TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA CÔNG NGHỆ THÔNG TIN

ĐỒ ÁN BIỂU DIỄN VÀ TÍNH TOÁN SỐ HỌC TRÊN MÁY TÍNH

NĂM HỌC: 2018 - 2019

1/ Đánh giá mức độ hoàn thành:

Số nguyên lớn	Hoàn thành
a. Hàm Nhập: void ScanQInt(QInt &x)	100%
b. Hàm xuất: void PrintQInt(QInt x)	100%
c. Hàm chuyển đổi số QInt thập phân sang nhị phân: bool * DecToBin (QInt x)	100%
d. Hàm chuyển đổi số QInt nhị phân sang thập phân: QInt BinToDec(bool *bit)	100%
e. Hàm chuyển đổi số QInt nhị phân sang thập lục phân: char *BinToHex(bool *bit)	100%
f. Hàm chuyển đổi số QInt thập phân sang thập lục phân: char *DecToHex(QInt x)	100%
g. Các operator toán tử : “+”, “-” , “*”, “/”	100%
h. Các toán tử so sánh và gán: “<”, “>”, “==”, “<=”, “>=”, “=”	100%
i. Các toán tử: AND “&”, OR “|”, XOR “^”, NOT “~”	100%
j. Các toán tử: dịch trái “<<”, dịch phải “>>”, xoay trái: “rol”, xoay phải: “ror”	100%

Số chấm động chính xác cao	Hoàn thành
a. Hàm Nhập: void ScanQfloat (Qfloat &x)	100%
b. Hàm xuất: void PrintQfloat(Qfloat x)	100%
c. Hàm chuyển đổi số Qfloat nhị phân sang thập phân: Qfloat BinToDec(bool *bit)	100%
d. Hàm chuyển đổi số Qfloat thập phân sang nhị phân bool *DecToBin(Qfloat x)	100%

***Tổng kết:**

Mức độ hoàn thành đồ án: 100%

2/ Đánh giá công việc:

MSSV	Họ tên	Công việc	Hoàn thành
1712345	Mai Thiên Định	Chạy Test chương trình	**10%**
1712187	Vương Bảo Trí	Chạy Test chương trình	**10%**
1712299	Nguyễn Hữu Chí	Các phép toán trong Qint	**10%**
1712159	Nguyễn Đỗ Chí Thảo	Biểu diễn số chấm động 128 bits, nghiên cứu thuật toán, viết báo cáo, tạo testcase, test chương trình.	**100%**
1712203	Trần Tử Văn	Biểu diễn số nguyên lớn 128 bits, nghiên cứu thuật toán, triển khai-quản lý code, xử lý file input-output.	**100%**

3/ Ý tưởng thực hiện:

-Để thiết kế kiểu dữ liệu có độ lớn 128 bits (16 bytes) ta sẽ cần phải thao tác trên từng bit của một mảng kiểu dữ liệu unsigned int, có 4 phần tử: `unsigned int a[4];`

-Thiết kế theo hướng đối tượng

4/ Chương trình minh họa:

Có hai giao diện:

1/ **Giao diện người dùng (nhập input từ bàn phím):**

Sử dụng giao diện console hoặc winform (MFC hoặc các thư viện hỗ trợ giao diện trên C/C++). Chương trình hỗ trợ 2 chế độ Qint và Qfloat.

2/ Thực thi theo tham số dòng lệnh:

-Dữ liệu đầu vào của Số nguyên lớn được lưu ở file input1.txt, kết quả xuất ra được lưu ở file output1.txt

- Dữ liệu đầu vào của Số chấm động độ chính xác cao được lưu ở file input2.txt, kết quả xuất ra được lưu ở file output2.txt

Lưu ý: Kiểu dữ liệu cho hệ nhị phân trong số chám động chính xác cao sẽ có dạng dãy số chấm động 128 bits

5/ Phạm vi biểu diễn:

1/ Số nguyên lớn: $[-10^{34},10^{34}]$

2/ Số chấm động chính xác cao:

$$[-(1+\sum_{K=-1}^{-127}2^k)*2^{16383},(1+\sum_{K=-1}^{-127}2^k)*2^{16383}\ ]\ ]$$

