**ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH**

**TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN**

**KHOA CÔNG NGHỆ THÔNG TIN**



**LÊ PHẠM HOÀNG TRUNG**

**BÁO CÁO HỆ THỐNG MÁY TÍNH**

**KHẢO SÁT SỐ CHẤM ĐỘNG**

**Chuyên ngành: Trí tuệ Nhân tạo**

Thành phố Hồ Chí Minh – 2023

**PHẦN I: ĐÁNH GIÁ**

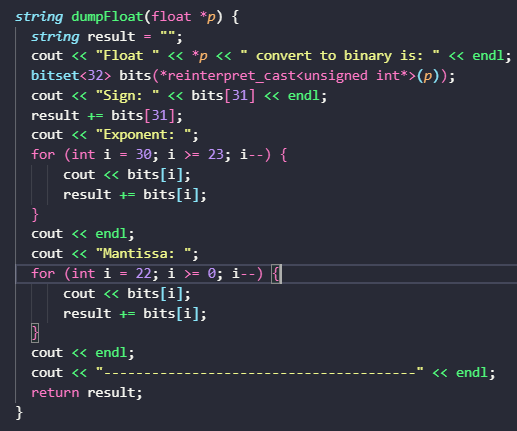
|  |  |  |
| --- | --- | --- |
| **Bài tập** | **Độ hoàn thiện** | **Chú thích** |
| Bài 1: Viết chương trình nhập vào số chấm động. Hãy xuất ra biểu diễn nhị phân từng thành phần (dấu, phần mũ, phần trị) của số chấm động vừa nhập | 100% | *- Code được viết trong file ex1.cpp.*  *- Chương trình đã có sẵn một số test case.* |
| Bài 2: Viết chương trình nhập vào biểu diễn nhị phân của số chấm động. Hãy xuất ra biểu diễn thập phân tương ứng | 100% | *- Code được viết trong file ex2.cpp.* |
| Bài 3: Dùng hai hàm đã viết để khảo sát các câu hỏi | 100% | - Câu 3.1 và câu 3.2 được thử nghiệm trong *file ex1.cpp.*  - Câu 3.3: Áp dụng cho số +inf; -inf và NaN cũng được thử nghiệm trong *file ex1.cpp* |
| Bài 4: Khảo sát các trường hợp | 100% | *- Code được viết trong file ex4.cpp* |

**- Đánh giá độ hoàn thiện tổng hợp cả bài:** 100%. Đã xử lý các trường hợp số thực đặc biệt trong bài 1 và bài 2. Ở bài 3 có nêu thêm các ví dụ về các số thực đặc biệt ngoài bài giảng trên lớp và cách nhận biết.

**PHẦN II: KẾT QUẢ BÀI LÀM**

**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 (*code được viết trong file ex1.cpp*):**

**- Chương trình C++:**

****

**- Test case lần lượt:**

+ Số chấm động (32-bit): 6 xuất 0 10000001 10000000000000000000000 + Số chấm động: -12.625 xuất 1 10000010 10010100000000000000000

+ Số chấm động: 0.1015625 tức 0 01111011 10100000000000000000000 + Số chấm động: 0.1 xuất 0 01111011 10011001100110011001101

+ Số chấm động: 0 xuất 0 00000000 00000000000000000000000

**Text

Description automatically generated**

**- Kết quả chạy chương trình:**

**Text

Description automatically generated**

**Bài 2: Viết chương trình nhập vào biểu diễn nhị phân của số chấm động. Hãy xuất ra biểu diễn thập phân tương ứng (*code được viết trong file ex2.cpp):***

**- Chương trình:**

**Text

Description automatically generated**

**- Test case lần lượt:**

Binary: 0 10001000 01101100001000000000000 -> 728.25

Binary: 1 01000110 01101011000000000000000 -> -9.83913471531 × 10^(-18) Binary: 0 01111011 10011001100110011001101 -> 0.1

Binary: 0 11111111 00000000000000000000000 -> +∞

Binary: 0 11111111 10000000000000000000000 -> NaN

**Text

Description automatically generated**

**- Kết quả chạy chương trình:**

**Text

Description automatically generated**

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

1) **1.3E+20** có biểu diễn nhị phân ra sao?

