Laporan Tugas Besar Struktur Data

Program Rental Film

Disusun oleh Kelompok 1 :

Josua Michael E. S. (103092400006)

Kayana Hayu C. (103092400017)

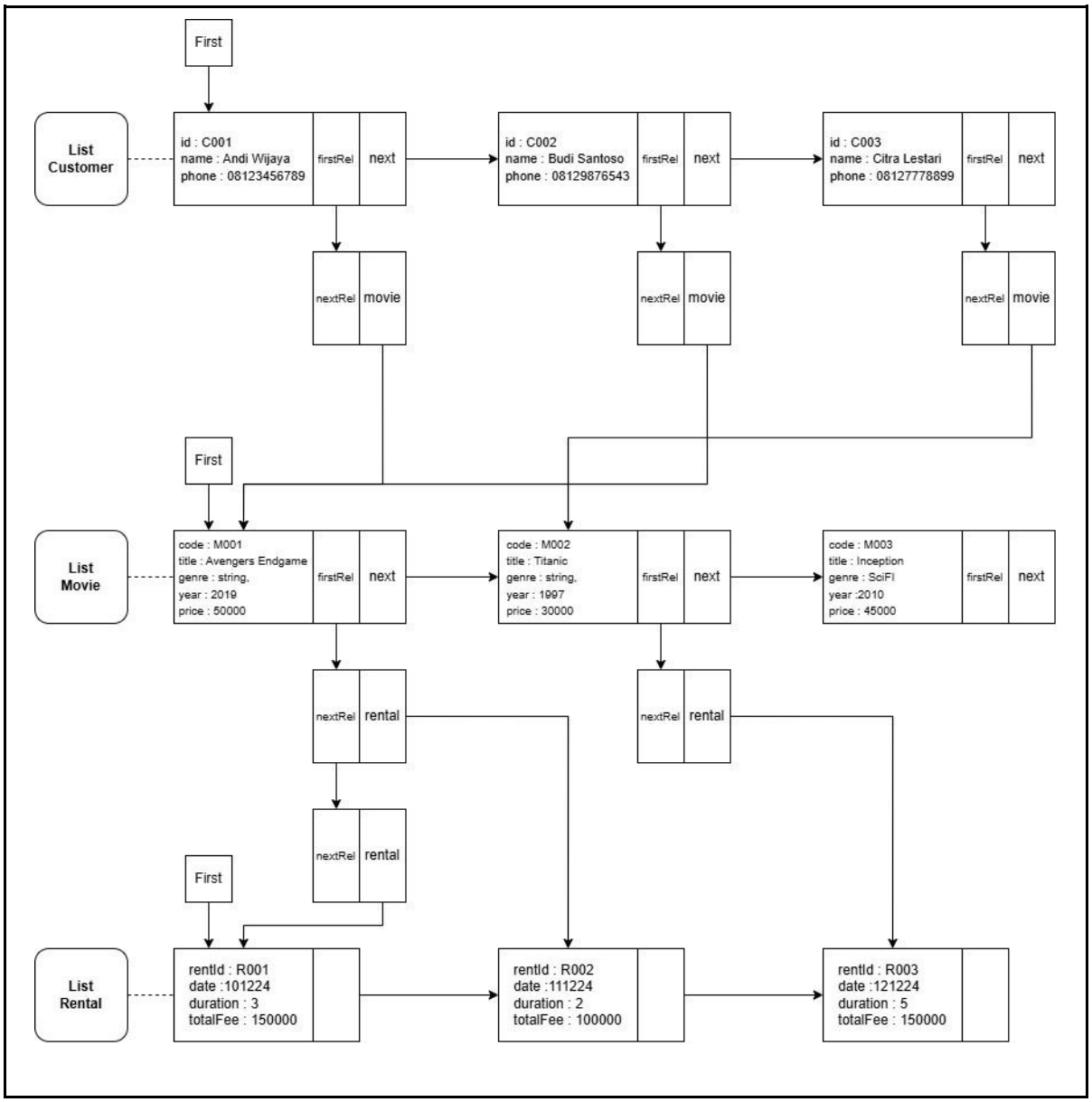
M. Virza Akbar H. (103092400013)

Mata Kuliah : Struktur Data

Dosen Pengampu : Yohanes Setiawan, S.Si., M.Kom.

Prodi Teknologi Informasi, Fakultas Informatika, Universitas Telkom, 2025

1. JUDUL : Program Rental Film
2. Tipe MLL : Tipe A (III-B, M-N)
3. Jenis list parent (customer) : SLL
4. Jenis list parent (movie) : SLL
5. Jenis list child (rental) : SLL
6. Model MLL :



7. Data Customer :

- ID customer : string
- Name : string
- Phone : string

8. Data Movie :

- Code : string
- Title : string
- Genre : string
- Year : integer
- Price : real

9. Data Rental

- rentId : string
- Date : string
- Duration : integer
- TotalFee : real

Spesifikasi Program :

Implementasi multi linked-list yang memodelkan data customer, film, dan transaksi penyewaan. Suatu customer bisa menyewa banyak film. Dan suatu film bisa disewa oleh banyak customer, dan fee rental dihitung berdasarkan durasi dan kode movienya.

Fungsionalitas:

a. Penambahan data customer.

b. Penambahan data film.

c. Penambahan data transaksi rental.

d. Penentuan relasi customer dan film yang disewa.

e. Menghapus data customer.

f. Menghapus data film.

g. Menghapus data transaksi rental.

h. Menampilkan data keseluruhan customer beserta data film yang disewanya.

i. Menampilkan data film yang disewa oleh customer tertentu.

j. Menampilkan data customer yang menyewa film tertentu.

k. Menampilkan data film yang paling banyak disewa dan yang paling sedikit.

