



fit@hcmus

KHOA CÔNG NGHỆ THÔNG TIN
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
227 Nguyễn Văn Cừ, Phường 4, Quận 5, TP.HCM
Điện thoại: (08) 38.354.266 – Fax: (08) 38.350.096

BÁO CÁO BÀI TẬP

HỆ THỐNG MÁY TÍNH

ĐỀ TÀI

KHẢO SÁT SỐ CHẤM ĐỘNG



Giảng viên hướng dẫn: Thầy Lê Viết Long

STT	Họ và tên	MSSV
1	Trương Tiến Anh	22120017

TP. Hồ Chí Minh, tháng 3 năm 2024

PHẦN ĐÁNH GIÁ

1. Bảng đánh giá

Yêu cầu	Mục	Trạng thái	Mức độ hoàn thành (%)
Bài 1. Chuyển số chấm động sang dãy nhị phân	Bài 1	Hoàn thành	100%
Bài 2. Chuyển dãy nhị phân sang số chấm động	Bài 2	Hoàn thành	100%
Bài 3. Dùng Bài 1, Bài 2 để khảo sát các câu hỏi	Bài 3.1	Hoàn thành	100%
	Bài 3.2	Hoàn thành	100%
	Bài 3.3	Hoàn thành	100%
Bài 4: Khảo sát các trường hợp (Viết chương trình thử nghiệm và giải thích kết quả)	Bài 4.1	Hoàn thành	100%
	Bài 4.2	Hoàn thành	100%
	Bài 4.3	Hoàn thành	100%
	Bài 4.4	Hoàn thành	100%
	Bài 4.5	Hoàn thành	100%
	Bài 4.6	Hoàn thành	100%
	Bài 4.7	Hoàn thành	100%
	Bài 4.8	Hoàn thành	100%
	Bài 4.9	Hoàn thành	100%

2. Đánh giá tổng kết

Mục đích đề tài:

- Tìm hiểu số chấm động trong các ngôn ngữ lập trình trên nền kiến trúc x86
- Hiểu rõ hơn về cách tổ chức số chấm động

Kết quả đạt được: Đã thực hiện được tất cả các yêu cầu mà bài toán đưa ra, sử dụng những kỹ năng đã được học trên lớp để vận dụng vào bài làm. Code được triển khai một cách cụ thể và rõ ràng, các thử nghiệm kết quả đã được thực hiện và giải thích kết quả một cách rõ ràng. Đánh giá tổng thể là hoàn thành tốt, với các yêu cầu của bài toán

=> Mức độ hoàn thành: 100%

KẾT QUẢ LÀM BÀI

Bài 1. Viết chương trình nhập vào số chấm động. Hãy xuất ra biểu diễn nhị phân từng thành phần(dấu, phần mũ, phần trị) của số chấm động vừa nhập

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

void dumpFloat(float* p)
{
    unsigned char* ptr = reinterpret_cast<unsigned char*>(p);
    unsigned int floatBits = 0;

    for (int i = sizeof(float) - 1; i >= 0; i--)
    {
        floatBits = (floatBits << 8) | *(ptr + i);
    }

    int sign = (floatBits >> 31) & 1;
    int exponent = (floatBits >> 23) & ((1 << 8) - 1);
    unsigned int significand = floatBits & ((1 << 23) - 1);

    cout << sign << " ";
    cout << bitset<8>(exponent) << " ";
    cout << bitset<23>(significand) << endl;
}

int main() {
    float x;

    while (true)
    {
        cout << "Nhap so cham dong (32-bit): ";
        cin >> x;

        if (cin.fail()) {
            cout << "Khong hop le" << std::endl;
            break;
        }