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

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

a) Số vô cùng (inf)

b) Số báo lỗi NaN

c) Ví dụ: X – (+∞), (+∞) – (+∞), X/0, 0/0, ∞/∞, sqrt(X) với X<0,…(Tham khảo thêm một số trường hợp trong slide 13 bài giảng Số chấm động)

***Trả lời***

**1) Số 1.3E+20 có biểu diễn:**

+ Float 1.3e+20 convert to binary is:

+ Sign: 0

+ Exponent: 11000001

+ Mantissa: 11000011000001110011001

- Kết quả bằng hình ảnh thực tế từ chương trình câu 1 (hàm dumpFloat):

***Text

Description automatically generated***

**2) Số float nhỏ nhất lớn hơn 0 là số:** 2^(-126) (theo như trong slide bài giảng). Biểu diễn nhị phân của nó là:

+ Float 1.17549e-38 convert to binary is:

+ Sign: 0

+ Exponent: 00000001

+ Mantissa: 00000000000000000000000

- Kết quả bằng hình ảnh thực tế từ chương trình:

Text

Description automatically generated

**3)** **Những trường hợp tạo ra các số đặc biệt:**

- Trong chuẩn IEEE 754 cho số dấu phẩy động, có một số giá trị đặc biệt được gọi là "giá trị đặc biệt". Các giá trị đặc biệt này bao gồm số không, vô cùng, NaN, và các giá trị số dấu phẩy động bình thường.

- **Số vô cùng:** Nếu bit exponent (8 bit kế tiếp bit dấu) đều bằng 1 và toàn bộ bit mantissa (23 bit tiếp theo) đều bằng 0, thì đây là giá trị vô cùng (+infinity) hoặc âm vô cùng (-infinity), tùy thuộc vào bit dấu.

- **Số báo lỗi NaN:** Nếu bit exponent đều bằng 1 và bit mantissa không đều bằng 0, thì đây là một giá trị NaN.

- Vì vậy, trong hàm dumpFloat trên, để kiểm tra một số float có phải là vô cùng hay NaN, ta chỉ cần kiểm tra bit exponent và bit mantissa theo các quy tắc trên. Nếu bit exponent đều bằng 1 và toàn bộ bit mantissa đều bằng 0, đó là số vô cùng. Nếu bit exponent đều bằng 1 và bit mantissa không đều bằng 0, đó là số NaN.

**- Test case:**

+ **Số vô cùng dương:** float x = std::numeric\_limits<float>::infinity();

Khi đưa số này vào hàm dumpFloat(&x) ta sẽ nhận được output tương tự như sau:

Text

Description automatically generated

+ **Số vô cùng âm:** float x = -std::numeric\_limits<float>::infinity();

Khi đưa số này vào hàm dumpFloat(&x) ta sẽ nhận được output tương tự như sau:

Text

Description automatically generated

+ **Số NaN:** float x = std::numeric\_limits<float>::quiet\_NaN();

Khi đưa số này vào hàm dumpFloat(&x) ta sẽ nhận được output tương tự như sau:

Text

Description automatically generated

- **Lưu ý** rằng giá trị mantissa của NaN không chỉ toàn bộ là 0 như với giá trị vô cùng. Thay vào đó, nó phải có ít nhất một bit là 1 để phân biệt với giá trị vô cùng. Trong ví dụ này, bit thứ 23 của mantissa là 1, do đó đó là một giá trị NaN hợp lệ.