Penjelasan kode setiap bagian :

a.	Insert Element Parent (5) Customer (Parent) PIC :

```
19
20  void insertFirstCustomer(listCustomer &LC, elementCustomer* p) {
21      if (LC.first == nullptr) {
22          LC.first = p;
23      } else {
24          p->next = LC.first;
25          LC.first = p;
26      }
27  }
28
29  void insertLastCustomer(listCustomer &LC, elementCustomer* p) {
30      if (LC.first == nullptr) {
31          LC.first = p;
32      } else {
33          elementCustomer* current = LC.first;
34          while (current->next != nullptr) {
35              current = current->next;
36          }
37          current->next = p;
38      }
39  }
40
41  void insertAfterCustomer(elementCustomer* prec, elementCustomer* p) {
42      if (prec != nullptr) {
43          p->next = prec->next;
44          prec->next = p;
45      }
46  }
47
48  void insertElementCustomer(listCustomer &LC, elementCustomer* p, string method, elementCustomer* prec) {
49      if (method == "first") {
50          insertFirstCustomer(LC, p);
51      } else if (method == "last") {
52          insertLastCustomer(LC, p);
53      } else if (method == "after" && prec != nullptr) {
54          insertAfterCustomer(prec, p);
55      }
56  }
```

Penjelasan :

insertFirstCustomer menyisipkan node customer di awal list sebagai elemen pertama dalam daftar, insertLastCustomer untuk menambah node di akhir list dengan menelusuri node sampai yang terakhir lalu menghubungkannya ke node baru, dan insertAfterCustomer untuk menyisipkan node setelah node tertentu dengan mengatur pointer next agar node baru berada setelah node yang ditunjuk. Ketiga fungsi ini memungkinkan penambahan data customer di posisi awal, akhir, maupun tengah list. InsertElementCustomer membungkus prosedur lain agar lebih mudah dipanggil di menu main.

a.	Insert Element Parent **(5)** Movie (Parent) PIC : Virza

```
20
21  void insertFirstMovie(listMovie &LM, elementMovie* p) {
22      if (LM.first == nullptr) {
23          LM.first = p;
24      } else {
25          p->next = LM.first;
26          LM.first = p;
27      }
28  }
29
30  void insertLastMovie(listMovie &LM, elementMovie* p) {
31      if (LM.first == nullptr) {
32          LM.first = p;
33      } else {
34          elementMovie* current = LM.first;
35          while (current->next != nullptr) {
36              current = current->next;
37          }
38          current->next = p;
39      }
40  }
41
42  void insertAfterMovie(elementMovie* prec, elementMovie* p) {
43      if (prec != nullptr) {
44          p->next = prec->next;
45          prec->next = p;
46      }
47  }
48
49  void insertElementMovie(listMovie &LM, elementMovie* p, string method, elementMovie* prec) {
50      if (method == "first") {
51          insertFirstMovie(LM, p);
52      } else if (method == "last") {
53          insertLastMovie(LM, p);
54      } else if (method == "after" && prec != nullptr) {
55          insertAfterMovie(prec, p);
56      }
```

Penjelasan :

Algoritmanya sama seperti insertCustomer, cuma beda pakai list movie saja.

b.	Insert Element Child **(5)** Rental (Child) PIC : Josua

```
19
20  void insertFirstRental(listRental &LR, elementRental* p) {
21      if (LR.first == nullptr) {
22          LR.first = p;
23      } else {
24          p->next = LR.first;
25          LR.first = p;
26      }
27  }
28
29  void insertLastRental(listRental &LR, elementRental* p) {
30      if (LR.first == nullptr) {
31          LR.first = p;
32      } else {
33          elementRental* current = LR.first;
34          while (current->next != nullptr) {
35              current = current->next;
36          }
37          current->next = p;
38      }
39  }
40
41  void insertAfterRental(elementRental* prec, elementRental* p) {
42      if (prec != nullptr) {
43          p->next = prec->next;
44          prec->next = p;
45      }
46  }
47
48  void insertElementRental(listRental &LR, elementRental* p, string method, elementRental* prec) {
49      if (method == "first") {
50          insertFirstRental(LR, p);
51      } else if (method == "last") {
52          insertLastRental(LR, p);
53      } else if (method == "after" && prec != nullptr) {
54          insertAfterRental(prec, p);
55      }
```

Penjelasan :

Algoritmanya sama seperti insertCustomer, cuma beda pakai list rental saja.

c.	Insert element relation **(5)** Customer-Movie Relation & Movie-Rental Relation PIC :

```
18  void insertElementRelCustMovie(elementCustomer* customer, elementMovie* movie) {
19      relCustMovie* newRel = createElementRelCustMovie(movie);
20
21      if (customer->firstRel == nullptr) {
22          customer->firstRel = newRel;
23      } else {
24          relCustMovie* current = customer->firstRel;
25          while (current->nextRel != nullptr) {
26              current = current->nextRel;
27          }
28          current->nextRel = newRel;
29      }
30  }
```

```
192 void insertElementRelMovieRent(elementMovie* movie, elementRental* rental) {
193     relMovieRent* newRel = createElementRelMovieRent(rental);
194
195     if (movie->firstRel == nullptr) {
196         movie->firstRel = newRel;
197     } else {
198         relMovieRent* current = movie->firstRel;
199         while (current->nextRel != nullptr) {
200             current = current->nextRel;
201         }
202         current->nextRel = newRel;
203     }
204 }
```

Penjelasan :

Jika node firstrel dari list parent kosong, maka langsung menunjuk node baru (newRel), jika tidak dilakukan penyisipan diakhir list (InsertLast), algoritma untuk kedua List Relasi sama, hanya listnya yang berbeda.