        cout << "Bieu dien nhi phan: ";
        dumpFloat(&x);
    }

    return 0;
}
```

Bài 2. Viết chương trình nhập vào biểu diễn nhị phân của số chấm động. Hãy xuất ra biểu diễn thập phân tương ứng.

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

void forceFloat(float* p, const char* s)
{
    int len = strlen(s);
    if (len > 32)
    {
        cout << "Do dai chuoai > 32." << endl;
        return;
    }
    for (int i = 0; i < len; ++i)
    {
        if (s[i] != '0' && s[i] != '1')
        {
            cout << "Chuoai khong hop le" << endl;
            return;
        }
    }

    unsigned int value = 0;
    for (int i = 0; i < len; ++i)
    {
        value = (value << 1) | (s[i] - '0');
    }
    memcpy(p, &value, sizeof(float));
}

void printFloat(float value)
{
    if (isinf(value))
    {
        if (value > 0)
            cout << "+ vo cung" << endl;
        else
            cout << "- vo cung" << endl;
        return;
    }
    if (isnan(value))
    {
        cout << "NaN" << endl;
        return;
    }
    cout << value << endl;
}
```

```
int main()
{
    float x;
    char binaryString[33];

    while (true)
    {
        cout << "Nhap day nhi phan (32-bit): ";
        cin >> binaryString;
        if (cin.fail())
        {
            cout << "Day khong hop le" << endl;
            break;
        }
        forceFloat(&x, binaryString);
        cout << "So cham dong: ";
        printFloat(x);
    }
    return 0;
}
```

Bài 3: Dùng 2 hàm đã viết để khảo sát các câu hỏi:

3.1) $1.3E+20$ có biểu diễn nhị phân ra sao.

```
int main()
{
    //1. Bieu dien nhi phan cua so 1.3E+20
    cout << "1.3E+20: ";
    float number1 = 1.3E+20;
    dumpFloat(&number1);
}
```

1.3E+20: 0 11000001 11000011000001110011001

3.2) Số float nhỏ nhất lớn hơn 0 là số nào? Biểu diễn nhị phân của nó?

```
int main()
{
    //2. So float nho nhat > 0 la so nao, bieu dien nhi phan
    cout << "So float nho nhat > 0: ";
    float temp = numeric_limits<float>::min();
    printFloat(temp);
    cout << "Bieu dien nhi phan cua no: ";
    dumpFloat(&temp);
}
```

So float nho nhat > 0: 1.17549e-38
 Bieu dien nhi phan cua no: 0 00000001 000000000000000000000000

3.3) Những trường hợp nào tạo ra các số đặc biệt (kiểu float) (viết chương trình thử nghiệm và giải thích kết quả):

- **Số vô cùng (inf):** Kết quả in ra sẽ là số (+ vô cùng) hoặc (- vô cùng) tùy thuộc vào giá trị truyền vào là âm hay dương.

```
//So vo cung
float inf = numeric_limits<float>::infinity();
cout << "So vo cung: ";
printFloat(inf);
```

So vo cung: + vo cung

- **Số báo lỗi NaN (Not a Number):** Là một giá trị đặc biệt trong dạng số thực không thể chứa 1 số nào cả. Do đó kết quả sẽ in ra là (NaN).

```
//So bao loi NaN
float nan = numeric_limits<float>::quiet_NaN();
cout << "So bao loi NaN: ";
printFloat(nan);
```

So bao loi NaN: NaN

- **Các VD:**

- *X - (+ vô cùng): Khi trừ 1 số cho 1 số vô cùng dương kết quả sẽ là (- vô cùng).*

```
//So vo cung
float inf = numeric_limits<float>::infinity();
cout << "So vo cung: ";
printFloat(inf);
```

```
//Mot so phép toan tao ra so dac biet
float x = 0.0;
float y = 1.0;
float z = -1.0;
float result;
```

```
// X - (+∞)
result = x - inf;
cout << "X - (+ vo cung): ";
printFloat(result);
```

```
So vo cung: + vo cung
X - (+ vo cung): - vo cung
```


- $(+\infty) - (+\infty)$: Khi trừ 2 số vô cùng dương kết quả sẽ là (NaN) vì phép toán này không xác định. Trong số thực, việc cộng, trừ, nhân, chia 2 số vô cùng nó không có ý nghĩa vì ta không thể biết giới hạn của số đó nên không thể thực hiện được phép toán giữa chúng.

```
//So vo cung
float inf = numeric_limits<float>::infinity();
cout << "So vo cung: ";
printfFloat(inf);

//Mot so phép toán tạo ra số đặc biệt
float x = 0.0;
float y = 1.0;
float z = -1.0;
float result;

//  $(+\infty) - (+\infty)$ 
result = inf - inf;
cout << "(+ vo cung) - (+ vo cung): ";
printfFloat(result);
```

```
So vo cung: + vo cung
(+ vo cung) - (+ vo cung): NaN
```

