Họ và tên: Trương Trung Đức

MSSV: 21880291

|  |  |
| --- | --- |
| **Báo cáo** | Bài tập 1 – Chuyên đề tổ chức dữ liệu |
| **Ngày lập báo cáo** | 15/08/2024 |
| **Mức độ hoàn thành** | 100% - hoàn thành các yêu cầu |
| **Mục lục** | [1. Chức năng của module:](#_Toc174700473)  [2. Giao diện cho module (tệp header .h):](#_Toc174700474)  [3. Triển khai module (tệp mã nguồn .c):](#_Toc174700475) |

Nội dung:

1. **Chức năng của module:**

Module định nghĩa kiểu dữ liệu BigInt giúp thao tác với “số nguyên lớn” là số nguyên có nhiều chữ số (không quá 100 chữ số). Kiểu dữ liệu này hỗ trợ các thao tác sau trên số nguyên:

- Nhập xuất và chuyển đổi số nguyên với chuỗi theo cơ số 10 và cơ số 16.

- Chuyển đổi với kiểu số nguyên int của C.

- Các phép toán số học thông thường: cộng, trừ, nhân, chia nguyên và chia lấy dư, đối, trị tuyệt đối, lũy thừa với số mũ nguyên không âm.

- Các phép toán so sánh: bằng, khác, nhỏ hơn, lớn hơn, nhỏ hơn hoặc bằng, lớn hơn hoặc bằng.

- Phép toán lấy phần nguyên của căn bậc 2, bậc 3.

1. **Giao diện cho module (tệp header .h):**

|  |  |
| --- | --- |
| Mô tả | Mã nguồn |
| Tệp header BigInt.h:  chứa khai báo của các hàm, biến, và kiểu dữ liệu | #ifndef BIGINT\_H  #define BIGINT\_H  #define MAX\_DIGITS 100  typedef struct {  char digits[MAX\_DIGITS];  int size;  int is\_negative; // Dấu hiệu số âm  } BigInt;  // Khởi tạo một BigInt từ chuỗi ký tự với cơ số  BigInt initBigInt(const char \*str, int base);  // Chuyển đổi BigInt thành chuỗi theo cơ số  void toString(BigInt num, char \*str, int base);  // Chuyển đổi BigInt từ và sang kiểu int của C  BigInt fromInt(int value);  int toInt(BigInt num);  // In ra một BigInt  void printBigInt(BigInt num, int base);  // Phép cộng, trừ, nhân, chia, chia lấy dư, đối, trị tuyệt đối, lũy thừa  BigInt addBigInt(BigInt a, BigInt b);  BigInt subtractBigInt(BigInt a, BigInt b);  BigInt multiplyBigInt(BigInt a, BigInt b);  BigInt divideBigInt(BigInt a, BigInt b);  BigInt modBigInt(BigInt a, BigInt b);  BigInt negateBigInt(BigInt a);  BigInt absBigInt(BigInt a);  BigInt powerBigInt(BigInt base, unsigned int exponent);  // Các phép so sánh  int compareBigInt(BigInt a, BigInt b);  int equalsBigInt(BigInt a, BigInt b);  int notEqualsBigInt(BigInt a, BigInt b);  int lessThanBigInt(BigInt a, BigInt b);  int greaterThanBigInt(BigInt a, BigInt b);  int lessOrEqualsBigInt(BigInt a, BigInt b);  int greaterOrEqualsBigInt(BigInt a, BigInt b);  // Phép toán lấy phần nguyên của căn bậc 2, bậc 3  BigInt sqrtBigInt(BigInt a);  BigInt cbrtBigInt(BigInt a);  #endif |

