1.

```
#include <iostream>

struct node {
	int key;
	node* prev, * next;
};

struct list {
	node* head = nullptr, * tail = nullptr;

	void pushFront(int value) {
		node* newNode = new node;
		newNode->key = value;
		if (head == nullptr) {
			head = newNode;
			tail = newNode;
			newNode->next = nullptr;
			newNode->prev = nullptr;
		}
		else {
			newNode->next = head;
			newNode->prev = nullptr;
			head->prev = newNode;
			head = newNode;
		}
	}

	void pushBack(int value) {
		node* newNode = new node;
		newNode->key = value;
		newNode->next = nullptr;
		newNode->prev = tail;
		if (head == nullptr)
			head = newNode;
		else
			tail->next = newNode;
		tail = newNode;
	}

	void popFront() {
		if (head == nullptr)
			return;
		node* toErase = head;
		if (head != tail) {
			head = head->next;
			head->prev = nullptr;
		}
		else {
			head = nullptr;
			tail = nullptr;
		}
		delete toErase;
	}

	void popBack() {
		if (head == nullptr)
			return;
```

```
		node* toErase = tail;
		if (head != tail) {
			tail = tail->prev;
			tail->next = nullptr;
		}
		else {
			head = nullptr;
			tail = nullptr;
		}
		delete toErase;
}

node* find(int value) {
		node* current = head;
		while (current != nullptr) {
			if (current->key == value)
				return current;
			current = current->next;
		}
		return nullptr;
}

void erase(node* Nod) {
		if (Nod == head) {
			popFront();
			return;
		}
		if (Nod == tail) {
			popBack();
			return;
		}
		else {
			node* toErase = Nod;
			Nod->prev->next = Nod->next;
			Nod->next->prev = Nod->prev;
			delete toErase;
		}
}

void remove(int value) {
		node* current = head;
		while (current != nullptr) {
			if (current->key == value) {
				node* aux = current->next;
				erase(current);
				current = aux;
			}
			else
				current = current->next;
		}
		if (current == tail)
			popBack();
}

void insert(node* Nod, int value) {
		if (Nod == nullptr)
			return;
		if (Nod == head) {
			pushFront(value);
```

```
                return;
            }
            node* newNode = new node;
            newNode->key = value;
            Nod->prev->next = newNode;
            newNode->prev = Nod->prev;
            Nod->prev = newNode;
            newNode->next = Nod;
        }

        void empty() {
            if (head == nullptr) {
                std::cout << "Lista este vida." << std::endl;
                return;
            }
            std::cout << "Lista are elemente." << std::endl;
        }

        void clear() {
            node* current = head;
            node* aux;
            while (current != nullptr) {
                aux = current->next;
                popFront();
                current = aux;
            }
        }

        void print() {
            node* current = head;
            while (current != nullptr) {
                std::cout << current->key << " ";
                current = current->next;
            }
            std::cout << std::endl;
        }

        int size() {
            node* current = head;
            int nrOfElements = 0;
            while (current) {
                nrOfElements++;
                current = current->next;
            }
            return nrOfElements;
        }

};

bool palindrom(list L) {
        node* front = L.head;
        node* back = L.tail;
        if (!front)
            return false;
        while (front) {
            if (front->key != back->key)
                return false;
            front = front->next;
            back = back->prev;
```

```
	}
	return true;
}

bool compare(list L1, list L2) {
	node* current1 = L1.head, * current2 = L2.head;
	int nr1 = L1.size();
	int nr2 = L2.size();
	if (nr1 != nr2)
		return false;
	while (current1) {
		if (current1->key != current2->key)
			return false;
		current1 = current1->next;
		current2 = current2->next;
	}
	return true;
}

int main()
{
	list Lista, Listaa;
	int choice = 0;
	int value;
	std::string run = "Porneste";
	while (run != "EXIT") {
		std::cout << "Introduceti comanda: ";
		std::cin >> choice;
		switch (choice) {
		case 0:
			std::cout << "Multumim pentru atentie!" << std::endl;
			run = "EXIT";
			break;
		case 1:
			std::cout << "Elementul pe care vreti sa il adaugati la inceputul listei este: ";
			std::cin >> value;
			Lista.pushFront(value);
			std::cout << "Lista arata acum asa: ";
			Lista.print();
			std::cout << std::endl;
			break;
		case 2:
			std::cout << "Elementul pe care vreti sa il adaugati la sfarsitul listei este: ";
			std::cin >> value;
			Lista.pushBack(value);
			std::cout << "Lista arata acum asa: ";
			Lista.print();
			std::cout << std::endl;
			break;
		case 3:
			std::cout << "Vom sterge un element de la inceputul listei." << std::endl;
			Lista.popFront();
			std::cout << "Lista arata acum asa: ";
			Lista.print();
