ADVANCED PROGRAMMING
(BETC 1353)

WEEK 6: UNIONS, BIT MANIPULATIONS AND ENUMERATIONS

AIMAN ZAKWAN BIN JIDIN
aimanzakwan@utem.edu.my

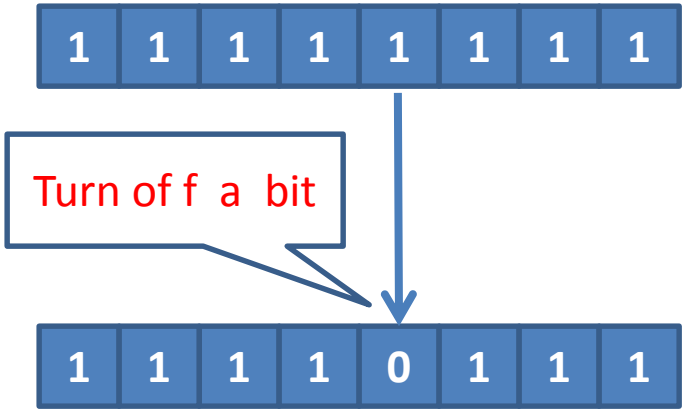
Learning Outcomes

At the end of this session, you should be able:

- To manipulate data using bitwise operators
- To use bit fields in C++ programs
- To use enumeration constants in C++ programs
- To use unions in C++ programs

Bitwise Operators

- Operators that are useful for manipulating bits of data
- Examples of manipulating bits:
  - Get a bit in a byte
  - Turn ON/OFF one or several bits



Bitwise Operators

Operator	Description
&	Perform **AND** operation on every single bit (the same position) of the operands.
|	Perform **OR** operation on every single bit (the same position) of the operands.
^	Perform **EXCLUSIVE OR (XOR)** operation on every single bit (the same position) of the operands.
~	Complement operator which inverts all the bit of the operand.
<<	Shift left operator that shifts the bits of the data one or several positions to the left. The bits shifted off the left side are removed.
>>	Shift rightoperator that shifts the bits of the data one or several positions to the right. The bits shifted off the right side are removed.

AND Bitwise Operator

```
operand1 & operand2
```

- Performing AND operation on each bit of the same position of both operands.
- Example:

```
a = 82;                a = 0101 0010
b = 155; → &           b = 1001 1011
c = a&b;               c = 0001 0010
```

OR Bitwise Operator

```
operand1 | operand2
```

- Performing AND operation on each bit of the same position of both operands.
- Example:

```
a = 82;                a = 0101 0010
b = 155; → |           b = 1001 1011
c = a|b;               c = 1101 1011
```

XOR Bitwise Operator

```
operand1 | operand2
```

- Performing XOR operation on each bit of the same position of both operands.
- Example:

```
a = 82;                a = 0101 0010
b = 155; → ^           b = 1001 1011
c = a^b;               c = 1100 1001
```

Complement Bitwise Operator

```
~ operand
```

- Inverting all bits of the `operand`:
  - 1 become 0
  - 0 become 1
- Example:

```
a = 72;  →   a = 0100 1000
b = ~a;  →   b = 1011 0111 (183)
```

Shift Left Bitwise Operator

`operand << n`

- Shifting all bits in `operand` to the left by `n` positions.
- The bit(s) shifted off the left side are removed.
- Example:

```
a = 70;      →  a = 0100 0110
b = a<<3;    →  b = 0011 0000
```

Shift Right Bitwise Operator

```
operand >> n
```

- Shifting all bits in `operand` to the right by `n` positions.
- The bit(s) shifted off the right side are removed.
- Example:

```
a = 70;      →  a = 0100 0110
b = a>>2;    →  b = 0001 0001
```

Examples of Bit Manipulations

81 & 99 ?

```
0101 0001 <- 81
0110 0011 <- 99
___________ &
0100 0001 -> 65
```

81 | 99 ?

```
0101 0001 <- 81
0110 0011 <- 99
___________ |
0111 0011 -> 115
```

81 ^ 99 ?

```
0101 0001 <- 81
0110 0011 <- 99
___________ ^
0011 0010 -> 50
```

Try Yourself!

