OOP345:
Structures,
Bit Operators &
Bit Fields

Sukhbir Tatla

sukhbir.tatla@senecacollege.ca

Introduction

- C-like techniques
- Useful for C++ programmers working with legacy C code
- Structures
  - Hold variables of different data types
  - Similar to classes, but all data members public
  - Examine how to use structures
    - Make card shuffling simulation

Structure: Definition

- Structure definition

```
struct Card {
    char *face;
    char *suit;
};
```

- Keyword **struct**
- **Card** is structure name
  - Used to declare variable of structure type
- Data/functions declared within braces
  - Structure members need unique names
  - Structure cannot contain instance of itself, only a pointer
- Definition does not reserve memory
- Definition ends with semicolon

Structure: Definition

- Declaration
  - Declared like other variables: use structure type in main() function
    - **Card oneCard, deck[52], *cPtr;**

    OR
  - Can declare variables when define structure

```
struct Card {
        char *face;
        char *suit;
} oneCard, deck[ 52 ], *cPtr;
```

Structure: Operations

- Structure Operations
  - Assignment to a structure of same type
  - Taking address (&)
  - Accessing members (oneCard.face)
  - Using sizeof
    - Structure may not be in consecutive bytes of memory
    - Byte-alignment (2 or 4 bytes) may cause "holes"

Structure: Initialize

- Initializer lists (like arrays)
  - **Card oneCard = { "Three", "Hearts" };**
  - Comma-separated values, enclosed in braces
    - If member unspecified, default of 0
- Initialize with assignment
  - Assign one structure to another
    - **Card threeHearts = oneCard;**
  - Assign members individually
    - **Card threeHearts;**
    - **threeHearts.face = “Three”;**
    - **threeHearts.suit = “Hearts”;**

Structures with Functions

- Two ways to pass structures to functions
  - Pass entire structure
  - Pass individual members
  - Both pass call-by-value
- To pass structures call-by-reference
  - Pass address
  - Pass reference to structure
- To pass arrays call-by-value
  - Create structure with array as member
  - Pass the structure
  - Pass-by-reference more efficient

TYPEDEF

- Keyword typedef
  - Makes synonyms (aliases) for previously defined data types
    - Does not create new type, only an alias
  - Creates shorter type names
- Example
  - typedef Card *CardPtr;
    - Defines new type name CardPtr as synonym for type Card *
  - CardPtr myCardPtr;
  - Card * myCardPtr;
-

Structures: Program Example

```
#include <iostream>
#include <iomanip>
using namespace std;
   struct student  {
        char name[50];
        int stid;
        float marks;
   } s[10];
```

Structures: Program Example Cont'd

```
int main()
{
        cout << "Enter information of students: " << endl;
        // storing information
        for (int i = 0; i < 10; ++i)
        {
                s[i].stid = i + 1;
                cout << "For Student ID: " << s[i].stid << "," << endl;
                cout << "Enter Student Name: ";
                cin >> s[i].name;
                cout << "Enter Student Marks: ";
                cin >> s[i].marks;
                cout << endl;
        }
```

Structures: Program Example Cont’d

```
int main () Cont’d:
        cout << "Displaying Information: " << endl;
        // Displaying information
        for (int i = 0; i < 10; ++i)
        {
                cout << "\nStudent ID: " << i + 1 << endl;
                cout << "Student Name: " << s[i].name << endl;
                cout << "Student Marks: " << s[i].marks << endl;
        }
        return 0;
  }     // End of main() function
```

Bitwise Operators

- Data represented internally as sequences of bits
  - Each bit can be 0 or 1
  - 8 bits form a byte
    - char is one byte
    - Other data types larger (int, long, etc.)
- Low-level software requires bit and byte manipulation
  - Operating systems, networking