d.	Delete Element Parent **(5)** Delete Customer PIC :

```
58  void deleteFirstCustomer(listCustomer &LC, elementCustomer* &p) {
59      if (LC.first == nullptr) {
60          p = nullptr;
61      } else {
62          p = LC.first;
63          LC.first = LC.first->next;
64          p->next = nullptr;
65      }
66  }
67
68  void deleteLastCustomer(listCustomer &LC, elementCustomer* &p) {
69      if (LC.first == nullptr) {
70          p = nullptr;
71      } else if (LC.first->next == nullptr) {
72          p = LC.first;
73          LC.first = nullptr;
74      } else {
75          elementCustomer* current = LC.first;
76          while (current->next->next != nullptr) {
77              current = current->next;
78          }
79          p = current->next;
80          current->next = nullptr;
81      }
82  }
83
84  void deleteAfterCustomer(elementCustomer* prec, elementCustomer* &p) {
85      if (prec != nullptr && prec->next != nullptr) {
86          p = prec->next;
87          prec->next = p->next;
88          p->next = nullptr;
89      } else {
90          p = nullptr;
91      }
92  }
```

```
94  void deleteElementCustomer(listCustomer &LC, elementCustomer* &p, string method, elementCustomer* prec) {
95      if (method == "first") {
96          deleteFirstCustomer(LC, p);
97      } else if (method == "last") {
98          deleteLastCustomer(LC, p);
99      } else if (method == "after" && prec != nullptr) {
100         deleteAfterCustomer(prec, p);
101     }
102 }
103
```

Penjelasan :

DeleteFirstCustomer menghapus node pertama dan memperbarui penunjuk awal daftar. DeleteLastCustomer menghapus node terakhir dengan cara menelusuri daftar untuk menemukan node kedua dari akhir (prev), lalu memutuskan sambungan ke node terakhir tersebut. Sementara itu, deleteAfterCustomer menghapus node yang berada setelah node tertentu (prec) yang sudah ditentukan. Secara keseluruhan, fungsi-fungsi ini memungkinkan

penghapusan data pelanggan yang fleksibel dari posisi mana pun dalam struktur data. Prosedur deleteElementCustomer membungus prosedur delete lain agar lebih mudah dipakai di menu main.

d.	Delete Element Parent **(5)** Delete Movie PIC : Virza

```
59  void deleteFirstMovie(listMovie &LM, elementMovie* &p) {
60      if (LM.first == nullptr) {
61          p = nullptr;
62      } else {
63          p = LM.first;
64          LM.first = LM.first->next;
65          p->next = nullptr;
66      }
67  }
68
69  void deleteLastMovie(listMovie &LM, elementMovie* &p) {
70      if (LM.first == nullptr) {
71          p = nullptr;
72      } else if (LM.first->next == nullptr) {
73          p = LM.first;
74          LM.first = nullptr;
75      } else {
76          elementMovie* current = LM.first;
77          while (current->next->next != nullptr) {
78              current = current->next;
79          }
80          p = current->next;
81          current->next = nullptr;
82      }
83  }
84
85  void deleteAfterMovie(elementMovie* prec, elementMovie* &p) {
86      if (prec != nullptr && prec->next != nullptr) {
87          p = prec->next;
88          prec->next = p->next;
89          p->next = nullptr;
90      } else {
91          p = nullptr;
92      }
93  }
```

```
94
95  void deleteElementMovie(listMovie &LM, elementMovie* &p, string method, elementMovie* prec) {
96      if (method == "first") {
97          deleteFirstMovie(LM, p);
98      } else if (method == "last") {
99          deleteLastMovie(LM, p);
100     } else if (method == "after" && prec != nullptr) {
101         deleteAfterMovie(prec, p);
102     }
103 }
```

Penjelasan :

Algoritmanya sama seperti deleteCustomer, cuma beda pakai list movie saja.

e.	Delete Element Child **(5)** Delete Rental PIC : Josua

```
58  void deleteFirstRental(listRental &LR, elementRental* &p) {
59      if (LR.first == nullptr) {
60          p = nullptr;
61      } else {
62          p = LR.first;
63          LR.first = LR.first->next;
64          p->next = nullptr;
65      }
66  }
67
68  void deleteLastRental(listRental &LR, elementRental* &p) {
69      if (LR.first == nullptr) {
70          p = nullptr;
71      } else if (LR.first->next == nullptr) {
72          p = LR.first;
73          LR.first = nullptr;
74      } else {
75          elementRental* current = LR.first;
76          while (current->next->next != nullptr) {
77              current = current->next;
78          }
79          p = current->next;
80          current->next = nullptr;
81      }
82  }
83
84  void deleteAfterRental(elementRental* prec, elementRental* &p) {
85      if (prec != nullptr && prec->next != nullptr) {
86          p = prec->next;
87          prec->next = p->next;
88          p->next = nullptr;
89      } else {
90          p = nullptr;
91      }
92  }
```

```
93
94  void deleteElementRental(listRental &LR, elementRental* &p, string method, elementRental* prec) {
95      if (method == "first") {
96          deleteFirstRental(LR, p);
97      } else if (method == "last") {
98          deleteLastRental(LR, p);
99      } else if (method == "after" && prec != nullptr) {
100         deleteAfterRental(prec, p);
101     }
102 }
103
```

Penjelasan :

Algoritmanya sama seperti deleteCustomer, cuma beda pakai list rental saja.