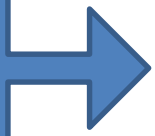
bitwise.cpp

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

int main()
{
    unsigned char x = 81;
    unsigned char y = 99;
    unsigned char z;

    z = x & y;
    cout << x << " & " << y << " = "
         << (int) z << endl;

    z = x | y;
    cout << x << " | " << y << " = "
         << (int) z << endl;

    z = x ^ y;
    cout << x << " ^ " << y << " = "
         << (int) z << endl;

    return 0;
}
```

Inverting Bits

char x = 138;

~x?



1000 1010 <- 138

~

0111 0101 -> 117

Try Yourself!

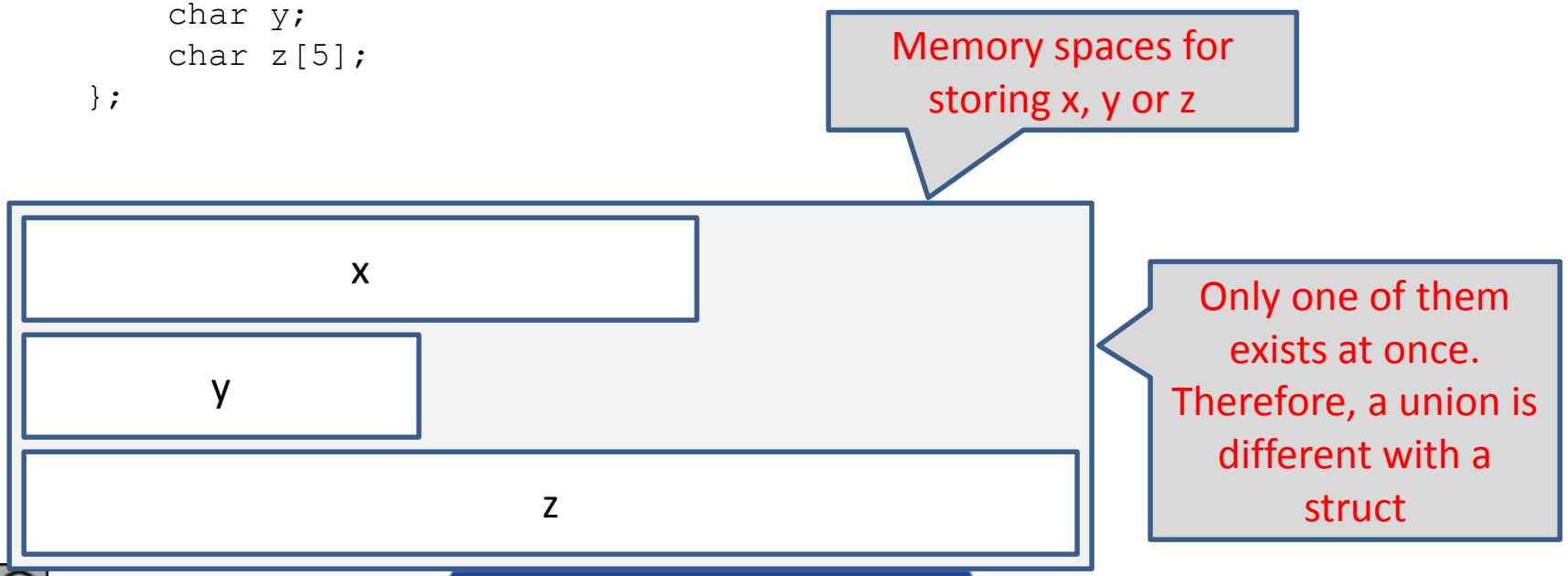
complem.cpp

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

int main()
{
    char x = 138;
    char y;

    y = ~x;

    cout << (int) y << endl;

    return 0;
}
```