1. **Triển khai module (tệp mã nguồn .c):**

|  |  |  |
| --- | --- | --- |
| Mô tả | Mã nguồn | Kết quả minh họa |
| Khởi tạo một đối tượng BigInt từ chuỗi ký tự với cơ số 10 hoặc 16 | BigInt initBigInt(const char \*str, int base) {  BigInt num = {{0}, 0, 0};  int len = strlen(str);  int start = 0;  if (str[0] == '-') {  num.is\_negative = 1;  start = 1;  }  for (int i = len - 1; i >= start; i--) {  if ('0' <= str[i] && str[i] <= '9') {  num.digits[num.size++] = str[i] - '0';  } else if (base == 16 && ('a' <= str[i] && str[i] <= 'f')) {  num.digits[num.size++] = str[i] - 'a' + 10;  } else if (base == 16 && ('A' <= str[i] && str[i] <= 'F')) {  num.digits[num.size++] = str[i] - 'A' + 10;  }  }  return num;  } | a (base 10): 123456789012345678901234567890  b (base 10): 987654321098765432109876543210 |
| Chuyển đổi BigInt thành chuỗi theo cơ số 10 hoặc 16 | void toString(BigInt num, char \*str, int base) {  int index = 0;  if (num.is\_negative) {  str[index++] = '-';  }  for (int i = num.size - 1; i >= 0; i--) {  if (base == 16 && num.digits[i] >= 10) {  str[index++] = 'A' + (num.digits[i] - 10);  } else {  str[index++] = '0' + num.digits[i];  }  }  str[index] = '\0';  } | numHex (base 16): 1A2B3C4D5E6F |
| Chuyển đổi BigInt từ kiểu int của C | BigInt fromInt(int value) {  char str[MAX\_DIGITS];  sprintf(str, "%d", value);  return initBigInt(str, 10);  } | c (from int): 123456789 |
| Chuyển đổi BigInt sang kiểu int của C | int toInt(BigInt num) {  int result = 0;  for (int i = num.size - 1; i >= 0; i--) {  result = result \* 10 + num.digits[i];  }  return num.is\_negative ? -result : result;  } | c (to int): 123456789 |
| Cộng hai BigInt | BigInt addBigInt(BigInt a, BigInt b) {  BigInt result = {{0}, 0, 0};  int carry = 0, i;  for (i = 0; i < a.size || i < b.size || carry; i++) {  int digitA = (i < a.size) ? a.digits[i] : 0;  int digitB = (i < b.size) ? b.digits[i] : 0;  int sum = digitA + digitB + carry;  result.digits[result.size++] = sum % 10;  carry = sum / 10;  }  return result;  } | a + b = 1111111110111111111011111111100 |
| Trừ hai BigInt | BigInt subtractBigInt(BigInt a, BigInt b) {  BigInt result = {{0}, 0, 0};  int borrow = 0, i;  for (i = 0; i < a.size || i < b.size; i++) {  int digitA = (i < a.size) ? a.digits[i] : 0;  int digitB = (i < b.size) ? b.digits[i] : 0;  int sub = digitA - digitB - borrow;  if (sub < 0) {  sub += 10;  borrow = 1;  } else {  borrow = 0;  }  result.digits[result.size++] = sub;  }  while (result.size > 1 && result.digits[result.size - 1] == 0) {  result.size--;  }  return result;  } | a - b = 864197532086419753208641975320 |
| Nhân hai BigInt | BigInt multiplyBigInt(BigInt a, BigInt b) {  BigInt result = {{0}, 0, 0};  for (int i = 0; i < b.size; i++) {  BigInt temp = {{0}, 0, 0};  int carry = 0;  for (int j = 0; j < a.size || carry; j++) {  int prod = b.digits[i] \* ((j < a.size) ? a.digits[j] : 0) + carry;  temp.digits[temp.size++] = prod % 10;  carry = prod / 10;  }  for (int k = 0; k < i; k++) {  temp.digits[temp.size++] = 0;  }  result = addBigInt(result, temp);  }  return result;  } | a \* b = 1219326311370217952237463801111266061688456047352920 |