f.	Delete Element Relation (5) Delete Customer – Movie & Delete Movie – Rental PIC : Semua Anggota

```cpp
void deleteFirstRelCustMovie(elementCustomer* customer, relCustMovie* &p) {
    if (customer->firstRel == nullptr) {
        p = nullptr;
    } else {
        p = customer->firstRel;
        customer->firstRel = customer->firstRel->nextRel;
        p->nextRel = nullptr;
    }
}

void deleteLastRelCustMovie(elementCustomer* customer, relCustMovie* &p) {
    if (customer->firstRel == nullptr) {
        p = nullptr;
    } else if (customer->firstRel->nextRel == nullptr) {
        p = customer->firstRel;
        customer->firstRel = nullptr;
    } else {
        relCustMovie* current = customer->firstRel;
        while (current->nextRel->nextRel != nullptr) {
            current = current->nextRel;
        }
        p = current->nextRel;
        current->nextRel = nullptr;
    }
}

void deleteAfterRelCustMovie(relCustMovie* prec, relCustMovie* &p) {
    if (prec != nullptr && prec->nextRel != nullptr) {
        p = prec->nextRel;
        prec->nextRel = p->nextRel;
        p->nextRel = nullptr;
    } else {
        p = nullptr;
    }
}

void deleteElementRelCustMovie(elementCustomer* customer, relCustMovie* &p, string method, relCustMovie* prec) {
    if (method == "first") {
        deleteFirstRelCustMovie(customer, p);
    } else if (method == "last") {
        deleteLastRelCustMovie(customer, p);
    } else if (method == "after" && prec != nullptr) {
        deleteAfterRelCustMovie(prec, p);
    }
}
```

```cpp
void deleteFirstRelMovieRent(elementMovie* movie, relMovieRent* &p) {
    if (movie->firstRel == nullptr) {
        p = nullptr;
    } else {
        p = movie->firstRel;
        movie->firstRel = movie->firstRel->nextRel;
        p->nextRel = nullptr;
    }
}

void deleteLastRelMovieRent(elementMovie* movie, relMovieRent* &p) {
    if (movie->firstRel == nullptr) {
        p = nullptr;
    } else if (movie->firstRel->nextRel == nullptr) {
        p = movie->firstRel;
        movie->firstRel = nullptr;
    } else {
        relMovieRent* current = movie->firstRel;
        while (current->nextRel->nextRel != nullptr) {
            current = current->nextRel;
        }
        p = current->nextRel;
        current->nextRel = nullptr;
    }
}

void deleteAfterRelMovieRent(relMovieRent* prec, relMovieRent* &p) {
    if (prec != nullptr && prec->nextRel != nullptr) {
        p = prec->nextRel;
        prec->nextRel = p->nextRel;
        p->nextRel = nullptr;
    } else {
        p = nullptr;
    }
}

void deleteElementRelMovieRent(elementMovie* movie, relMovieRent* &p, string method, relMovieRent* prec) {
    if (method == "first") {
        deleteFirstRelMovieRent(movie, p);
    } else if (method == "last") {
        deleteLastRelMovieRent(movie, p);
    } else if (method == "after" && prec != nullptr) {
        deleteAfterRelMovieRent(prec, p);
    }
}
```

Penjelasan :

Prosedur delete single linked list standard algoritma prosedur delete relasi algoritmanya sama, dan algoritmanya juga sama seperti prosedur delete di list sebelumnya, dan juga memakai deleteElementRelCusMovie & deleteElemenRelMovieRent untuk membungkus prosedur delete lain agar lebih mudah dipanggil di menu main.

g.	Find Element Parent (5) Find Customer PIC : Virza

```
105
106 elementCustomer* findCustomerByID(listCustomer LC, string id) {
107     elementCustomer* current = LC.first;
108     while (current != nullptr) {
109         if (current->info.id == id) {
110             return current;
111         }
112         current = current->next;
113     }
114     return nullptr;
115 }
116
117 elementCustomer* findElementCustomer(listCustomer LC, string id) {
118     return findCustomerByID(LC, id);
119 }
120
```

Penjelasan :

Traversal linear dari head list customer. Untuk setiap customer: bandingkan info.id dengan ID target. Jika sama, return pointer ke customer tersebut. Jika tidak sama, pindah ke customer berikutnya. Return null jika mencapai akhir list tanpa ketemu.

g.	Find Element Parent (5) Find Movie PIC : Kayana

```
106
107 elementMovie* findMovieByCode(listMovie LM, string code) {
108     elementMovie* current = LM.first;
109     while (current != nullptr) {
110         if (current->info.code == code) {
111             return current;
112         }
113         current = current->next;
114     }
115     return nullptr;
116 }
117
118 elementMovie* findElementMovie(listMovie LM, string code) {
119     return findMovieByCode(LM, code);
120 }
```

Penjelasan :

Traversal linear dari head list movie. Untuk setiap movie: bandingkan info.code dengan kode target. Jika match, return pointer ke movie tersebut. Jika tidak match, pindah ke movie berikutnya. Return null jika mencapai akhir list tanpa ketemu.

h.	Find Element Child (5) Find Rental PIC : Josua

```
105
106 elementRental* findRentalByID(listRental LR, string id) {
107     elementRental* current = LR.first;
108     while (current != nullptr) {
109         if (current->info.rentId == id) {
110             return current;
111         }
112         current = current->next;
113     }
114     return nullptr;
115 }
116
117 elementRental* findElementRental(listRental LR, string id) {
118     return findRentalByID(LR, id);
119 }
120
```

Penjelasan :

Traversal linear dari head list rental. Untuk setiap rental: bandingkan info.rentId dengan ID target. Jika cocok, return pointer ke rental. Jika tidak cocok, lanjut ke rental berikutnya. Return null jika mencapai akhir list tanpa ketemu.

i.	Find Element Relation (5) Find Customer – Movie & Movie Rental PIC : Semua Anggota

```
78  relCustMovie* findRelCustMovieByMovie(elementCustomer* customer, string movieCode) {
79      if (customer->firstRel == nullptr) {
80          return nullptr;
81      }
82
83      relCustMovie* current = customer->firstRel;
84      while (current != nullptr) {
85          if (current->movie->info.code == movieCode) {
86              return current;
87          }
88          current = current->nextRel;
89      }
90      return nullptr;
91  }
```

```
291 relMovieRent* findRelMovieRentByRental(elementMovie* movie, string rentalId) {
292     if (movie->firstRel == nullptr) {
293         return nullptr;
294     }
295
296     relMovieRent* current = movie->firstRel;
297     while (current != nullptr) {
298         if (current->rental->info.rentId == rentalId) {
299             return current;
300         }
301         current = current->nextRel;
302     }
303     return nullptr;
304 }
```

Penjelasan :

Fungsi mencari Node di List relasi berdasarkan code movie seccara traversal dari head firstRel hingga ujung. Jika code movie sesuai maka mereturn alamat node relasinya, jika tidak return null.