**\*\* Một số trường hợp tạo số thực đặc biệt khác thường gặp trong lập trình (đây là các trường hợp tham khảo thêm trên mạng và slide bài giảng):**

**- Số vô cùng (infinity):** Nếu thực hiện phép chia mà mẫu (divisor) bằng 0 hoặc khi một số âm được chuyển đổi thành số dương vô cùng, thì kết quả của phép tính sẽ là số vô cùng. Ví dụ:

*float a = 1.0 / 0.0; // a sẽ có giá trị là infinity*

*float b = -3.5;*

*float c = b / 0; // c sẽ có giá trị là infinity*

**- Số báo lỗi NaN (not-a-number):** Nếu thực hiện các phép tính không hợp lệ hoặc kết quả không xác định, thì kết quả của phép tính sẽ là NaN. Ví dụ:

*float d = 0.0 / 0.0; // d sẽ có giá trị là NaN*

*float e = sqrt(-1); // e sẽ có giá trị là NaN*

**- Phép tính với số vô cùng hoặc NaN:**

+ X – (+∞) hoặc (+∞) – (+∞): Kết quả sẽ là NaN nếu cả 2 số đều là số vô cùng, hoặc là số vô cùng có dấu phù hợp nếu chỉ có một số là số vô cùng.

+ X/0 hoặc ∞/∞: Kết quả sẽ là NaN nếu số chia hoặc số bị chia là số vô cùng, hoặc là số vô cùng có dấu phù hợp nếu số chia bằng 0 và số bị chia khác 0.

+ 0/0: Kết quả sẽ là NaN.

+ sqrt(X) với X < 0: Kết quả sẽ là NaN nếu X là số âm.

**- Ví dụ:**

*float f = 5.0 / 0.0; // f sẽ có giá trị là infinity*

*float g = f - f; // g sẽ có giá trị là NaN*

*float h = sqrt(-2.0); // h sẽ có giá trị là NaN*

*float i = 0.0 / 0.0; // i sẽ có giá trị là NaN*

*float j = 1.0 / 0.0 + f; // j sẽ có giá trị là infinity nếu f là số dương,*

*// và NaN nếu f là số âm hoặc là NaN*

**- Tràn số (overflow):** Nếu giá trị của một số vượt quá giới hạn tối đa mà kiểu dữ liệu float có thể biểu diễn, kết quả sẽ là số vô cùng. Ví dụ:

*float k = 3.4e38 \* 2; // k sẽ có giá trị là infinity vì tràn số*

*float l = -3.4e38 \* 2; // l sẽ có giá trị là -infinity vì tràn số*

**- Số nhỏ dưới ngưỡng (underflow):** Nếu giá trị của một số nhỏ hơn ngưỡng tối thiểu mà kiểu dữ liệu float có thể biểu diễn, kết quả sẽ là số 0. Ví dụ:

*float m = 1.0e-40; // m sẽ có giá trị là 0 vì số quá nhỏ dưới ngưỡng*

*float n = 2.0e-45; // n sẽ có giá trị là 0 vì số quá nhỏ dưới ngưỡng*

Các trường hợp này thường xảy ra khi tính toán số dấu phẩy động trong các thuật toán phức tạp hoặc khi giải quyết các bài toán về khoa học, kỹ thuật và toán học. Việc hiểu rõ cách các số đặc biệt này được tạo ra và cách xử lý chúng là rất quan trọng để đảm bảo tính toàn vẹn và chính xác của các phép tính số học.

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

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

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

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**

4. i = (int) (3.14159 \* f);

5. f = f + (float) i;

6. if (i == (int)((float) i)) { printf(“true”); }

7. if (i == (int)((double) i)) { printf(“true”); }

8. if (f == (float)((int) f)) { printf(“true”); }

9. if (f == (double)((int) f)) { printf(“true”); }

***Giải***

**1) Chuyển đổi từ float -> int -> float (Chương trình thử nghiệm nằm trong file ex4.cpp):**

**- Chương trình:**

Text

Description automatically generated

**- Kết quả chạy:**

**A screenshot of a computer

Description automatically generated with medium confidence**

**- Giải thích:** Kết quả có thể khác so với ban đầu. Trong trường hợp này, ta muốn chuyển đổi một số kiểu float sang kiểu int và sau đó chuyển ngược trở lại kiểu float. Việc chuyển đổi từ float sang int sẽ làm mất thông tin phần thập phân của số ban đầu. Sau đó, khi ta chuyển đổi lại từ kiểu int sang kiểu float, số đó sẽ được làm tròn lại với phần thập phân bị mất. Do đó, kết quả có thể khác so với số ban đầu.