Bitwise Operators Cont’d

- Bit operators
  - Many are overloaded
  - & (bitwise AND)
    - 1 if both bits 1, 0 otherwise
  - | (bitwise inclusive OR)
    - 1 if either bit 1, 0 otherwise
  - ^ (bitwise exclusive OR (XOR))
    - 1 if exactly one bit is 1, 0 otherwise
    - Alternatively: 1 if the bits are different
  - ~ (bitwise one's complement (NOT))
    - Flips 0 bits to 1, and vice versa

Bitwise Operators Cont’d

- Bit operators
  - << (left shift)
    - Moves all bits left by specified amount
    - Fills from right with 0
  - >> (right shift with sign extension)
    - Moves bits right by specified amount
    - Fill from left can vary

Bitwise Operators: Example

- Next program
  - Print values in their binary representation
  - Example: unsigned integer 3
    - **00000000 00000000 00000000 00000011**
    - (For a machine with 4-byte integers)
    - Computer stores number in this form
- Using masks
  - Integer value with specific bits set to 1
  - Used to hide some bits while selecting others
  - Use with AND

Bitwise Operators: Example Cont’d

- Mask example
  - Suppose we want to see leftmost bit of a number
  - AND with mask
    - **10000000 00000000 00000000 00000000 (mask)**
    - **10010101 10110000 10101100 00011000 (number)**
  - If leftmost bit of number 1
    - Bitwise AND will be nonzero (true)
      - Leftmost bit of result will be 1
    - All other bits are "masked off" (ANDed with 0)
  - If leftmost bit of number 0
    - Bitwise AND will be 0 (false)

Bitwise Operators: Example Cont'd

- To print every bit
  - Print leftmost digit
  - Shift number left
  - Repeat
- To create mask
  - Want mask of 1000000 … 0000
  - How many bits in unsigned?
    - **sizeof(unsigned) * 8**
  - Start with mask of 1
    - Shift one less time (mask is already on first bit)
    - **1 << sizeof(unsigned) * 8 - 1**
    - **10000000 00000000 00000000 00000000**

Bitwise Operators: Program Example

```
#include <iostream>
#include <iomanip>
using namespace std;
int add(int, int);
void main()
{
        int a = 9, b = 8;
        cout << (a>>1)<< endl;
        int x, y;
        cout << "Enter the numbers to add: ";
        cin >> x >> y;
        cout << "The Result is: " << add(x, y);
        cout << endl << endl;
}
```

Bitwise Operators: Program Example Cont'd

```
//Function defined to add the values using Bitwise AND and XOR.
  int add(int x, int y)
  {
        int carry;
        while (y != 0)
        {
              carry = x & y;
              x = x^y;
              y = carry << 1;
              //cout << "Y = " << y;
        }
        return x;
 }
```

Bit Fields

- Bit field
  - Member of structure whose size (in bits) has been specified
  - Enables better memory utilization
  - Must be declared int or unsigned
  - Example:

```
Struct BitCard {
  unsigned face : 4;
  unsigned suit : 2;
  unsigned color : 1;
};
```

  - Declare with name : width
    - Bit width must be an integer

Bit Fields Cont’d

- Accessing bit fields
  - Access like any other structure member

```
Struct BitCard {
  unsigned face : 4;
  unsigned suit : 2;
  unsigned color : 1;
};
```

  - myCard.face = 10;
    - face has 4 bits, can store values 0 - 15
    - suit can store 0 - 3
    - color can store 0 or 1

Bit Fields Cont'd

- Other notes
  - Bit fields are not arrays of bits (cannot use [])
  - Cannot take address of bit fields
  - Use unnamed bit fields to pad structure

```
Struct Example {
        unsigned a : 13;
        unsigned   : 3;
        unsigned b : 4;
};
```

- Use unnamed, zero-width fields to align to boundary

```
Struct Example {
        unsigned a : 13;
        unsigned   : 0;
        unsigned b : 4;
};
```

- Automatically aligns b to next boundary

Thank You!

Any Questions Please?