Sedangkan untuk fungsi findRelMovieRentByRental, dilakukan traversal linear dimulai dari head list relasi yang dimiliki oleh suatu movie. Untuk setiap relasi dalam list tersebut: bandingkan nilai rental->info.rentId pada relasi saat ini dengan ID rental target yang dicari. Jika terjadi kecocokan, fungsi langsung mengembalikan pointer ke relasi tersebut. Jika tidak cocok, proses dilanjutkan dengan berpindah ke relasi berikutnya dalam list. Fungsi akan mengembalikan nilai null jika traversal telah mencapai akhir list relasi tanpa menemukan rental dengan ID yang sesuai.

j.	Show All Data Di List Parent (5) Show All Customer PIC : Virza

```
122
123  void showAllCustomer(listCustomer LC) {
124      elementCustomer* current = LC.first;
125      int count = 1;
126
127      cout << "=== ALL CUSTOMERS ===" << endl;
128      if (current == nullptr) {
129          cout << "No customers found." << endl;
130          return;
131      }
132
133      while (current != nullptr) {
134          cout << count << ". Customer ID: " << current->info.id << endl;
135          cout << "   Name: " << current->info.name << endl;
136          cout << "   Phone: " << current->info.phone << endl;
137          cout << "   Movies rented: " << countCustomerMovies(current) << endl;
138          cout << "--------------------------------" << endl;
139          current = current->next;
140          count++;
141      }
142  }
143
144  void showCustomerData(elementCustomer* cust) {
145      if (cust != nullptr) {
146          cout << "Customer ID: " << cust->info.id << endl;
147          cout << "Name: " << cust->info.name << endl;
148          cout << "Phone: " << cust->info.phone << endl;
149      }
150  }
```

Penjelasan :

Traversal linear dari head hingga ujung list customer. Untuk setiap customer: tampilkan data (ID, nama, telepon), lalu traversal relasi melalui firstRel untuk ambil dan tampilkan semua judul film yang disewa. Selesai saat pointer mencapai null.

j.	Show All Data Di List Parent (5) Show All Movie PIC : Kayana

```
124  void showAllMovie(listMovie LM) {
125      elementMovie* current = LM.first;
126      int count = 1;
127
128      cout << "=== ALL MOVIES ===" << endl;
129      if (current == nullptr) {
130          cout << "No movies found." << endl;
131          return;
132      }
133
134      while (current != nullptr) {
135          cout << count << ". Movie Code: " << current->info.code << endl;
136          cout << "   Title: " << current->info.title << endl;
137          cout << "   Genre: " << current->info.genre << endl;
138          cout << "   Year: " << current->info.year << endl;
139          cout << "   Price: $" << current->info.price << endl;
140          cout << "   Active rentals: " << countMovieRentals(current) << endl;
141          cout << "--------------------------------" << endl;
142          current = current->next;
143          count++;
144      }
145  }
```

Penjelasan :

Traversal linear list movie. Untuk setiap film: tampilkan detail (kode, judul, genre, tahun, harga), dan count rental aktif pakai prosedur countMovieRentals. Selesai saat pointer null.

k.	Show All Data Di List Child (5) Show All Rental PIC : Josua

```cpp

void showAllRental(listRental LR) {
    elementRental* current = LR.first;
    int count = 1;

    cout << "=== ALL RENTALS ===" << endl;
    if (current == nullptr) {
        cout << "No rentals found." << endl;
        return;
    }

    while (current != nullptr) {
        cout << count << ". Rental ID: " << current->info.rentId << endl;
        cout << "   Date: " << current->info.date << endl;
        cout << "   Duration: " << current->info.duration << " days" << endl;
        cout << "   Total Fee: $" << current->info.totalFee << endl;
        cout << "--------------------------------" << endl;
        current = current->next;
        count++;
    }
}

void showRentalData(elementRental* rent) {
    if (rent != nullptr) {
        cout << "Rental ID: " << rent->info.rentId << endl;
        cout << "Date: " << rent->info.date << endl;
        cout << "Movie: " << rent->info.movieCode << endl;
        cout << "Duration: " << rent->info.duration << " days" << endl;
        cout << "Total Fee: $" << rent->info.totalFee << endl;
    }
}
```

Penjelasan :

Traversal linear list rental. Untuk setiap rental: tampilkan data transaksi (ID, tanggal, durasi, biaya). Tidak ada traversal relasi karena rental adalah child terbawah. Selesai saat pointer null.

1.	Show Data Child Dari Parent Tertentu (5) Show Movie For Specific Customer & Show Rentals for Specific Movie PIC : Semua Anggota

```
112 void showMoviesForCustomer(listCustomer LC, string customerId) {
113     elementCustomer* cust = findCustomerByID(LC, customerId);
114
115     if (cust == nullptr) {
116         cout << "Customer with ID '" << customerId << "' not found!" << endl;
117         return;
118     }
119
120     cout << "Movies rented by " << cust->info.name << " (" << customerId << "):" << endl;
121
122     if (cust->firstRel == nullptr) {
123         cout << "No movies rented." << endl;
124         return;
125     }
126
127     relCustMovie* rel = cust->firstRel;
128     int count = 1;
129
130     while (rel != nullptr) {
131         cout << count << ". " << rel->movie->info.code << " - "
132              << rel->movie->info.title << endl;
133         rel = rel->nextRel;
134         count++;
135     }
136 }
```

Penjelasan :

Untuk prosedur showMoviesForCustomer, dilakukan pencarian linear customer berdasarkan ID yang diberikan. Jika customer tidak ditemukan dalam list, fungsi langsung mengembalikan pesan error. Jika customer ditemukan, dilakukan traversal linear dari head list relasi yang dimiliki customer tersebut. Untuk setiap relasi dalam list: tampilkan informasi movie berupa kode dan judul film yang direferensikan oleh relasi saat ini. Proses dilanjutkan dengan berpindah ke relasi berikutnya hingga mencapai akhir list relasi. Jika customer tidak memiliki relasi sama sekali, fungsi akan menampilkan pesan bahwa tidak ada movie yang disewa.

