سلسلة تعلم البرمجة بلغة C++ الحديثة

Learn Modern C++ Programming Course

إعداد المهندس أحمد الديب

#10: Structures

User-Defined Types

- struct (a structure) is a sequence of elements (called members) of arbitrary types.
- union is a struct that holds the value of just one of its elements at any one time.
- enum (an enumeration) is a type with a set of named constants (called enumerators).
- enum class (a scoped enumeration) is an enum where the enumerators are within the scope of the enumeration and no implicit conversions to other types are provided.

Source: The C++ Programming Language (4th Edition), Bjarne Stroustrup

Structures

- Variables of type Address can be declared exactly like other variables, and the individual members can be accessed using the . (dot) operator.
- Variables of struct types can be initialized using the {} notation.
- Two structs are different types even when they have the same members.

```
struct Address {
 const char* name;
 int number;
 const char* street;
 const char* town;
 char state[2];
 const char* zip;
};

int main() {
 Address jd;
 jd.name = "Jim Dandy";
 jd.number = 61;

 Address jd2 = {"Jim Dandy", 61, "South St",
  "New Providence", {'N', 'J'}, "07974"};
}
```

Source: The C++ Programming Language (4th Edition), Bjarne Stroustrup

Structures

- Structures are often accessed through pointers using the –> (struct pointer dereference) operator.

- When p is a pointer, p–>m is equivalent to (*p).m.

```
void print_addr(Address* p) {
  std::cout << p->name << '\n'
            << p->number << ' ' << p->street << '\n'
            << p->town << '\n'
            << p->state[0] << p->state[1] << ' ' << p->zip << '\n';
}
```

Source: The C++ Programming Language (4th Edition), Bjarne Stroustrup

Structures

- struct can be passed by reference and accessed using the . (struct member access) operator.

- Objects of structure types can be assigned, passed as function arguments, and returned as the result from a function.

```
void print_addr2(const Address& r) {
  std::cout << r.name << '\n'
            << r.number << ' ' << r.street << '\n'
            << r.town << '\n'
            << r.state[0] << r.state[1] << ' ' << r.zip << '\n';
}
```

Source: The C++ Programming Language (4th Edition), Bjarne Stroustrup

Structures and Arrays

```
constexpr size_t buffer_size = 14;

struct Data {
  char buffer[buffer_size];
};

int main() {
  Data output;
  strcpy(output.buffer, "Welcome to C++");

  char input_buffer[buffer_size];
  input_buffer = output.buffer;  // error: array type 'char [10]'
                                 // is not assignable

  Data input = output;  // OK
  std::cout << input.buffer << "\n";
}
```

- Placing a built-in array in a struct allows us to treat that array as an object

Source: The C++ Programming Language (4th Edition), Bjarne Stroustrup

Structures and Classes

```
struct Address {
 public:
  Address(const std::string n, int nu, const std::string& s,
          const std::string& t, const std::string& st)
      : _name(n), _number(nu), _street(s), _town(t), _state(st){};
  void print() const;

 private:
  std::string _name;
  int _number;
  std::string _street;
  std::string _town;
  std::string _state;
};

Address jd{"Jim Dandy", 61, "South St", "New Providence", "NJ"};
jd.print();
```

- A struct is simply a class where the members are public by default. So, a struct can have member functions, constructors, etc.

Source: The C++ Programming Language (4th Edition), Bjarne Stroustrup

Fields

```
struct Status {
  unsigned int TXE : 1;   // Transmit data register empty
  unsigned int TC : 1;    // Transmission complete
  unsigned int RXNE : 1;  // Read data register not empty
  unsigned int ERR : 3;   // Error
  unsigned int : 26;      // Reserved
};

struct Data {
  unsigned int DATA : 8;  // Data
  unsigned int : 24;      // Reserved
};

struct Uart {
  Status SR;  // Status register
  Data DR;    // Data register
};
```

- Often called a bit-field. A member is defined to be a field by specifying the number of bits it is to occupy.
- Unnamed fields used to make the layout better in some machine-dependent way.
- Using fields to pack several variables into a single byte does not necessarily save space. It saves data space, but the size of the code needed to manipulate these variables increases on most machines.

Source: The C++ Programming Language (4th Edition), Bjarne Stroustrup

Thank you