Unions

- Lets programmers share memory spaces for several kinds of data
- For example:

```
union data
{
    short x;
    char y;
    char z[5];
};
```



Try Yourself!

union.cpp

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

int main()
{
    union data
    {
        short x;
        char y;
        char z[5];
    };

    data d;

    d.x = 12345;
    cout << d.x << endl; // Ok
    cout << d.y << endl; // ????
    cout << d.z << endl; // ????
    cout << "---------" << endl;

    d.y = 'A';
    cout << d.x << endl; // ????
    cout << d.y << endl; // Ok
    cout << d.z << endl; // ????
    cout << "---------" << endl;

    strcpy(d.z, "test");
    cout << d.x << endl; // ????
    cout << d.y << endl; // ????
    cout << d.z << endl; // Ok
    cout << "---------" << endl;

    return 0;
}
```

Only one field in the union contains a right data!

Bit-field

- Lets programmers define how many bits will be used by a field to store the data
- Using struct to define bit-fields
- For example:

```
struct bits
{
    unsigned bit0: 1;
    unsigned bit1: 1;
    unsigned bit2: 1;
    unsigned bit3: 1;
    unsigned bit4: 1;
    unsigned bit5: 1;
    unsigned bit6: 1;
    unsigned bit7: 1;
};
```

Definition of a bit-field

Bit-field

- The use of a bit-field is related to a union
- For example:

```
union bit_data
{
    unsigned char data;
    bits byte;
};
```

- By using that union, `byte` and `data` utilize the same memory. Therefore, bit information can be accessed from `byte` and a byte data can be sent through `data`

Try Yourself!

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

struct bits
{
    unsigned bit0: 1;
    unsigned bit1: 1;
    unsigned bit2: 1;
    unsigned bit3: 1;
    unsigned bit4: 1;
    unsigned bit5: 1;
    unsigned bit6: 1;
    unsigned bit7: 1;
};

union bit_data
{
    unsigned char data;
    bits byte;
};

void display_bits(bit_data b);

int main()
{
    bit_data number;
    number.data = 65;
    cout << (int) number.data
         << " -> ";
    display_bits(number);

    number.data = 97;
    cout << (int) number.data << " -> ";
    display_bits(number);

    // Turn on bit 3
    number.byte.bit3 = 1;
    cout << "After bit 3 was turned on : ";
    display_bits(number);

    // Turn on bit 6
    number.byte.bit6 = 0;
    cout << "After bit 6 was turned off: ";
    display_bits(number);

    return 0;
}

void display_bits(bit_data b)
{
    cout << b.byte.bit7;
    cout << b.byte.bit6;
    cout << b.byte.bit5;
    cout << b.byte.bit4;
    cout << b.byte.bit3;
    cout << b.byte.bit2;
    cout << b.byte.bit1;
    cout << b.byte.bit0;

    cout << endl;
}
```

Try to understand each code. Then, you will get the benefit of bit-field for some applications.

bitfield.cpp

Enum Constants

- Named constants that are stored as a list of symbols

- For example:

```
enum Day { MONDAY, TUESDAY, WEDNESDAY,
           THURSDAY, FRIDAY, SATURDAY,
           SUNDAY };
```

Based on that definition, MONDAY = 0, TUESDAY = 1, WEDNESDAY = 2 and so on

- Other example:

```
enum Month { JAN = 1, FEB, MAR, APR, MAY, JUN,
             JUL, AUG, SEP, OCT, NOV, DEC };
```