Intinya menampilkan Data Movie dari Customer tertentu yang dicari berdasarkan IdCustomernya.

m.	Show Setiap Data Parent Beserta Data Child Yang Berelasi Dengannya (5) Show All Customers with Movies PIC : Semua Anggota

```
138 void showAllCustomersWithMovies(listCustomer LC) {
139     elementCustomer* current = LC.first;
140     int customerCount = 1;
141 
142     cout << "\n=== ALL CUSTOMERS WITH THEIR RENTED MOVIES ===" << endl;
143 
144     if (current == nullptr) {
145         cout << "No customers found in the system." << endl;
146         return;
147     }
148 
149     while (current != nullptr) {
150         cout << "\n[" << customerCount << "] CUSTOMER:" << endl;
151         cout << "ID: " << current->info.id << endl;
152         cout << "Name: " << current->info.name << endl;
153         cout << "Phone: " << current->info.phone << endl;
154 
155         if (current->firstRel != nullptr) {
156             cout << "RENTED MOVIES DETAILS:" << endl;
157             relCustMovie* rel = current->firstRel;
158             int movieCount = 0;
159             while (rel != nullptr) {
160                 movieCount++;
161                 cout << "   " << movieCount << ". " << rel->movie->info.code
162                      << " - " << rel->movie->info.title << endl;
163                 cout << "      Genre: " << rel->movie->info.genre << endl;
164                 cout << "      Year: " << rel->movie->info.year << endl;
165                 cout << "      Price: $" << rel->movie->info.price << endl;
166                 if (rel->nextRel != nullptr) cout << endl;
167                 rel = rel->nextRel;
168             }
169             cout << "Total movies rented: " << movieCount << endl;
170         } else {
171             cout << "Rented Movies: None" << endl;
172         }
173 
174         cout << "---" << endl;
175         current = current->next;
176         customerCount++;
177     }
```

Penjelasan :

Mencetak Semua data customer dan movie yang berelasi, dengan urutan:

Customer1 – movie n1- movie n2 – hingga list relasi habis,

Customer 2 – movie n1- movie n2 – hingga list relasi habis.

Customer 3...

Untuk prosedur showAllCustomersWithMovies, dilakukan traversal linear dari head list customer. Untuk setiap customer: tampilkan data customer, lalu traversal linear di list relasinya. Untuk setiap relasi: tampilkan detail movie yang direferensikan. Ulangi hingga semua customer diproses. Jika customer tidak ada relasi, tampilkan "None".

p.	Count Relation Dari Setiap Element Parent (5) Relation Count per Customer & Movie PIC : Semua Anggota

```
154 int countCustomerMovies(elementCustomer* customer) {
155     int count = 0;
156     if (customer != nullptr) {
157         relCustMovie* current = customer->firstRel;
158         while (current != nullptr) {
159             count++;
160             current = current->nextRel;
161         }
162     }
163     return count;
164 }
```

```
159 int countMovieRentals(elementMovie* movie) {
160     int count = 0;
161     if (movie != nullptr) {
162         relMovieRent* current = movie->firstRel;
163         while (current != nullptr) {
164             count++;
165             current = current->nextRel;
166         }
167     }
168     return count;
169 }
```

Penjelasan :

Untuk fungsi countCustomerMovies, dilakukan traversal linear dari head list relasi yang dimiliki customer. Untuk setiap relasi: increment counter sebanyak satu, lalu pindah ke relasi berikutnya. Ulangi hingga mencapai akhir list relasi. Return total jumlah relasi yang dihitung.

Untuk fungsi countMovieRentals, dilakukan traversal linear dari head list relasi yang dimiliki movie. Untuk setiap relasi: increment counter sebanyak satu, lalu pindah ke relasi berikutnya. Ulangi hingga mencapai akhir list relasi. Return total jumlah relasi yang dihitung.

q.	Count relation yang dimiliki oleh child tertentu (5) Count Movies For Rental PIC : Semua Anggota

```
184  int countMoviesForRental(listMovie LM, string rentalId) {
185      int count = 0;
186      elementMovie* current = LM.first;
187
188      while (current != nullptr) {
189          relMovieRent* rel = current->firstRel;
190          while (rel != nullptr) {
191              if (rel->rental->info.rentId == rentalId) {
192                  count++;
193                  break;
194              }
195              rel = rel->nextRel;
196          }
197          current = current->next;
198      }
199      return count;
200  }
```

Penjelasan :

Untuk fungsi countMoviesForRental, dilakukan traversal linear dari head list movie. Untuk setiap movie: traversal linear di list relasi movie tersebut. Untuk setiap relasi: bandingkan rental ID dengan target. Jika cocok, increment counter dan lanjut ke movie berikutnya. Ulangi hingga semua movie diproses. Return total jumlah movie yang terhubung dengan rental tersebut.

r.	Count element child yang tidak memiliki relasi (5) Count Rentals without Movie Relation PIC : Semua Anggota

```
156  bool hasMovieRelation(elementRental* rental, listMovie LM) {
157      elementMovie* movieCurrent = LM.first;
158
159      while (movieCurrent != nullptr) {
160          relMovieRent* rel = movieCurrent->firstRel;
161          while (rel != nullptr) {
162              if (rel->rental == rental) {
163                  return true;
164              }
165              rel = rel->nextRel;
166          }
167          movieCurrent = movieCurrent->next;
168      }
169      return false;
170  }
171
172  int countRentalsWithNoMovie(listRental LR, listMovie LM) {
173      int count = 0;
174      elementRental* current = LR.first;
175
176      while (current != nullptr) {
177          if (!hasMovieRelation(current, LM)) {
178              count++;
179          }
180          current = current->next;
181      }
182      return count;
183  }
```

Penjelasan :

Untuk fungsi hasMovieRelation, dilakukan traversal linear dari head list movie. Untuk setiap movie: traversal linear di list relasinya. Untuk setiap relasi: bandingkan pointer rental dengan target. Jika cocok, return true. Jika tidak, lanjut hingga semua movie diproses. Return false jika tidak ditemukan.

