```
// Set em int
    int x = (1<<1)+(1<<3)+(1<<4)+(1<<8);    // set {1,3,4,8}
    int y = (1<<3)+(1<<6)+(1<<8)+(1<<9);    // set {3,6,8,9}
    int z = x|y;                            // union of the sets
    for (int i = 0; i < 32; i++)            // print the elements in the union
        if (z&(1<<i)) cout << i << " ";
    cout << endl;

// Informações do binário de um número
    cout << __builtin_clz(x) << endl; // Número de zeros no inicio (mais significativos)
    cout << __builtin_ctz(x) << endl; // Número de zeros no final (menos significativos)
    cout << __builtin_popcount(x) << endl; // Número de Uns
    cout << __builtin_parity(x) << endl; // Se o número de uns ímpar

// Inteiro para binário
    for (int i = 31; i >= 0; i--)
        if (x & (1<<i)) cout << "1";
        else cout << "0";
    cout << endl;

// String em binário para inteiro
    string binaryCharArray="01111111111111111111111111111111";
    char *start = &binaryCharArray[0];
    while (*start)
        x = (x << 1) | (*start++ & 1);
    cout << x << endl;


// Catalan number
    typedef unsigned long long ull;
    ull DP[37] = { 0 };     // 0..36, 37 já dá overflow
    ull catalan(ull n) {
        if (n <= 1) return 1;
        if (DP[n] != 0) return DP[n];
        ull i, res{0};
        for (i = 0; i < n; i++)
            res += catalan(i) * catalan(n-i-1);
        return DP[n] = res;
    }


// Quebra string em vector usando delimitador
    #include <string.h>
    vector<string> explode(string s, const char *delim) {
        char buf[MAX], *p;  // trocar para tamanho máximo da palavra
        vector<string> ret;
        sprintf(buf, "%s", s.c_str());
        for(p = strtok(buf, delim); p != NULL; p = strtok(NULL, delim))
            ret.push_back(p);
        return ret;
    }


// Knapsack 0,1 max
    #define MAX 100
    int auxTable[MAX][MAX], cost[MAX], bnft[MAX];

    void cleanTable(int N, int W) {
        for(int i = 0; i <= N; ++i)
            for(int j = 0; j <= W; ++j)
                auxTable[i][j] = (i==0 || j==0 ? 0 : -1);
    }
    inline int myMax(int a, int b) { return a > b ? a : b; }

    int knapsack(int i, int w) {
        if(auxTable[i][w] != -1) return auxTable[i][w];
        if(cost[i] > w) auxTable[i][w] = knapsack(i - 1, w);
        else auxTable[i][w] = myMax(knapsack(i - 1, w),
                                    knapsack(i - 1, w - cost[i]) + bnft[i]);
        return auxTable[i][w];
    }
    int main() {
        int totalWeight, numItens;
        cin >> numItens >> totalWeight;
        cleanTable(numItens, totalWeight);
        // bnft[0] = cost[0] = 0;
        for (int i = 1; i <= numItens; ++i) cin >> bnft[i] >> cost[i];
                cout << knapsack(numItens, totalWeight) << endl;
        return 0;
    }


// Conta inversões com merge sort
    #define INF 1000000000
    int merge_count(vector<int> &v){
        if(v.size()==1) return 0;
        int inv = 0; // Número de inversões
        vector<int> u1, u2;
        for(int i = 0; i < (int)v.size()/2; i++) u1.push_back(v[i]);
        for(int i = v.size()/2; i < (int)v.size(); i++) u2.push_back(v[i]);
        inv += merge_count(u1) + merge_count(u2);
        u1.push_back(INF);
        u2.push_back(INF);
        int ini1=0, ini2=0;
        for(int i = 0; i < (int)v.size(); i++)
            if(u1[ini1] <= u2[ini2])
                v[i] = u1[ini1++];
            else {
                v[i] = u2[ini2++];
                inv += u1.size()-ini1-1;
            }
        return inv;
    }


// Union Find
    vector<int> p, myRank;
    int numSets;
    void unionFind(int N) {
        numSets = N;
        myRank.assign(N, 0);
        p.assign(N, 0);
        for (int i = 0; i < N; ++i)
            p[i] = i;
    }
    int findSet(int i) {
        return (p[i] == i) ? i : (p[i] = findSet(p[i]));
    }
    bool isSameSet(int i, int j) {
        return findSet(i) == findSet(j);