			std::cout << std::endl;
			break;
		case 4:
			std::cout << "Vom sterge un element de la sfarsitul listei." << std::endl;
			Lista.popBack();
```

```
                        std::cout << "Lista arata acum asa: ";
                        Lista.print();
                        std::cout << std::endl;
                        break;
                case 5:
                        std::cout << "Elementul pe care il cautati este: ";
                        std::cin >> value;
                        if (!Lista.find(value))
                                std::cout << "Valoarea respectiva nu se afla in lista." << std::endl;
                        else
                                std::cout << "Valoarea cautata se afla la adresa: " << Lista.find(value) << std::endl
<< std::endl;
                        break;
                case 6:
                        std::cout << "Nodul pe care vreti sa l stergeti are valoarea: ";
                        std::cin >> value;
                        Lista.erase(Lista.find(value));
                        std::cout << "Lista arata acum asa: ";
                        Lista.print();
                        std::cout << std::endl;
                        break;
                case 7:
                        std::cout << "Nodul pe care vreti sa l eliminati complet din lista este: ";
                        std::cin >> value;
                        Lista.remove(value);
                        std::cout << "Lista arata acum asa: ";
                        Lista.print();
                        std::cout << std::endl;
                        break;
                case 8:
                        int value2;
                        std::cout << "Doriti sa inserati valoarea: ";
                        std::cin >> value;
                        std::cout << "inainte de valoarea: ";
                        std::cin >> value2;
                        Lista.insert(Lista.find(value2), value);
                        std::cout << "Lista arata acum asa: ";
                        Lista.print();
                        std::cout << std::endl;
                        break;
                case 9:
                        Lista.empty();
                        std::cout << std::endl;
                        break;
                case 10:
                        std::cout << "Stergem toate elementele listei."<<std::endl;
                        Lista.clear();
                        std::cout << "Lista arata acum asa: ";
                        Lista.print();
                        std::cout << std::endl;
                        break;
                case 11:
                        std::cout << "Numarul de elemente din lista: "<<Lista.size();
                        std::cout << std::endl << std::endl;
                        break;
                default:
                        std::cout << "Comanda nu a fost gasita. Introduceti alta comanda!" << std::endl << std::endl;
                }
        }
```

```
	return 0;
}
```

2.

```
#include <iostream>
#include <fstream>
#include <vector>

struct node {
	float key;
	node* next, * prev;
};

struct list {
	node* head = nullptr;
	node* tail = nullptr;
	void pushBack(float value) {
		node* newNode = new node;
		newNode->key = value;
		newNode->next = nullptr;
		newNode->prev = tail;
		if (head == nullptr)
			head = newNode;
		else
			tail->next = newNode;
		tail = newNode;
	}
	void print() {
		node* current = head;
		while (current != nullptr) {
			std::cout << current->key << " ";
			current = current->next;
		}
	}
	bool empty() {
		if (head == nullptr) {
			return false;
		}
		return true;
	}
	void insertionSort() {
		float key;
		node* current = head->next;
		while (current) {
			key = current->key;
			node* N = current->prev;
			while (N != nullptr && key < N->key) {
				N->next->key = N->key;
				N = N->prev;
			}
			if (!N)
				head->key = key;
			else
				N->next->key = key;
			current = current->next;
		}
	}
```

```
};

void bucketSort(int nr, std::vector<float>arr) {
	for (int i = 0; i < 10; i++) {
		list Lista;
		for (int j = 0; j < nr; j++) {
			int numar = arr[j] * 10;
			if (numar == i) {
				Lista.pushBack(arr[j]);
			}
		}
		if (Lista.empty()) {
			Lista.insertionSort();
			Lista.print();
		}
		else {
			delete Lista.head;
			delete Lista.tail;
		}
	}
	std::cout << std::endl;
}

int main()
{
	int nr;
	float x;
	std::vector<float>arr;
	std::ifstream fin("date.in");
	fin >> nr;
	std::cout << "Numarul de elemente din vectorul initial: " << nr << std::endl;
	std::cout << "Elementele din vector sunt: ";
	for (int i = 0; i < nr; i++) {
		fin >> x;
		std::cout << x << " ";
		arr.push_back(x);
	}
	std::cout << std::endl << "Vectorul sortat arata asa: ";
	bucketSort(nr, arr);
	fin.close();
	return 0;
}
```

3.