Untuk fungsi countRentalsWithNoMovie, dilakukan traversal linear dari head list rental. Untuk setiap rental: panggil hasMovieRelation. Jika tidak memiliki relasi, increment counter. Ulangi hingga semua rental diproses. Return total rental tanpa relasi movie.

n.	Show data child beserta data parent yang masing-masing child miliki (10) Show All Rentals with Movies PIC : Semua Anggota

```
371 void showAllRentalsWithMovies(listRental LR, listMovie LM) {
372     cout << "=== ALL RENTALS WITH THEIR MOVIES ===" << endl;
373 
374     elementRental* rentCurrent = LR.first;
375     int count = 1;
376 
377     if (!rentCurrent) {
378         cout << "No rentals found." << endl;
379         return;
380     }
381 
382     while (rentCurrent) {
383         cout << "\n[" << count << "] RENTAL:" << endl;
384         cout << "ID: " << rentCurrent->info.rentId << endl;
385         cout << "Date: " << rentCurrent->info.date << endl;
386         cout << "Duration: " << rentCurrent->info.duration << " days" << endl;
387         cout << "Fee: $" << rentCurrent->info.totalFee << endl;
388 
389         cout << "Connected Movies: ";
390         elementMovie* movieCurrent = LM.first;
391         bool hasMovies = false;
392         int movieCount = 0;
393 
394         while (movieCurrent) {
395             relMovieRent* rel = movieCurrent->firstRel;
396             while (rel) {
397                 if (rel->rental->info.rentId == rentCurrent->info.rentId) {
398                     if (movieCount > 0) cout << ", ";
399                     cout << movieCurrent->info.title << " (" << movieCurrent->info.code << ")";
400                     hasMovies = true;
401                     movieCount++;
402                     break;
403                 }
404                 rel = rel->nextRel;
405             }
406             movieCurrent = movieCurrent->next;
407         }
408 
409         if (!hasMovies) {
410             cout << "None";
411         }
412         cout << endl;
413 
414         rentCurrent = rentCurrent->next;
415         count++;
416     }
417 }
```

Penjelasan : Mencetak List Rental dan mengecek dari list movie jika ada node di list relasi dari list movie yang cocok id rentalnya didalam. Jika ada maka cetak data movienya (parent). jika tidak cetak “None”.

o.	Show data parent yang berelasi dengan child tertentu (5) Show Movies for Specific Rental PIC : Semua Anggota

```cpp
void showCustomersForMovie(listCustomer LC, listMovie LM, string movieCode) {
    // Cari movie dulu
    elementMovie* movie = findMovieByCode(LM, movieCode);
    if (!movie) {
        cout << "Movie with code '" << movieCode << "' not found!" << endl;
        return;
    }

    cout << "=== CUSTOMERS RENTING: " << movie->info.title << " (" << movieCode << ") ===" << endl;
    cout << "Genre: " << movie->info.genre << " | Year: " << movie->info.year << " | Price: $" << movie->info.price << endl;
    cout << "========================================\n";

    elementCustomer* current = LC.first;
    int count = 0;

    while (current != nullptr) {
        relCustMovie* rel = current->firstRel;
        while (rel != nullptr) {
            if (rel->movie->info.code == movieCode) {
                count++;
                cout << "\n[" << count << "] CUSTOMER:" << endl;
                cout << "ID: " << current->info.id << endl;
                cout << "Name: " << current->info.name << endl;
                cout << "Phone: " << current->info.phone << endl;
                cout << "Movies rented by this customer: " << countCustomerMovies(current) << endl;
                break;
            }
            rel = rel->nextRel;
        }
        current = current->next;
    }

    if (count == 0) {
        cout << "\nNo customers are currently renting this movie." << endl;
    } else {
        cout << "\n========================================" << endl;
        cout << "TOTAL CUSTOMERS RENTING THIS MOVIE: " << count << endl;
    }
}
```


```cpp
void showCustomersForMovie(listCustomer LC, listMovie LM, string movieCode) {
    // Cari movie dulu
    elementMovie* movie = findMovieByCode(LM, movieCode);
    if (!movie) {
        cout << "Movie with code '" << movieCode << "' not found!" << endl;
        return;
    }

    cout << "=== CUSTOMERS RENTING: " << movie->info.title << " (" << movieCode << ") ===" << endl;
    cout << "Genre: " << movie->info.genre << " | Year: " << movie->info.year << " | Price: $" << movie->info.price << endl;
    cout << "========================================\n";

    elementCustomer* current = LC.first;
    int count = 0;

    while (current != nullptr) {
        relCustMovie* rel = current->firstRel;
        while (rel != nullptr) {
            if (rel->movie->info.code == movieCode) {
                count++;
                cout << "\n[" << count << "] CUSTOMER:" << endl;
                cout << "ID: " << current->info.id << endl;
                cout << "Name: " << current->info.name << endl;
                cout << "Phone: " << current->info.phone << endl;
                cout << "Movies rented by this customer: " << countCustomerMovies(current) << endl;
                break;
            }
            rel = rel->nextRel;
        }
        current = current->next;
    }

    if (count == 0) {
        cout << "\nNo customers are currently renting this movie." << endl;
    } else {
        cout << "\n========================================" << endl;
        cout << "TOTAL CUSTOMERS RENTING THIS MOVIE: " << count << endl;
    }
}
```

Penjelasan : Fungsi showCustomersForMovie bertujuan untuk menampilkan semua data customer yang memiliki relasi dengan movie tertentu. Proses dimulai dengan pencarian movie berdasarkan kode yang diberikan. Jika movie ditemukan, fungsi kemudian melakukan traversal linear melalui seluruh list customer. Untuk setiap customer yang diproses, dilakukan pengecekan terhadap semua relasi customer-movie yang dimilikinya dengan melakukan traversal pada list relasi customer tersebut. Apabila ditemukan relasi yang mengarah ke movie target, informasi lengkap customer termasuk ID, nama, nomor telepon, dan jumlah total movie yang disewa akan ditampilkan. Fungsi ini memberikan kemampuan untuk melacak dari child (movie) ke parent (customer), menunjukkan semua customer yang sedang menyewa movie tertentu beserta statistik terkait. Proses berlanjut hingga semua customer dalam sistem telah diperiksa, dengan hasil akhir yang menunjukkan total jumlah customer yang terkait dengan movie tersebut.

