**BITSET**

#include <iostream>

#include <bitset>

using namespace std;

int main()

{

bitset<16> bit1;

bitset<16> bit2(8);

bitset<16> bit3("110111");

const char \*p = "1101";

bitset<16> bit4(p);

bitset<8> bit5;

cout << "bit1 : " << bit1 << "\n" << endl;

cout << "bit2 : " << bit2 << "\n" << endl;

cout << "bit3 : " << bit3 << "\n" << endl;

cout << "bit4 : " << bit4 << "\n" << endl;

cout << "bit5 : " << bit5 << "\n" << endl;

bit1.set(7, 1);

bit2.flip();

bit3.reset();

int onesCount = bit4.count();

bit5 = bitset<8>(bit4.to\_string().substr(8, 16));

cout << "bit1 8th bit changed to 1 : " << bit1 << "\n" << endl;

cout << "bit2 flipped : " << bit2 << "\n" << endl;

cout << "bit3 reset : " << bit3 << "\n" << endl;

cout << "bit4 ones count : " << onesCount << "\n" << endl;

cout << "bit5 re-initialized : " << bit5 << "\n" << endl;

cout << "bit5[0] access : " << bit5[0] << "\n" << endl;

cout << "bit5 test pos 1 : " << bit5.test(1) << "\n" << endl;

return 0;

}

/\*

\*Constructor:

std::bitset<N> bitset;: Default constructor creates a bitset with N bits, initialized to zero.

\*Conversion:

to\_ulong(): Converts the bitset to an unsigned long integer.

to\_ullong(): Converts the bitset to an unsigned long long integer.

to\_string(): Converts the bitset to a binary string.

\*Element Access:

operator[]: Accesses individual bits using the array subscript operator.

test(pos): Checks if the bit at position pos is set.

\*Counting and Testing:

count(): Counts the number of set bits in the bitset.

any(): Checks if any bit is set.

all(): Checks if all bits are set.

\*Modifiers:

set(pos): Sets the bit at position pos.

reset(pos): Resets the bit at position pos.

flip(pos): Flips the value of the bit at position pos.

reset(): Resets all bits to zero.

\*Size and Information:

size(): Returns the size (number of bits) of the bitset.

\*Bitwise Operations:

operator&, operator|, operator^: Performs bitwise AND, OR, and XOR operations.

operator~: Performs bitwise NOT operation.

operator<<, operator>>: Performs left and right shifts.

\*Relational Operators:

operator==, operator!=, operator<, etc.: Compare bitsets.

\*/

**Parity**

#include <iostream>

#include <bitset>

bool isEvenParity(const std::bitset<7>& binaryData) {

int countOnes = binaryData.count();

return (countOnes % 2 == 0);

}

int main() {

std::bitset<7> binaryData;

// Input 7-bit binary data

std::cout << "Enter 7-bit binary data (e.g., 1101101): ";

std::cin >> binaryData;

// Check parity

if (isEvenParity(binaryData)) {

std::cout << "Even parity check passed. Data is correct.\n";

} else {

std::cout << "Even parity check failed. Data may be corrupted.\n";

}

return 0;

}

**Parity : sender and receiver**

#include <iostream>

#include <bitset>

bool isEvenParity(const std::bitset<7>& binaryData) {

int countOnes = binaryData.count();

return (countOnes % 2 == 0);

}

std::bitset<8> addParityBit(const std::bitset<7>& binaryData) {

std::bitset<8> dataWithParity(binaryData.to\_string());

// Calculate parity bit and append it to the data

bool parityBit = isEvenParity(binaryData);

dataWithParity.set(7, !parityBit);

return dataWithParity;

}

bool checkParity(const std::bitset<8>& binaryData) {

// Only consider the first 7 bits for parity check

std::bitset<7> dataWithoutParity(binaryData.to\_string().substr(0, 7));

return !isEvenParity(dataWithoutParity);

}

int main() {

std::bitset<7> sendData;

std::bitset<8> receivedData;

// Sender

std::cout << "Sender:\n";

std::cout << "Enter 7-bit binary data to be sent (e.g., 1101101): ";

std::cin >> sendData;

// Add parity bit

std::bitset<8> sendDataWithParity = addParityBit(sendData);

std::cout << "Data with parity bit: " << sendDataWithParity << "\n\n";

// Receiver

std::cout << "Receiver:\n";

std::cout << "Enter received 8-bit binary data (including parity bit): ";

std::cin >> receivedData;

// Check parity

if (checkParity(receivedData)) {

std::cout << "Parity check passed. Data is correct.\n";

std::cout << "Received data without parity bit: " << receivedData << "\n";

} else {

std::cout << "Parity check failed. Data may be corrupted.\n";

}

return 0;

}