6/ Chức năng hàm

1. **Các hàm hỗ trợ - class Utils:**

1.1 Hàm kiểm tra cú pháp chuỗi nhập QInt

```
bool Utils::CheckStringForIntType(string value)
```

- Hàm sẽ duyệt chuỗi value và kiểm lỗi. Nếu có 1 ký tự khác ký tự số sẽ trả về false

Demo:

```
int main()
{
    string str;
    cout << "Nhap QInt:" << endl;
    cin >> str;
    cout<<Utils::CheckStringForIntType(str)<<endl;
    _getch();
}
```

Kết quả trả về:

```
Nhap QInt:
1234
1
```

1.2 Hàm kiểm tra cú pháp chuỗi nhập Qfloat

```
bool Utils::CheckStringForFloatType(string value)
```

- Hàm sẽ duyệt chuỗi value. Chuỗi nhập gồm các ký tự số, có thế có 1 dấu '.' hoặc không có, đầu chuỗi có thể có dấu âm '-' hoặc không. Sai quy tắc này trả về false

Demo:

```
int main()
{
    string str;
    cout << "Nhap QFloat:" << endl;
    cin >> str;
    cout<<Utils::CheckStringForFloatType(str)<<endl;
    _getch();
}
```

Kết quả trả về:

```
Nhap QFloat:
12.34
1
```

1.3 Hàm chuỗi nhân 2

```
string Utils::StringNhan2(string value)
```

- Hàm sẽ trả về chuỗi với giá trị gấp 2 lần chuỗi value, tính toán dựa trên mã ASCII của các ký tự

Demo:

```
int main()
{
    string str;
    cout << "Nhap QInt:" << endl;
    cin >> str;
    cout << Utils::StringNhan2(str)<<endl;
    _getch();
}
```

Kết quả trả về:

```
Nhap QInt:
1212
2424
```

1.4 Chuỗi chia 2

`string Utils::StringChia2(string value)`

- Tương tự hàm nhân, hàm chia sẽ trả về chuỗi có giá trị chia 2 so với chuỗi value, tính toán dựa trên mã ASCII của các ký tự

Demo:

```
int main()
{
    string str;
    cout << "Nhap QInt:" << endl;
    cin >> str;
    cout << Utils::StringChia2(str)<<endl<<endl;
    _getch();
}
```

Kết quả trả về:

```
Nhap QInt:
2468
1234
```

1.5 Hàm cộng chuỗi

```
string Utils::CongStringTheoInt(string a, string b)
```

- Hàm sẽ trả về chuỗi result là kết quả của phép cộng chuỗi a,b theo nguyên tắc cộng số nguyên. Tính toán dựa vào mã ASCII của các ký tự

Demo:

```
int main()
{
    string str1;
    string str2;
    cout << "Nhap hai chuoi:" << endl;
    cin >> str1;
    cin >> str2;
    cout << Utils::CongStringTheoInt(str1, str2);
    _getch();
}
```

Kết quả trả về:

```
Nhap hai chuoi:
1234
9876
11110
```

1.6 Hàm tìm mũ của 2 dưới dạng chuỗi

```
string Utils::MuCua2(unsigned int x)
```

- Hàm tìm mũ 2 với tham số mũ là số dương, hàm tính toán dựa trên hàm cộng chuỗi.

  VD : 2^3 sẽ là 2 + 2 + 2

Demo:

```
int main()
{
    int n;
    cin >> n;
    cout << Utils::MuCua2(n)<<endl;
    _getch();
}
```

Kết quả trả về:

```
4
16
```

1.7 Hàm Count Space

```
int Utils::CountSpace(string value)
```

- Hàm sẽ trả về số khoảng trắng trong 1 chuỗi. Hàm này sẽ hỗ trợ cho việc xử lý file input

Demo:

```
int main()
{
    string str;
    getline(cin, str);
    cout << Utils::CountSpace(str)<<endl;
    _getch();
}
```