**2) Chuyển đổi từ int -> float -> int (Chương trình thử nghiệm nằm trong file ex4.cpp):**

**- Chương trình:**

**Text

Description automatically generated**

**- Kết quả chạy:**

****

**- Giải thích:** Kết quả giống nhau so với ban đầu. Trong trường hợp này, ta muốn chuyển đổi một số kiểu int sang kiểu float và sau đó chuyển ngược trở lại kiểu int. Việc chuyển đổi từ kiểu int sang kiểu float không làm mất thông tin nào của số ban đầu, vì kiểu float có độ chính xác cao hơn so với kiểu int. Sau đó, khi ta chuyển đổi lại từ kiểu float sang kiểu int, số đó sẽ được làm tròn lại về kiểu int, nhưng sẽ giữ nguyên giá trị chính xác của số ban đầu.

**3) Phép cộng số chấm động có tính kết hợp ? (x+y)+z = x+(y+z).**

**- Chương trình thử nghiệm:**

**Text

Description automatically generated**

**- Kết quả chạy:**

**Text

Description automatically generated**

**- Giải thích:** Phép cộng số chấm động có tính kết hợp. Tức là, cho bất kỳ 3 số chấm động x, y, và z, ta luôn có (x+y)+z = x+(y+z). Điều này có thể được chứng minh bằng cách sử dụng tính chất kết hợp của phép cộng trong đại số.

**- Trong các câu dưới (4, 5, 6, 7, 8, 9) ta sẽ sử dụng 1 đoạn code với giá trị i và f đã được khởi tạo:**

**A screenshot of a computer

Description automatically generated with medium confidence**

**- Kết quả trả về:**

**Text

Description automatically generated**

**4. i = (int) (3.14159 \* f);**

- Giải thích: Câu này, ta sẽ nhân 3.14159 với một số thực (float), tức: 3.14159 \* 3.14 = 9.8645926, sau đó ta ép kiểu dữ liệu thành int ta sẽ nhận được kết quả cuối là 9.

**5. f = f + (float) i;**

- Giải thích: Câu này ta sẽ ép i sang kiểu float (tức i = 9.0). Sau đó ta lấy f = 3.14 + i = 9.0 ra kết quả là 12.14.

**6. if (i == (int)((float) i)) { printf(“true”); }**

- Giải thích: Câu này i đang ở kiểu int tức i = 9. Ta ép i sang float = 9.0 sau đó ta lại ép i về int là = 9. So sánh với i ban đầu = 9 nên ta có “Statement 6 is true”.

**7. if (i == (int)((double) i)) { printf(“true”); }**

- Giải thích: Câu này ta sẽ ép i = 9 đang ở dạng int về kiểu double, tức i = 9.0. Sau đó ta lại ép nó về kiểu int tức i trở về = 9. So sánh với i ban đầu = 9 nên ta có “Statement 7 is true”.

**8. if (f == (float)((int) f)) { printf(“true”); }**

- Giải thích: Câu này ta sẽ ép f = 12.14 đang ở dạng float về kiểu int, tức f = 12. Sau đó ta lại ép nó về kiểu float tức f = 12.0. So sánh với f ban đầu = 12.14 sẽ trả về False, không xuất gì ra màn hình.

**9. if (f == (double)((int) f)) { printf(“true”); }**

- Giải thích: Câu này ta sẽ ép f = 12.14 đang ở dạng float về kiểu int, tức f = 12. Sau đó ta lại ép nó về kiểu double tức f = 12.0. So sánh với f ban đầu = 12.14 sẽ trả về False, không xuất gì ra màn hình.