```
#include <iostream>
#include <fstream>
#include <vector>
#include <string>

struct node {
	std::string key;
	node* next = nullptr;
};

struct Queue {
	node* head = nullptr;
	node* last = nullptr;
	int nr_elem = 0;

	void push(std::string elem) {
		node* newNode = new node;
		newNode->key = elem;
		if (!head)
			head = newNode;
		else
			last->next = newNode;
		last = newNode;
		newNode->next = nullptr;
		nr_elem++;
	}

	void pop() {
		node* toErase = head;
		head = head->next;
		delete toErase;
		nr_elem--;
	}

	std::string front() {
		return head->key;
	}

	std::string back() {
		return last->key;
	}

	void print() {
		node* current = head;
		while (current != nullptr) {
			std::cout << current->key << " ";
			current = current->next;
		}
	}

	bool empty() {
		if (head)
			return true;
		return false;
	}

	void clear() {
```

```
		while (head)
			pop();
	}

	int size() {
		return nr_elem;
	}
};

void citireDinFisier(int& nrCandidati, int& timp, Queue& numeCandidati) {
	std::ifstream fin("candidati.in");
	std::ifstream fiin("candidatii.in");
	fin >> nrCandidati >> timp;
	std::cout << "Numarul de candidati: " << nrCandidati << std::endl;
	std::cout << "Numarul de ore alocate in prima zi: " << timp << std::endl;
	std::string line;

	while (!fiin.eof()) {
		std::getline(fiin, line);
		numeCandidati.push(line);
	}
	fin.close();
	fiin.close();
}

void extragereCandidati(Queue& numeCandidati, int timp) {
	int timpPrimaZi = timp * 60, minuteCandidat;
	node* current = numeCandidati.head;
	while (timpPrimaZi != 0 && current != nullptr) {
		std::cout << "Numarul de min pt candidatul curent: ";
		std::cin >> minuteCandidat;
		timpPrimaZi -= minuteCandidat;
		current = current->next;
		numeCandidati.pop();
	}
}

void candidatiRamasi(Queue numeCandidati) {
	std::ofstream fout("candidati.out");
	node* current = numeCandidati.head;
	while (current) {
		fout << current->key << " ";
		current = current->next;
	}
	fout.close();
}

int main()
{

	Queue numeCandidati;

	int nrCandidati = 0, timp = 0;
	citireDinFisier(nrCandidati, timp, numeCandidati);
	extragereCandidati(numeCandidati, timp);
	candidatiRamasi(numeCandidati);
	return 0;
}
```

4.

```
#include <iostream>
#include <stack>
#include <string>

bool esteCorect(std::string paranteze) {
    std::stack<char> stiva;
    for (char paranteza : paranteze) {
        if (paranteza == '(' || paranteza == '[' || paranteza == '{') {
            stiva.push(paranteza);
        }
        else if (paranteza == ')' || paranteza == ']' || paranteza == '}') {
            if (stiva.empty()) {
                return false;
            }
            char ultimaParantezaDeschisa = stiva.top();
            stiva.pop();
            if (ultimaParantezaDeschisa == '{' && !stiva.empty())
                return false;
            else if (ultimaParantezaDeschisa == '[' && !stiva.empty() && stiva.top() == '(')
                return false;
            if ((paranteza == ')' && ultimaParantezaDeschisa != '(') ||
                (paranteza == ']' && ultimaParantezaDeschisa != '[') ||
                (paranteza == '}' && ultimaParantezaDeschisa != '{')) {
                return false;
            }
        }
    }
    return stiva.empty();
}

int main() {
    std::string paranteze1 = "[{()()}]";
    std::string paranteze2 = "(){[]}()";
    std::string paranteze3 = "[()]()";
    std::cout << paranteze1 << " este corect: " << esteCorect(paranteze1) << std::endl;
    std::cout << paranteze2 << " este corect: " << esteCorect(paranteze2) << std::endl;
    std::cout << paranteze3 << " este corect: " << esteCorect(paranteze3) << std::endl;
    return 0;
}
```

5.

```
#include <iostream>
#include <queue>
#include <string>


struct stiva {
    std::queue<char> q1, q2;

    void Push(char val) {
        q2.push(val);
        while (!q1.empty()) {
            q2.push(q1.front());
            q1.pop();
```

```
        }
        std::swap(q1, q2);
    }

    void Pop() {
        while (!q1.empty())
            q1.pop();
    }
};

bool parantezareCorecta(std::string paranteze) {
    stiva st;
    for (char paranteza : paranteze) {
        if (paranteza == '(' || paranteza == '[' || paranteza == '{')
            st.Push(paranteza);
        else if (paranteza == ')' || paranteza == ']' || paranteza == '}') {
            if (st.q1.empty())
                return false;
            char ultimaParantezaDeschisa = st.q1.front();
            st.q1.pop();
            if (ultimaParantezaDeschisa == '{' && !st.q1.empty())
                return false;
            else if (ultimaParantezaDeschisa == '[' && !st.q1.empty() && st.q1.front() == '(')
                return false;
            if ((paranteza == ')' && ultimaParantezaDeschisa != '(')
                || (paranteza == ']' && ultimaParantezaDeschisa != '[')
                || (paranteza == '}' && ultimaParantezaDeschisa != '{'))
                return false;
        }
    }
    return st.q1.empty();
}

int main()
{
    std::string paranteze1 = "[()]";
    std::string paranteze2 = "}[()]";
    std::string paranteze3 = "[{()}]";
    std::cout << "Parantezare corecta: " << parantezareCorecta(paranteze1) << std::endl;
    std::cout << "Parantezare corecta: " << parantezareCorecta(paranteze2) << std::endl;
    std::cout << "Parantezare corecta: " << parantezareCorecta(paranteze3) << std::endl;
    return 0;
}
```