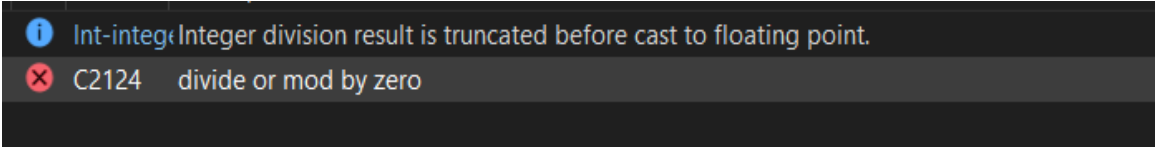
- $X/0$: Là một phép toán không xác định kết quả sẽ in ra là (NaN).

```
//Mot so phép toán tạo ra số đặc biệt
float x = 0.0;
float y = 1.0;
float z = -1.0;
float result;

//  $X/0$ 
result = x / 0;
cout << "X/0: ";
printfFloat(result);
```

```
X/0: NaN
```

- $0/0$: Là phép toán không được định nghĩa trong ngôn ngữ C++ làm cho trình biên dịch báo lỗi



The screenshot shows two compiler messages. The first is an information message: 'Int-integer Integer division result is truncated before cast to floating point.' The second is an error message: 'C2124 divide or mod by zero'.

- $Sqrt(X)$ với $X < 0$: Khi ta thực hiện khai căn bậc 2 của 1 số âm, Kết quả sẽ là (NaN) vì đây là 1 phép toán không xác định trong tập hợp số thực.

```
//Mot so phép toan tao ra so dac biet
float x = 0.0;
float y = 1.0;
float z = -1.0;
float result;

//sqrt(X) với X < 0
float negative = -1.0;
result = sqrt(negative);
cout << "sqrt(X) voi X < 0: ";
printfFloat(result);
```

```
sqrt(X) voi X < 0: NaN
```

Bài 4: Khảo sát các trường hợp sau đây (viết chương trình thử nghiệm và giải thích kết quả):

```
//Khai bao cac gia tri
float f = 3.14159;
int i = 10;
float x = 1.0, y = 2.0, z = 3.0;
```

4.1) Chuyển đổi float -> int -> float. Kết quả như ban đầu ?

```
// 1. Chuyen doi float -> int -> float
int temp = static_cast<int>(f);
float result1 = static_cast<float>(temp);
cout << "1. Chuyen doi float -> int -> float: ";
if (result1 == f) {
    cout << "true" << endl;
}
else {
    cout << "false" << endl;
}
```

1. Chuyen doi float -> int -> float: false

Giải thích: Khi chuyển đổi float->int->float thì giá trị mới không giống giá trị ban đầu, điều này có thể là do mất mát dữ liệu trong quá trình chuyển từ float -> int

4.2) Chuyển đổi int -> float -> int. Kết quả như ban đầu ?

```
// 2. Chuyen doi int -> float -> int. ket qua nhu nao?
float temp2 = static_cast<float>(i);
int result2 = static_cast<int>(temp2);
cout << "2. Chuyen doi int -> float -> int: ";
if (result2 == i) {
    cout << "true" << endl;
}
else {
    cout << "false" << endl;
}
```

2. Chuyển đổi `int -> float -> int`: `true`

Giải thích: Khi chuyển đổi `int -> float -> int` thì giá trị mới giống giá trị ban đầu, vì giá trị `int` có thể được biểu diễn chính xác bằng `float` nên dữ liệu không bị mất mát trong quá trình chuyển đổi

4.3) Phép cộng số chấm động có tính kết hợp ? $(x+y)+z = x+(y+z)$ Với i là biến kiểu `int`, f là biến kiểu `float`.

```
// 3. Phép cộng số chấm động có tính kết hợp?
float result3a = (x + y) + z;
float result3b = x + (y + z);
cout << "3. Phép cộng số chấm động có tính kết hợp: ";
if (result3a == result3b) {
    cout << "true" << endl;
}
else {
    cout << "false" << endl;
}
```

3. Phép cộng số chấm động có tính kết hợp: `true`

Giải thích: Từ kết quả ta có thể kết luận phép cộng số chấm động có tính chất kết hợp và kết quả lúc sau = kết quả lúc đầu.

