系统程序设计作业Unit2

2014302580341 卓越二班 余璞轩

练习1

- 源代码

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

int main()
{
    int oh_my_word = 7;

    if (oh_my_word & 0x00000004) { cout << "third bit set" << e
ndl; }

    // 0xFFFFFFFB, or 1111 1111 1111 1111 1111 1111 1111 1011
    oh_my_word = oh_my_word & 0xFFFFFFFB;
    cout << oh_my_word << endl;

    // 0x00000004, or 0000 0000 0000 0000 0000 0000 0000 0100
    oh_my_word = oh_my_word | 0x00000004;
    cout << oh_my_word << endl;

    // 0x00000004, or 0000 0000 0000 0000 0000 0000 0000 0100
    oh_my_word = oh_my_word ^ 0x00000004;
    cout << oh_my_word << endl;

    // Get bit N from a word (right-most is bit 0).
    // First, shift bit N to right-most place:
    int temp = oh_my_word >> 2;
    cout << temp << endl;

    // Second, mask the right-most bit:
    temp = temp & 0x00000001;
    cout << temp << endl;
}
```

- 结果为：

```
third bit set
3
7
3
0
0
```

7的二进制值是0000 0000 0000 0000 0000 0000 0000 0111，转换为十六进制是0x00000007。

- 因为二进制下第三位是1，所以与0100取与会得出1，第一句话会被输出
- 7与0xFFFFFFFB按位取与，最高的28位都为0，0111 & 1011 = 0011 = $3_{(10)}$
- 3与0100按位取或，最高的28位都为0，0011 | 0100 = 0111 = $7_{(10)}$
- 7与0100按位异或，最高的28位都是0，0111 ^ 0100 = 0011 = $3_{(10)}$
- 想要得到第N=3位
  - 0011 = $3_{(10)}$向右移2两位变成0000 = $0_{(10)}$
  - 0000再与0001按位取与得到0000 = $0_{(10)}$，即0011 = $3_{(10)}$的第三位是0

练习2

- 源代码

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

int handle_overflow() {
    cout << "Over-flow occurs!" << endl;
    return 0;
}

int main() {
    unsigned long x, y, sum;
    y = -1;
    sum = x + y;

    // If overflow occurred, sum will be smaller
    // than either x or y. Otherwise, sum will
    // be greater than either x or y.
    if (sum < x) { handle_overflow(); }
```

```
    return 0;
}
```

- 结果为：

```
Over-flow occurs!
```

- 结果表明：
  - –1转成unsigned long 类型会产生溢出。

练习3

1.

- 源代码

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

int main() {
    float x = 1.2F;
    double y = x;
    cout << setprecision(20) << x << ", " << y << endl;
    cout << "1.2F == 1.2: " << (1.2F == 1.2) << endl;
    return 0;
}
```

- 结果为：

```
1.2000000476837158203, 1.2000000476837158203
1.2F == 1.2: 0
```

- 结果表明：
  - double和float的前20位都一样；

- 1.2F和1.2不相等。

2.

- 源代码

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

int main() {
    double x = 1.3;
    double y = 0.4;
    if (x + y != 1.7) {
        cout << "addition failed?" << endl;
    }
    return 0;
}
```

- 结果为：

```
addition failed?
```

- 结果表明：
  - double后事实上还有很多位，且不为0；
  - double不应该用于条件判断。

3.

- 源代码

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

const double epsilon = 0.000001;

bool about_equal(double x, double y) {
    return (x < y + epsilon) && (x > y - epsilon);
}
```

```
int main() {
    cout << "1.3 + 0.4 == 1.7: " << (1.3 + 0.4 == 1.7) << endl;
    cout << "about_equal(1.3 + 0.4, 1.7): " << about_equal(1.3
+ 0.4, 1.7) << endl;
    return 0;
}
```

- 结果为：

```
1.3 + 0.4 == 1.7: 0
about_equal(1.3 + 0.4, 1.7): 1
```

- 结果表明：
  - double型变量 `1.3 + 0.4 != 1.7`
  - 但是左右两边的误差范围在 `epsilon = 0.000001` 之内

练习4

- 源代码

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

int main() {
    double x = 1.0E160;
    x = x * x;
    cout << x << endl;
    return 0;
}
```

- 结果为： `inf`
- 结果表明：
  - double型变量在产生溢出的情况下，值会被设置为 `inf` (无限大)

练习5

1.

- 源代码

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

int main() {
    int myarray[5];
    int N = 1;
    if (myarray + N == &(myarray[N])) {
        cout << "First condition checks out!" << endl;
    }
    if (myarray[N] == *(myarray + N)) {
        cout << "Second condition checks out!" << endl;
    }
    return 0;
}
```

- 结果为

```
First condition checks out!
Second condition checks out!
```

- 结果表明：
  - 在数组首地址加上N的时候会自动加成 `N*sizeOf(array的类型)` ，而不是真的加N个字节

2.

- 源代码

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

struct mystruct {
    char a, b;
    double d;
    int i;
};
```

```
int main() {

    cout << "Align of struct: " << __alignof(struct mystruct) <
< endl;
    cout << "Size of struct: " << sizeof(struct mystruct) << en
dl;
    cout << "==============" << endl;

    mystruct s1;
    cout << (void*)(&(s1.a)) << endl;
    cout << (void*)(&(s1.b)) << endl;
    cout << &s1.d << endl;
    cout << &s1.i << endl;
    cout << "==============" << endl;

    cout << offsetof(mystruct, a) << endl;
    cout << offsetof(mystruct, b) << endl;
    cout << offsetof(mystruct, d) << endl;
    cout << offsetof(mystruct, i) << endl;

    return 0;
}
```

- 结果为

```
Align of struct: 8
Size of struct: 24
==============
0x7fff5fbff6b0
0x7fff5fbff6b1
0x7fff5fbff6b8
0x7fff5fbff6c0
==============
0
1
8
16
```

- 结果表明：
  - 编译器默认取struct中最长的类型作为对齐的数值。
  - 这里最长的类型是double，所以以8字节对齐。
  - 可以看到a和d的地址都是8的倍数

如果修改align值，改变对齐方式（以2为例）

- 源代码

```
#include <iostream>
/* 修改这里的值来改变struct对齐方式 */
#pragma pack(2)
using namespace std;

struct mystruct {
    char a, b;
    double d;
    int i;
};

int main() {

    cout << "Align of struct: " << __alignof(struct mystruct) <
< endl;
    cout << "Size of struct: " << sizeof(struct mystruct) << en
dl;
    cout << "==============" << endl;

    mystruct s1;
    cout << (void*)(&(s1.a)) << endl;
    cout << (void*)(&(s1.b)) << endl;
    cout << &s1.d << endl;
    cout << &s1.i << endl;
    cout << "==============" << endl;

    cout << offsetof(mystruct, a) << endl;
    cout << offsetof(mystruct, b) << endl;
    cout << offsetof(mystruct, d) << endl;
    cout << offsetof(mystruct, i) << endl;

    return 0;
}
```

- 结果为

```
Align of struct: 2
Size of struct: 14
==============
0x7fff5fbff6b8
```

```
0x7fff5fbff6b9
0x7fff5fbff6ba
0x7fff5fbff6c2
==============
0
1
2
10
```

- 相对应的对齐值为4的结果为

```
Align of struct: 4
Size of struct: 16
==============
0x7fff5fbff6b8
0x7fff5fbff6b9
0x7fff5fbff6bc
0x7fff5fbff6c4
==============
0
1
4
12
```