Kyle @KylePlantEmoji · 23 godz.

Me: I'm so sorry, my dog ate my homework

Comp Sci Professor: your dog ate your coding assignment?

Me:

Prof:

Me: it took him a couple bytes

255 18,7K 146K

CS1428
Foundation of Computer Science

Lecture 9: Structs

Data Types in Programming:

- Any value stored in memory is represented using a binary number.
- The data type informs the compiler how much memory is needed, and how to represent fetched values.
- But why can’t we tell the compiler to allocate more or less memory, and interpret fetched values using a different scheme?
- We *can* actually define new data types in our program.

structs

- By putting together primitive data types we can create new structures.
- The **struct** keyword is used to defines these structures.
- A **struct** contains one or more primitive variables, each of which has its own name. These are the “members” of the **struct**.
- We can use the dot . operator to reference the individual elements of our new data type.

structs

```
struct Dog{
    string name;
    string breed;
    int age;
}; //remember this semicolon!

int main() {
   Dog myDog;
   myDog.name = "Ruffles";
   myDog.breed = "Chiweenie";
   cout << myDog.name << " is a " << myDog.breed << endl;
    return 0;
}
```

structs

```
/home/gentry/Desktop/1428_testDir/junk
Ruffles is a Chiweenie

Process returned 0 (0x0)   execution time : 0.002 s
Press ENTER to continue.
```

Initializing structs

```
struct Dog{
    string name;
    string breed;
    int age;
}; //remember this semicolon!

int main() {
   Dog myDog = {"Fluffles", "Teacup Mastiff", 4};
   cout << myDog.name << " is a " << myDog.breed << endl;
    return 0;
}
```

Initializing structs

```
/home/gentry/Desktop/1428_testDir/junk
Fluffles is a Teacup Mastiff

Process returned 0 (0x0)   execution time : 0.001 s
Press ENTER to continue.
```

Passing structs

```
struct Dog{
    string name;
    string breed;
    int age;
}; //remember this semicolon!

void printDog(const Dog& d){ //by reference!
    cout << d.name << " is a " << d.breed << endl;
}

int main() {
   Dog myDog = {"Fluffles", "Teacup Mastiff", 4};
   printDog(myDog);
    return 0;
}
```

Passing structs

```
/home/gentry/Desktop/1428_testDir/junk
Fluffles is a Teacup Mastiff

Process returned 0 (0x0)   execution time : 0.001 s
Press ENTER to continue.
```

Returning structs

```
struct Dog{
    string name;
    string breed;
    int age;
};

Dog makeDog(string n, string b, int a){
    Dog d = {n, b, a};
    return d;
}

int main() {
    Dog myDog = makeDog("Floppy", "Laso Apso", 12);
    cout << "This dog is named: " << myDog.name;
    return 0;
}
```

Returning structs

```
/home/gentry/Desktop/1428_testDir/junk
This dog is named: Floppy
Process returned 0 (0x0)   execution time : 0.002 s
Press ENTER to continue.
```

structs with functions

- **structs** can be passed to or returned from functions.
- Passing a **struct** to a function should be done by reference whenever possible.
- Returning a **struct** from a function is a trick that can be used to return more than one value from a function.
- Remember that **struct** is not a data type, it's our tool to *make* new data types.

A Common Pitfall

```
struct Dog{
    string name;
    string breed;
    int age;
};

//Don't do this!!!!!!
struct makeDog(string n, string b, int a){
    Dog d = {n, b, a};
    return d;
}
```

Arrays of structs

```
struct Point{
    float x, y;
};


int main() {
    Point drawing[100];
    for(int i = 0; i < 100; i++){
        cout << "Please enter an X-Y cordinate: ";
        cin >> drawing[i].x >> drawing[i].y;
    }
    return 0;
}
```

struct of structs

```
struct Point{
    float x, y;
};

struct Line{
    Point start, finish;
};


int main() {
    Line doodle;
    cout << "Enter the XY coordinate of the start: ";
    cin >> doodle.start.x >> doodle.start.y;
    cout << "Enter the XY coordinate of the finish: ";
    cin >> doodle.finish.x >> doodle.finish.y;
    return 0;
}
```

struct of Arrays

```
struct Student{
    string name;
    string ID_num;
    float grades[10];
};

int main() {
    Student me;
    me.name = "Gentry";
    for(int i = 0; i < 10; i++){
        cout << "Enter one grade for " << me.name << ": \n";
        cin >> me.grades[i];
    }
}
```

Constructors for structs

- We can define a special function along with a **struct** called a **constructor**.
- This function runs once when the **struct** is instantiated.
- The constructor has the same name as the **struct** and does not have a return.

Constructor

```
struct Point{
    float x, y;
    Point(int new_x, int new_y){
        x = new_x;
        y = new_y;
    }
};

int main() {
    Point myPoint(3, 7);
    cout << "My point X:" << myPoint.x << " Y:" << myPoint.y << endl;
    return 0;
}
```

Constructor

```
/home/gentry/Desktop/1428_testDir/junk
My point X:3 Y:7

Process returned 0 (0x0)   execution time : 0.002 s
Press ENTER to continue.
```

Other Data Types in C++

- **structs** were the most common way to create new data types in C, although C++ has added **classes**.
- There are other ways to create data types in C++:
  - **typedef: rename a primitive data type**
  - **enum: create named indexes**
  - **union: a collection of variables with only one being used**

typedef

```
typedef long int Number;

int main() {
    Number myNumber = 5;
    cout << "My number is: " << myNumber << endl;
    return 0;
}
```

typedef

```
/home/gentry/Desktop/1428_testDir/junk
My number is: 5

Process returned 0 (0x0)   execution time : 0.002 s
Press ENTER to continue.
```

enum

```
enum Color{Red, Blue, Green};

int main() {
    Color house = Red;
    switch(house){
        case Red:
            cout << "Red is " << Red <<endl;
            break;
        default:
            cout << "Not Red"<<endl;
    }
    return 0;
}
```

enum

```
/home/gentry/Desktop/1428_testDir/junk
Red is 0

Process returned 0 (0x0)   execution time : 0.002 s
Press ENTER to continue.
```

union

```
union generic_variable{
    int number;
    char character;
};

int main() {
    generic_variable a, b;
    a.number = 2;
    b.character = 'c';
    cout << "a is " << a.number << " and b is " << b.character << endl;
    return 0;
}
```

union

```
/home/gentry/Desktop/1428_testDir/junk

a is 2 and b is c

Process returned 0 (0x0)   execution time : 0.002 s
Press ENTER to continue.
```

union Only Holds One Value!

```
union generic_variable{
    int number;
    char character;
};

int main() {
    generic_variable a, b;
    a.number = 2;
    //This overwrites the previous value!!!
    a.character = 'd';
    cout << "a.number is " << a.number << " and a.character is "
        << a.character << endl;

    return 0;
}
```

union Only Holds One Value!

```
/home/gentry/Desktop/1428_testDir/junk
a.number is 100 and a.character is d

Process returned 0 (0x0)   execution time : 0.002 s
Press ENTER to continue.
```

When to Use a struct

- When you want to logically encapsulate a collection of related values.
- When you need to be able to return a collection of values from a function.
- When you want to pass several values to a function as a single, logically related package.
- When you need to remind readers of your code that groups of values are related.