4.4) $i = (\text{int}) (3.14159 * f);$

```
// 4. i = (int) (3.14159 * f);
i = static_cast<int>(3.14159f * f);
cout << "4. i = (int) (3.14159 * f): " << i << endl;
```

4. $i = (\text{int}) (3.14159 * f): 9$

*Giải thích: giá trị của biến (i) sẽ được gán bằng phần nguyên của $(3.14159 * f)$ Nên kết quả là số nguyên 9.*

4.5) $f = f + (\text{float}) i$;

```
// 5. f = f + (float) i;  
f = f + static_cast<float>(i);  
cout << "5. f = f + (float) i: " << f << endl;
```

5. f = f + (float) i: 13.1416

Giải thích: giá trị của biến (f) sẽ được cộng với giá trị của biến (i) (đã chuyển đổi thành float). Kết quả sẽ là $13.1416 = 3.1416 + 10$

4.6) $\text{if } (i == (\text{int})(\text{float}) i) \{ \text{printf}(\text{"true"}); \}$

```
// 6. if (i == (int)(float) i)  
cout << "6. if (i == (int)(float) i): ";  
if (i == static_cast<int>(static_cast<float>(i))) {  
    cout << "true" << endl;  
}  
else {  
    cout << "false" << endl;  
}
```

6. if (i == (int)(float) i): true

Giải thích: giá trị của biến (i) được chuyển đổi thành float và chuyển đổi trở lại thành int, vẫn giữ nguyên giá trị. Vì int không thể bị mất mát dữ liệu khi chuyển đổi qua float và trở lại

4.7) $\text{if } (i == (\text{int})(\text{double}) i) \{ \text{printf}(\text{"true"}); \}$

```
// 7. if (i == (int)(double) i)  
cout << "7. if (i == (int)(double) i): ";  
if (i == static_cast<int>(static_cast<double>(i))) {  
    cout << "true" << endl;  
}  
else {  
    cout << "false" << endl;  
}
```

7. if (i == (int)(double) i): true

Giải thích: giá trị của biến (i) được chuyển đổi thành double và chuyển đổi trở lại thành int, vẫn giữ nguyên giá trị. Vì int không thể bị mất mát dữ liệu khi chuyển đổi qua double và trở lại

4.8) `if (f == (float)((int) f)) { printf("true"); }`

```
// 8. if (f == (float)((int) f))
cout << "8. if (f == (float)((int) f)): ";
if (f == static_cast<float>(static_cast<int>(f))) {
    cout << "true" << endl;
}
else {
    cout << "false" << endl;
}
```

8. if (f == (float)((int) f)): false

Giải thích: giá trị của biến (f) sau khi được chuyển đổi thành int và sau đó chuyển đổi trở lại float không giống giá trị ban đầu. Vì có sự mất mát dữ liệu trong quá trình chuyển đổi từ float sang int

4.9) `if (f == (double)((int) f)) { printf("true"); }`

```
// 9. if (f == (double)((int) f))
cout << "9. if (f == (double)((int) f)): ";
if (f == static_cast<double>(static_cast<int>(f))) {
    cout << "true" << endl;
}
else {
    cout << "false" << endl;
}
```

9. if (f == (double)((int) f)): false

Giải thích: giá trị của biến (f) sau khi được chuyển đổi thành int và sau đó chuyển đổi trở lại double không giống giá trị ban đầu. Vì có sự mất mát dữ liệu trong quá trình chuyển đổi từ float sang int và sau đó từ int sang double.