Based on that definition, JAN = 1, FEB = 2, MAR = 3 and so on

Enum variables

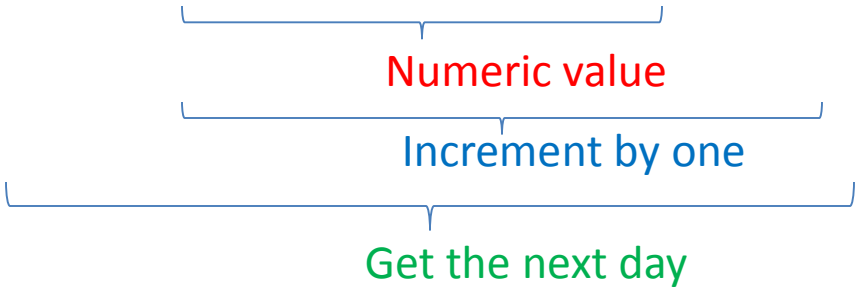
- An enum variable is a variable declared by an enum type
- The variables can only store a symbol defined in an enum type
- For example:

```
enum Day working_day;

working_day = MONDAY;   // OK
working_day = Day(0);   // OK
working_day = 0;        // Not OK
```

Processing Enum Variables

- To the next day:

```
work_day = Day((int) work_day + 1);
```



- Converting from enum type to string:

```
string dayName(enum Day d)
{
    string tmpDay;

    switch(d)
    {
    case MONDAY:
       tmpDay = "Monday";
       break;
    case TUESDAY:
       tmpDay = "Tuesday";
       break;
    ...
    case SUNDAY:
       tmpDay = "Sunday";
       break;
    }

    return tmpDay;
}
```

Try Yourself!

enum.cpp (part 1)

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

using namespace std;

enum Day { MONDAY, TUESDAY, WEDNESDAY,
           THURSDAY, FRIDAY, SATURDAY,
           SUNDAY };

string dayName(enum Day d); // Prototype

int main()
{
    enum Day work_day;

    cout << "Working days:"<< endl;
    work_day = MONDAY;

    while (work_day <= FRIDAY)
    {
        cout << "*** " << dayName(work_day) << endl;

        // To the next day
        work_day = Day((int) work_day + 1);
    }

    return 0;
}
```

Please continue to the next page.

enum.cpp (part 2)

```
string dayName(enum Day d)
{
    string tmpDay;

    switch(d)
    {
    case MONDAY:
       tmpDay = "Monday";
       break;
    case TUESDAY:
       tmpDay = "Tuesday";
       break;
    case WEDNESDAY:
       tmpDay = "Wednesday";
       break;
    case THURSDAY:
       tmpDay = "Thursday";
       break;
    case FRIDAY:
       tmpDay = "Friday";
       break;
    case SATURDAY:
       tmpDay = "Saturday";
       break;
    case SUNDAY:
       tmpDay = "Sunday";
       break;
    }

    return tmpDay;
}
```

Self-Review Questions

Question 1

Given

```
var1 = 140;     // 1000 1100
```

a) Write C++ statements to change bit 3 of `var1` to 0.

b) Write C++ statements to change bit 5 of `var1` to 1.

c) What is the value in decimal for `var2`: `var2 = ~var1;`

Self-Review Questions

Answers:

```
a)    b = 247;                  // 1111 0111
      var1 = var1 & b;          // 1000 0100

b)    c = 32;                   // 0010 0000
      var1 = var1 | c;          // 1010 1100

c)    var2 = ~var1;             // 0111 0011
                                → 115
```

Self-Review Questions

Question 2

Determine the value for each of the members of enumeration `color`, which has been declared as follows:

```
enum color = { RED = 1, GREEN, BLUE, YELLOW = 7,
               BLACK, PURPLE, WHITE = 16, ORANGE};
```

Self-Review Questions

Answers:

```
RED = 1
GREEN = 2
BLUE = 3
YELLOW = 7
BLACK = 8
PURPLE = 9
WHITE = 16
ORANGE = 17
```

Self-Review Questions

Question 3

Based on the following statements, determine TRUE or FALSE.

```
union myUnion
    {
        int  m;
        char n;
        float o;
    };
myUnion x;
```

a) Size of union `myUnion` is 4 bytes.

b) If `x.m = 123` is executed, `cout << d.o` will display 123.

Self-Review Questions

Answers:

a) **TRUE**, because it depend on the size of its largest members (`int` or `float`)

b) **FALSE**, because only one field in `union d` contains right data, which is `m`. The data is not valid for field `n` and `o`.