Kết quả trả về:

```
e d d u fff
4
```

2. Các hàm và thuật toán trong class Qint:

2.1 Toán tự nhập (>>) – Scanf

```
istream& operator>>(istream& inDev, QInt &x)
```

- Thuật toán chính của hàm này chính là xử lý chuỗi. Người dùng sẽ nhập vào 1 chuỗi, hàm sẽ kiểm tra đúng sai về mặt kiểu dữ liệu (đúng sẽ tiếp tục, sai mời nhập lại). Sau đó sẽ đến phần xử lý chuỗi và lưu vào Qint. Như trong nội dung đã học ở môn KTMT&HN, để chuyển 1 số thập phân thành nhị phân, ta cần chia 2 số và lấy phần dư đến khi nào số bằng 0. Và ở đây cũng vậy, em đã thực hiện phép chia chuỗi nhờ vào hàm trợ giúp StringChia2, sau đó lưu kết quả tạm vào 1 mảng kiểu bool. Sau đó em bật các bit tương ứng ở các vị trí vào Qint

2.2 Toán tử xuất(<<) – Printf

```
ostream& operator<<(ostream &outDev, QInt x)
```

- Thuật toán chính của hàm này tiếp tục là xử lý chuỗi. Tận dụng 2 hàm CongString & MuCua2 thực hiện trên chuỗi đã viết ở class Utils. Em đã thực hiện tuần tự và cho ra một kết quả số nguyên ở dạng chuỗi.

Demo:

```
int main()
{
    QInt x;
    cout << "Nhap QInt" << endl;
    cin >> x;
    cout << x<<endl;
    _getch();
}
```

Kết quả trả về:

```
Nhap QInt
1234
1234
```

2.3 Các hàm chuyển đổi cơ số

2.3.1 Chuyển từ hệ 10 sang hệ 2:

- Lấy số cần chuyển đổi chia cho 2 (kết quả chỉ lấy phần nguyên), sau đó tiếp tục lấy kết quả chia 2 (và cũng chỉ lấy phần nguyên), kết quả số nhị phân thu được là tập hợp các số dư của các phép chia.

```
int main()
{
    QInt x;
    cout << "Nhap QInt : "; cin >> x;
    cout << "Binary    : "; Output2(QInt::DecToBin(x));
    _getch();
}
```

```
Nhap QInt : 123456789
Binary    : 11101011011110011010010101
```

2.3.2 Chuyển từ hệ 2 sang hệ 10:

-Tính tổng các tích của kí tự nhị phân nhân 2 lũy thừa vị trí.

```
int main()
{
    QInt x;
    cout << "Nhap QInt : "; cin >> x;
    bool *bit = QInt::DecToBin(x);
    cout << "Binary    : "; Output2(bit);
    QInt y = QInt::BinToDec(bit);
    cout << "QInt Y    : " << y << endl;
    _getch();
}
```

```
Nhap QInt : 12345678912
Binary    : 10110111111101111000001110001000000
QInt Y    : 12345678912
```

2.3.3 Chuyển từ hệ 2 sang hệ 16:

Ta gom 4 chữ số của số cần chuyển theo thứ tự lần lượt từ phải sang trái, sau đó chuyển đổi theo bảng sau:

Nhị phân (BIN)	Thập lục phân (HEX)
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

```
int main()
{
    QInt x;
    cout << "Nhap QInt : "; cin >> x;
    bool *bit = QInt::DecToBin(x);
    cout << "Binary    : "; Output2(bit);
    QInt y = QInt::BinToDec(bit);
    char *hex = QInt::BinToHex(bit);
    cout << "Hexa      : "; Output16(hex);
    _getch();
}
```

```
Nhap QInt : 12345678910
Binary    : 1011011111110111000001110000111110
Hexa      : 2DFDC1C3E
```

2.3.4 Chuyển từ hệ 10 sang hệ 16:

- Cần chuyển số từ hệ 10 sang hệ 2, sau đó tiếp tục chuyển từ hệ 2 sang hệ 16

```
int main()
{
    QInt x;
    cout << "Nhap QInt : "; cin >> x;
    char *hex = QInt::DecToHex(x);
    cout << "Hexa      : "; Output16(hex);
    _getch();
}
```

```
Nhap QInt : 12345678910
Hexa      : 2DFDC1C3E
```

