Risposte Esame di Programmazione - 2021-12-22

Esercizio 1A

Realizzare due struct indirizzo e cliente:

```
struct Indirizzo {
    std::string via;
    int numero_civico;
    std::string CAP;
    std::string citta;
};

struct Cliente {
    std::string cognome;
    std::string nome;
    Indirizzo indirizzo;
};
```

Esercizio 1B

Realizzare una funzione che verifichi se due clienti abitano nella stessa zona della citta.

```
bool stessaZona(Cliente c1, Cliente c2) {
    return c1.indirizzo.CAP == c2.indirizzo.CAP;
}
```

Esercizio 1C

Risposte Esame di Programmazione - 2021-12-22

Realizzare il prototipo di una funzione che acquisisca gli opportuni parametri formali, passati in modo adeguato, per aggiornare l?indirizzo di un cliente.

```
void aggiornaIndirizzo(Cliente &c, Indirizzo nuovoIndirizzo);
```

Esercizio 2A

Assumiamo di voler implementare il tipo di dato ?coda di clienti? basandoci sui vector. Produrre i prototipi (o interfacce) delle 3 funzioni principali:

```
void enqueue(std::vector<Cliente> &coda, Cliente cliente);
void dequeue(std::vector<Cliente> &coda);
Cliente front(const std::vector<Cliente> &coda);
```

Esercizio 2B

Implementare la funzione enqueue trattando opportunamente il caso coda vuota.

```
void enqueue(std::vector<Cliente> &coda, Cliente cliente) {
    coda.push_back(cliente);
}
```

Esercizio 3A

Realizzare una funzione ricorsiva che inserisca un elemento in ordine in una lista.

```
void insert_elem(int x, lista &l) {
```

```
    if ((l == nullptr) || (l->head > x)) { //base
        // inserisci in testa
        cell *aux = new cell;
        aux->head = x;
        aux->next = l;
        l = aux;
    } else {
        insert_elem(x, l->next); // inserisci nel resto
    }
}
```

Esercizio 3B

Realizzare una funzione booleana che restituisce true se due liste sono consecutive (il primo elemento della 2a lista è maggiore dell?ultimo della 1a).

```
bool listeConsecutive(lista l1, lista l2) {
    if (l1 == NULL || l2 == NULL) {
        return false;
    }
    while (l1->next != NULL) {
        l1 = l1->next;
    }
    return l1->head < l2->head;
}
```