    }
    void unionSet(int i, int j) {
        if (!isSameSet(i, j)) {
            int x = findSet(i), y = findSet(j);
            if (myRank[x] > myRank[y])
                p[y] = x;
            else {
                p[x] = y;
                if (myRank[x] == myRank[y])
                    myRank[y]++;
            }
            numSets--;
        }
    }


// Números primos segundo Crivo de Eratóstenes
    #include <bitset>
    #include <vector>
    typedef long long ll;
    typedef vector<int> vi;

    ll _sieve_size;
    bitset<10000010> bs;
    vi primes;
    void sieve(ll upperbound) {
        _sieve_size = upperbound + 1;
        bs.set();
        bs[0] = bs[1] = 0;
        for (ll i = 2; i <= _sieve_size; i++)
            if (bs[i]) {
                for (ll j = i * i; j <= _sieve_size; j += i)
                    bs[j] = 0;
                primes.push_back((int)i);
            }
    }
    bool isPrime(ll N) {
        if (N <= _sieve_size)
            return bs[N];
        for (int i = 0; i < (int)primes.size(); i++)
            if (N % primes[i] == 0)
                return false;
        return true;
    }


// KMP
    #define MAX_N 100010
    char T[MAX_N], P[MAX_N];
    int b[MAX_N], n, m;

    void kmpPreprocess() {
        call this before calling kmpSearch() int i = 0, j = -1;
        b[0] = -1;
        while (i < m) {
            while (j >= 0 && P[i] != P[j])
                j = b[j]; // different, reset j using b
            i++;
            j++;
            b[i] = j; // observe i = 8, 9, 10, 11, 12, 13 with j = 0, 1, 2, 3, 4, 5
        }
    }
    void kmpSearch() {
        int i = 0, j = 0;
        while (i < n) {
            while (j >= 0 && T[i] != P[j])
                j = b[j]; // different, reset j using b
            i++; j++;
            if (j == m) {
                printf("P is found at index %d in T\n", i - j);
                j = b[j];
            }
        }
    }

// Ordenação Topológica - algoritmo de Kahn
    vector<set<int> > LG;
    vector<int> ts, inc;

    void kahnTopoSort() {
        ts.clear();
        priority_queue<int, vector<int>, greater<int> > Q;
        for(int i = 0; i <= N + 1; ++i)
            if(inc[i] == 0)
                Q.push(i);

        while(!Q.empty()) {
            int u = Q.top(); Q.pop();
            ts.push_back(u);
            for(int w : LG[u])
                if(--inc[w] == 0)
                    Q.push(w);
        }
    }


// BFS para computar a distância entre S e T
    int V, E;
    vector<string> cidades;
    map<string, vector<string> > adjList;

    int bfs(string s, string t) {
        string u, w;
        map<string, int> dist;
        queue<string> fila;
        for(int i = 0; i < V; i++)
            dist[cidades[i]] = INFTO;
        dist[s] = 0;
        fila.push(s);
        while(!fila.empty()) {
            u = fila.front();
            fila.pop();
            int tam = (int) adjList[u].size();
            for(int i = 0; i < tam; i++) {
                w = adjList[u][i];
                if(dist[w] == INFTO) {
                    dist[w] = dist[u] + 1;
                    fila.push(w);
                }
            }
        }
        return dist[t];
    }

// DFS: detecta se tem ciclo no grafo
    int N;
    vector <vector <int> > LG;

    bool dfs(int s, int t) {
        int u, w;
        bool temCiclo = false;
        vector<int> pai(N, -1);
        vector<bool> vis(N, false);
        vis[s] = true;
        pai[s] = s;
        stack<int> pilha;
        pilha.push(s);
        while(!pilha.empty()) {
            u = pilha.top();
            pilha.pop();
            for(w : LG[u])
                if(!vis[w]) {
                    vis[w] = true;
                    pai[w] = u;
                    pilha.push(w);
                }
                else if(w != pai[u])
                    temCiclo = true;
        }
        return temCiclo;
    }
```