2.4 Các hàm so sánh (==, !=, >=, <=, >, <)

- Chúng ta chỉ tập xét 2 phép chính là: == và >. Vì các phép còn lại có thể suy ra từ 2 phép này
- Ở phép == : thuật toán vô cùng đơn giản, em chỉ việc duyệt hết các bit của 2 tham số, nếu tất cả đều bằng nhau thì tương tự 2 Qint tham số cũng sẽ bằng nhau

Demo:

```
int main()
{
    QInt x,y;
    cout << "Nhap QInt" << endl;
    cin >> x;
    cin >> y;
    if (x == y) cout << 1<<endl;
    else cout << 0<<endl;
    _getch();
}
```

Kết quả trả về:

```
Nhap QInt
1234
5670
0
```

- Ở phép > : Cụ thể hơn là a > b. Ở đây em sẽ xét từ bit trái nhất của 2 tham số qua phải. Cũng như phép so sánh ở hệ 10. Ta sẽ so sánh 2 số ở cùng 1 cột, tuần tự từ trái qua phải. Nếu có 1 vị trí mà a.bit[i] > b.bit[i] trước thì a > b và ngược lại

Demo:

```
int main()
{
    QInt x,y;
    cout << "Nhap QInt" << endl;
    cin >> x;
    cin >> y;
    if (x > y) cout << 1<<endl;
    else cout << 0<<endl;
    _getch();
}
```

Kết quả trả về:

```
Nhap QInt
1234
5670
0
```

2.5 Các hàm phép toán logic (&, ^, |, ~, <<, >>, rol, ror, =)

Đầu tiên cần phải chuyển đổi cơ số thành 1 mảng 128 bit và đã trực tiếp thực thi thuật toán trên đó. Sau đó em lại chuyển mảng nhị phân về Qint trả về kết quả cuối cùng

2.5.1 Toán tử &

- Lấy 2 toán hạng nhị phân có chiều dài bằng nhau và thực hiện phép toán lý luận & trên mỗi cặp bit tương ứng bằng cách nhân chúng lại với nhau. Nhờ đó, nếu cả hai bit ở vị trí được so sánh đều là 1, thì bit hiển thị ở dạng nhị phân sẽ là 1 (1 & 1 = 1); ngược lại thì kết quả sẽ là 0 (1 & 0 = 0)

Demo:

```
int main()
{
    QInt x,y,z;
    cout << "Nhap QInt" << endl;
    cin >> x;
    cin >> y;
    z = x&y;
    cout << z << endl;
    _getch();
}
```

Kết quả trả về:

```
Nhap QInt
12
13
12
```

2.5.2 Toán tử ^

- Lấy hai dãy bit có cùng độ dài và thực hiện phép toán logic bao hàm ^ trên mỗi cặp bit tương ứng. Kết quả ở mỗi vị trí là 1 chỉ khi bit đầu tiên là 1 hoặc nếu chỉ khi bit thứ hai là 1, nhưng sẽ là 0 nếu cả hai là 0 hoặc cả hai là 1. Ở đây ta thực hiện phép so sánh hai bit, kết quả là 1 nếu hai bit khác nhau và là 0 nếu hai bit giống nhau.

Demo:

```
int main()
{
    QInt x,y,z;
    cout << "Nhap QInt" << endl;
    cin >> x;
    cin >> y;
    z = x^y;
    cout << z << endl;
    _getch();
}
```

Kết quả trả về:

```
Nhap QInt
12
13
1
```

2.5.3 Toán tử |

- Lấy hai dãy bit có độ dài bằng nhau và thực hiện phép toán lý luận bao hàm | trên mỗi cặp bit tương ứng. Kết quả ở mỗi vị trí sẽ là 0 nếu cả hai bit là 0, ngược lại thì kết quả là 1

Demo:

```
int main()
{
    QInt x,y,z;
    cout << "Nhap QInt" << endl;
    cin >> x;
    cin >> y;
    z = x|y;
    cout << z << endl;
    _getch();
}
```

Kết quả trả về:

```
Nhap QInt
12
13
13
```

2.5.4 Toán tử ~

- Thực hiện phủ định luận lý trên từng bit, tạo thành bù 1 của giá trị nhị phân cho trước. Bit nào là 0 thì sẽ trở thành 1, và 1 sẽ trở thành 0

Demo:

```
int main()
{
    QInt x,y,z;
    cout << "Nhap QInt" << endl;
    cin >> x;
    z = ~x;
    cout << z << endl;
    _getch();
}
```

