\_\_

**The problem with structures**

**Two main limitations of structs:**

1. **structures are used to store data. They are not used to store functions.**

If we need to make changes, we have to modify BOTH structures AND the functions in our program!

Especially in massive programs with tons of data and functions, it can be a big problem to have data and functions kept separate.

\*In the C programming language, structs could only store data and could not store functions. In C++ a struct can contain both variables and functions. But conventionally, programmers generally use structs to store sets of data and classes when we need to store both functions and data.

**2. Everything is public within a struct (there is no data protection)**

Sometimes we don’t want all parts of a program to be able to access certain data all the time. Examples:

* Programmer in industry working on a program with a big team.
  + Do you want your teammates’ code to be able to modify your data without restriction?
* Programs storing important data
  + we want to have protections in place so we don’t accidentally modify data
  + ensure updates are made properly without corrupting other data or calculations.

**How to improve a struct**

A class is basically an improved struct with more capabilities:

* **Encapsulation:** we can put both data and functions together to make self-contained, modular programs.
* **Data hiding:** we can declare some items to be public and other items to be private to help protect data.

An object is an instance of a class, in the same way that a variable can be an instance of a struct