| Chia hai BigInt (trả về phần nguyên) | BigInt divideBigInt(BigInt a, BigInt b) {  BigInt result = {{0}, 0, 0};  BigInt remainder = {{0}, 0, 0};  for (int i = a.size - 1; i >= 0; i--) {  // Dịch chuyển các chữ số của remainder sang phải  for (int j = remainder.size; j > 0; j--) {  remainder.digits[j] = remainder.digits[j - 1];  }  remainder.digits[0] = a.digits[i];  remainder.size++;  // Loại bỏ các chữ số 0 thừa ở đầu  while (remainder.size > 1 && remainder.digits[remainder.size - 1] == 0) {  remainder.size--;  }  // Đếm số lần b trừ được từ remainder  int count = 0;  while (compareBigInt(remainder, b) >= 0) {  remainder = subtractBigInt(remainder, b);  count++;  }  result.digits[result.size++] = count;  }  // Loại bỏ các chữ số 0 thừa ở đầu kết quả  while (result.size > 1 && result.digits[result.size - 1] == 0) {  result.size--;  }  // Đảo ngược kết quả  for (int i = 0; i < result.size / 2; i++) | b / a = 8 |
| Lấy phần dư của phép chia hai BigInt | BigInt modBigInt(BigInt a, BigInt b) {  BigInt quotient = divideBigInt(a, b);  BigInt remainder = subtractBigInt(a, multiplyBigInt(quotient, b));  return remainder;  } | b % a = 864197532086419753208641975320 |
| Đối của BigInt | BigInt negateBigInt(BigInt a) {  a.is\_negative = !a.is\_negative;  return a;  } | negate(a) = -123456789012345678901234567890 |
| Tính trị tuyệt đối của BigInt | BigInt absBigInt(BigInt a) {  a.is\_negative = 0;  return a;  } | abs(a) = 123456789012345678901234567890 |
| Lũy thừa của BigInt với số mũ nguyên không âm | BigInt powerBigInt(BigInt base, unsigned int exponent) {  BigInt result = fromInt(1);  while (exponent > 0) {  if (exponent % 2 == 1) {  result = multiplyBigInt(result, base);  }  base = multiplyBigInt(base, base);  exponent /= 2;  }  return result;  } | a ^ 2 = 162 |
| So sánh hai số BigInt (a < b: -1, a == b: 0, a > b: 1) | // So sánh hai BigInt (a < b: -1, a == b: 0, a > b: 1)  int compareBigInt(BigInt a, BigInt b) {  if (a.size != b.size) {  return a.size > b.size ? 1 : -1;  }  for (int i = a.size - 1; i >= 0; i--) {  if (a.digits[i] != b.digits[i]) {  return a.digits[i] > b.digits[i] ? 1 : -1;  }  }  return 0;  }  // Kiểm tra bằng nhau  int equalsBigInt(BigInt a, BigInt b) {  return compareBigInt(a, b) == 0;  }  // Kiểm tra khác nhau  int notEqualsBigInt(BigInt a, BigInt b) {  return !equalsBigInt(a, b);  }  // Kiểm tra a < b  int lessThanBigInt(BigInt a, BigInt b) {  return compareBigInt(a, b) < 0;  }  // Kiểm tra a > b  int greaterThanBigInt(BigInt a, BigInt b) {  return compareBigInt(a, b) > 0;  }  // Kiểm tra a <= b  int lessOrEqualsBigInt(BigInt a, BigInt b) {  return compareBigInt(a, b) <= 0;  }  // Kiểm tra a >= b  int greaterOrEqualsBigInt(BigInt a, BigInt b) {  return compareBigInt(a, b) >= 0;  } | -1 ( a<b ) |
| Phép toán lấy phần nguyên của căn bậc 2 | BigInt sqrtBigInt(BigInt a) {  BigInt left = fromInt(0);  BigInt right = a;  BigInt mid, square;  while (compareBigInt(left, right) <= 0) {  mid = divideBigInt(addBigInt(left, right), fromInt(2));  square = multiplyBigInt(mid, mid);  if (compareBigInt(square, a) == 0) {  return mid;  } else if (compareBigInt(square, a) < 0) {  left = addBigInt(mid, fromInt(1));  } else {  right = subtractBigInt(mid, fromInt(1));  }  }  return right;  } | BigInt a1 = fromInt(16);  sqrt(a1) = 4 |
| Phép toán lấy phần nguyên của căn bậc 3 | BigInt cbrtBigInt(BigInt a) {  BigInt left = fromInt(0);  BigInt right = a;  BigInt mid, cube;  while (compareBigInt(left, right) <= 0) {  mid = divideBigInt(addBigInt(left, right), fromInt(2));  cube = multiplyBigInt(multiplyBigInt(mid, mid), mid);  if (compareBigInt(cube, a) == 0) {  return mid;  } else if (compareBigInt(cube, a) < 0) {  left = addBigInt(mid, fromInt(1));  } else {  right = subtractBigInt(mid, fromInt(1));  }  }  return right;  } | BigInt b1 = fromInt(27);  cbrt(b1) = 3 |