Kết quả trả về:

```
Nhap QInt
12
-13
```

2.5.5 Toán tử >>

- Chuyển tất cả các bit sang phải, bỏ bit phải nhất, thêm 0 ở bit trái nhất

Demo:

```
int main()
{
    QInt x,z;
    cout << "Nhap QInt" << endl;
    cin >> x;
    z = x>>2;
    cout << z << endl;
    _getch();
}
```

Kết quả trả về:

```
Nhap QInt
12
3
```

2.5.6 Toán tử <<

- Chuyển tất cả các bit sang trái, bỏ bit trái nhất, thêm 0 ở bit phải nhất

Demo:

```
int main()
{
    QInt x,z;
    cout << "Nhap QInt" << endl;
    cin >> x;
    z = x<<2;
    cout << z << endl;
    _getch();
}
```

Kết quả trả về:

```
Nhap QInt
12
48
```

2.5.7 Toán tử rol

- Chuyển tất cả các bit sang trái, bit trái nhất thành bit phải nhất

Demo:

```
int main()
{
    QInt x,z;
    cout << "Nhap QInt" << endl;
    cin >> x;
    z = z.RotateLeft(2);
    cout << z << endl;
    _getch();
}
```

Kết quả trả về:

```
Nhap QInt
12
48
```

2.5.8 Toán tử ror

- Chuyển tất cả các bit sang phải, bit phải nhất thành bit trái nhất

Demo:

```
int main()
{
    QInt x,z;
    cout << "Nhap QInt" << endl;
    cin >> x;
    z = x.RotateRight(2);
    cout << z << endl;
    _getch();
}
```

Kết quả trả về:

```
Nhap QInt
12
3
```

2.5.9 Toán tử =

- Sao chép từng bit của của bit cần gán lên bit được gán

Demo: Tất cả các demo ở trên đều sử dụng toán tử gán

2.6 Các hàm tính toán (+, -, *, \)

Thuật toán được thực thi trên mảng các bit. Sau đó lại chuyển mảng bit về kết quả trả về (Qint)

2.6.1 Toán tử +

-Ta lần lượt lấy các bit của từng số hạng(theo chiều từ phải sang trái) rồi cộng lại với nhau theo nguyên tắc: 0 + 0 = 0, 1 + 0 = 1, 0 + 1 = 1

Demo:

```
int main()
{
    QInt x,y,z;
    cout << "Nhap QInt" << endl;
    cin >> x;
    cin >> y;
    z = x+y;
    cout << z << endl;
    _getch();
}
```

Kết quả trả về:

```
Nhap QInt
12
13
25
```

2.6.2 Toán tử -

- Đưa về phép cộng giữa số bị trừ và số bù 2 của số trừ

Demo:

```
int main()
{
    QInt x,y,z;
    cout << "Nhap QInt" << endl;
    cin >> x;
    cin >> y;
    z = x-y;
    cout << z << endl;
    _getch();
}
```

Kết quả trả về:

2.6.3 Toán tử *

-Giả sử ta muốn thực hiện phép nhân M x Q với

+ Q có n bit

- Ta định nghĩa các biến:

+ C (1 bit): đóng vai trò bit nhớ

+ A (n bit): đóng vai trò 1 phần kết quả nhân ([C, A, Q]: kết quả nhân)

+ [C, A] (n + 1 bit) ; [C, A, Q] (2n + 1 bit): coi như các thanh ghi ghép

Thuật toán:

Khởi tạo: A = 0; k = n; Q-1 = 0 (thêm 1 bit = 0 vào cuối Q)

Lặp khi k > 0

{

Nếu 2 bit cuối của Q_0Q_{-1}

{

= 10 thì A – M -> A

= 01 thì A + M -> A

= 00, 11 thì A không thay đổi

}

Shift right [A, Q, Q-1] k = k – 1

}

Kết quả: [A, Q]

Demo:

```
int main()
{
    QInt x,y,z;
    cout << "Nhap QInt" << endl;
    cin >> x;
    cin >> y;
    z = x*y;
    cout << z << endl;
    _getch();
}
```

Kết quả trả về:

```
Nhap QInt
12
13
156
```

2.6.4 Toán tử /

Giả sử ta muốn thực hiện Q / M với M có n bit

Khởi tạo: A = n bit 0 nếu Q > 0; A = n bit 1 nếu Q < 0; k = n

Lặp khi k > 0

{

Shift left (SHL) [A, Q]

A – M -> A

Nếu A < 0: $Q_0 = 0$ và A + M -> A

Ngược lại: $Q_0 = 1$

k = k – 1

}

Kết quả: Q là thương, A là số dư

Demo:

```
int main()
{
    QInt x,y,z;
    cout << "Nhap QInt" << endl;
    cin >> x;
    cin >> y;
    z = x/y;
    cout << z << endl;
    _getch();
}
```

Kết quả trả về:

```
Nhap QInt
49
4
12
```

3. Các hàm và thuật toán class Qfloat

3.1 Toán tự nhập (>>) – Scanf

```
friend istream& operator>>(istream&, QFloat&);
```

- Hàm nhập Qfloat lại tiếp tục xoay quanh xử lý chuỗi. Người dùng sẽ nhập vào 1 chuỗi, chương trình sẽ kiểm tra dữ liệu và phản hồi. Ở đây em tách chuỗi ra làm 2 phần : phần nguyên & phần thập phân. Sau đó em chuyển 2 phần này thành 2 mảng bit bằng các hàm hỗ trợ StringNhan2, StringChia2, MuCua2. Sau đó em áp dụng quy tắc của số chấm động vào và xử lý trên 2 mảng bit. Đúc kết sẽ nhận được 1 mảng 128 bit – chính là số chấm động. Cuối cùng em bật các bit tương ứng ở Qfloat

3.2 Toán tử xuất(<<) – Printf

```
friend ostream& operator<<(ostream&, QFloat);
```

- Hàm xuất Qfloat lại tiếp tục xoay quanh các thuật toán về xử lý chuỗi. Đầu tiên em sẽ copy các bit từ Qfloat ra mảng 128 bit. Sau đó em thực hiện phân tách số chấm động bằng các kiến thức đã học trên lớp nhờ các hàm hỗ trợ StringNhan2, StringChia2, MuCua2. Xong phần này em nhận được 2 chuỗi : phần nguyên & phần thập phân. Cuối cùng em ghép 2 phần lại được 1 chuỗi hoàn chỉnh và cũng là kết quả trả về

3.3 Các hàm chuyển đổi

3.3.1 Chuyển từ hệ 10 sang hệ 2:

Đầu tiên chuyển phần nguyên từ hệ 10 sang hệ 2 như trong QInt. Sau đó chuyển phần thập phân sang hệ 2 theo quy tắc: Nhân phần thập phân với 2, nếu:

+Nếu kết quả lớn hơn 1 thì thêm 1 vào phần thập phân hệ 2, lấy phần thập phân của kết quả rồi tiếp tục thực hiện

+Nếu kết quả bé hơn 2 thì thêm 0 vào phần thập phân hệ 2 rồi tiếp tục thực hiện

+Nếu bằng 1 thì dừng lại

Kết quả trả về sẽ là dãy số chấm động 128 bits

```
int main()
{
    QFloat x;
    cout << "Nhap QFloat : "; cin >> x;
    bool *bit = QFloat::DecToBin(x);
    cout << "Binary      : "; Output2(bit);
    _getch();
}
```

```
Nhap QFloat : -123456.65789
Binary      : 11000000000011111111000100100000010101000011010110111101010100010010111011000110101111001110100001010011001110110001000001110111
```

3.3.2 Chuyển từ hệ 2 sang hệ 10

Số hệ nhị phân cần chuyển đổi sẽ có dạng dãy số chấm động 128 bits

Chuyển phần bên trái dấu chấm(phần nguyên) tương tự như trong Qint, sau đó chuyển phần thập phân theo quy tắc: Nhân lần lượt từng bit với 2 lũy thừa âm vị trí sau dấu chấm: Vd $(0.101)_2 = (1* 2^{-1} + 0* 2^{-2} + 1*2^{-3})_{10}$

Sau đó cộng kết quả phần thập phân và phần nguyên với nhau ta được kết quả.

```
int main()
{
    QFloat x;
    cout << "Nhap QFloat : "; cin >> x;
    bool *bit = QFloat::DecToBin(x);
    cout << "Binary      : "; Output2(bit);
    QFloat y = QFloat::BinToDec(bit);
    cout << "QFloat y    : " << y << endl;
    _getch();
}
```

```
Nhap QFloat : -1234567454.12154654
Binary      : 11000000000111010010011001011000000001000111100001111100011
01101011001000111000111101111100001101000101100011111000000100111101
QFloat y    : -1234567454.1215465400
```

4. Xử lý file input-output – lớp Program

4.1 Xử lý file QInt

```
static void ProgrammingQInt(string filein = "input1.txt",
string fileout = "output1.txt");
```

- Nhìn một cách tổng quát về file input1 của đối tượng Qint, em đã chia ra làm 3 loại chính

  + Chuyển đổi cơ số : 3 tham số - 2 khoảng trắng

  + Tính toán : 4 tham số - 3 khoảng trắng

  + Phép not : 3 tham số với tham số thứ 2 luôn là '~'

  ⇨ Từ đây em đã đúc kết thành 3 hàm con để xử lý riêng từng trường hợp

```
static string ChuyenDoiQInt(string h1, string h2, string
str);

static string NotQInt(string h, string s);

static string TinhToanQInt(string h, string s1, string
toantu, string s2);
```

- Em sẽ đọc từng tuần từng dòng 1 trong file input1.txt. Sau đó em đếm khoảng trắng và tách chuỗi, và đưa tham số vào các hàm tương ứng để xử lý, kết quả trả

về em sẽ push_pack vào 1 vector<string> result. Sau khi đã xử lý xong hết em tuần từ xuất các đối tượng trong vector<string> result ra file output1.txt

4.2 Xử lý file QFLoat

```
static void ProgrammingQFloat(string filein = "input2.txt",
string fileout = "output2.txt");
```

- Khá đơn giản hơn so với Qint vì Qfloat chỉ có chuyển đổi cơ số 10-2,2-10
- Em chỉ viết thêm 1 hàm hỗ trợ

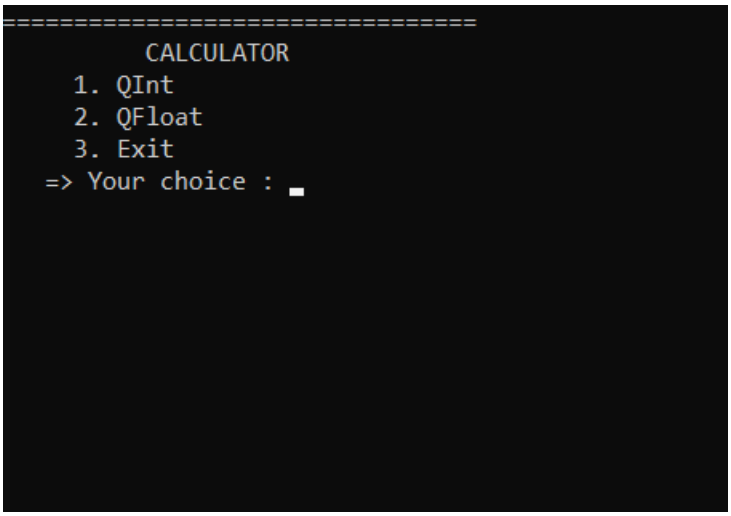
  ```
  static string ChuyenDoiQFloat(string h1, string h2, string
  s);
  ```

- Cũng tương tự như Qint, em cũng đọc từng dòng file input2.txt, và tách chuỗi thành các tham số truyền vào hàm. Sau khi xử lý hàm sẽ trả về kết quả, em lưu vào vector<string> result. Sau khi đã xử lý xong hết em tuần từ xuất các đối tượng trong vector<string> result ra file output2.txt

4.3 Xử lý giao diện người dùng:

```
static void ProgramConsole()
```

- Xuất ra màn menu hướng dẫn, hỗ trợ người dùnng thực hiện thao tác dễ dàng



Tài liệu tham khảo:

Giáo trình môn Kiến trúc máy tính và hợp ngữ, trường Đại Học Khoa Học Tự Nhiên