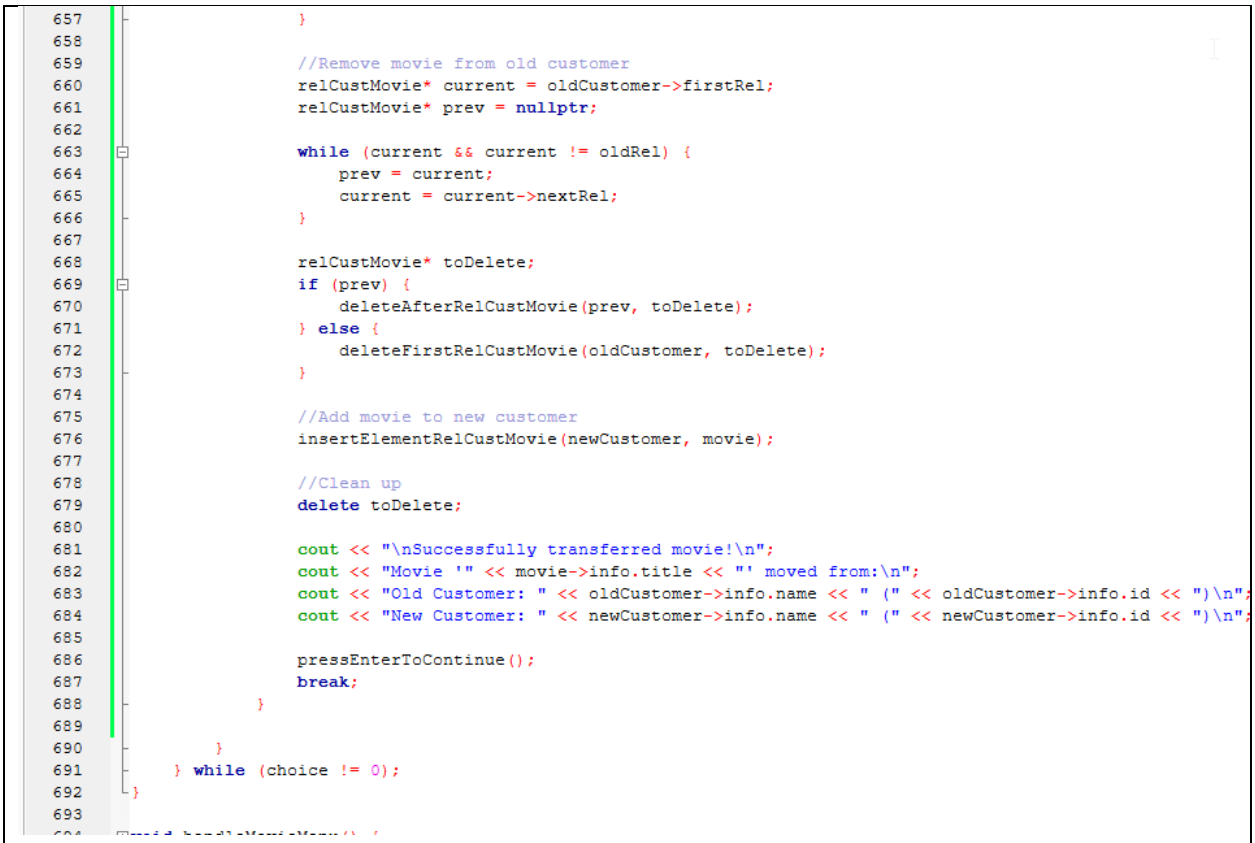
s.	Edit relasi /mengganti child dari parent tertentu (5) PIC : Josua

main.cpp | data.h | customer.h | movie.h | rental.h | relation.h | customer.cpp | movie.cpp | rental.cpp | relation.cpp

```cpp
            case 10: {
                clearScreen();
                cout << "=== CUSTOMER CHANGE MOVIE (Change Child)===\n";
                string customerId, oldMovieCode, newMovieCode;
                cout << "Customer ID: ";
                getline(cin, customerId);
                cout << "Current Movie Code (to replace): ";
                getline(cin, oldMovieCode);
                cout << "New Movie Code: ";
                getline(cin, newMovieCode);

                // Find the customer
                elementCustomer* cust = findCustomerByID(LC, customerId);
                if (!cust) {
                    cout << "Customer not found!\n";
                    pressEnterToContinue();
                    break;
                }

                // Find the new movie
                elementMovie* newMovie = findMovieByCode(LM, newMovieCode);
                if (!newMovie) {
                    cout << "New movie not found!\n";
                    pressEnterToContinue();
                    break;
                }

                // Check if customer already has the new movie
                if (findRelCustMovieByMovie(cust, newMovieCode)) {
                    cout << "Customer already has movie " << newMovieCode << "!\n";
                    pressEnterToContinue();
                    break;
```

```
560                     break;
561                 }
562
563                 // Check if customer has the old movie
564                 relCustMovie* oldRel = findRelCustMovieByMovie(cust, oldMovieCode);
565                 if (!oldRel) {
566                     cout << "Customer doesn't have movie " << oldMovieCode << "!\n";
567                     pressEnterToContinue();
568                     break;
569                 }
570
571                 // Find the old movie
572                 elementMovie* oldMovie = findMovieByCode(LM, oldMovieCode);
573                 if (!oldMovie) {
574                     cout << "Old movie not found in system!\n";
575                     pressEnterToContinue();
576                     break;
577                 }
578
579                 // Remove old movie from customer
580                 relCustMovie* current = cust->firstRel;
581                 relCustMovie* prev = nullptr;
582
583                 while (current && current != oldRel) {
584                     prev = current;
585                     current = current->nextRel;
586                 }
587
588                 relCustMovie* toDelete;
589                 if (prev) {
590                     deleteAfterRelCustMovie(prev, toDelete);
591                 } else {
592                     deleteFirstRelCustMovie(cust, toDelete);
593                 }
594
595                 // Add new movie to customer
596                 insertElementRelCustMovie(cust, newMovie);
597
598                 // Clean up
599                 delete toDelete;
600
601                 cout << "\n=== MOVIE CHANGED SUCCESSFULLY ===\n";
602                 cout << "Customer: " << cust->info.name << " (" << customerId << ")\n";
603                 cout << "Changed from: " << oldMovie->info.title << " (" << oldMovieCode << ")\n";
604                 cout << "Changed to: " << newMovie->info.title << " (" << newMovieCode << ")\n";
605
606                 pressEnterToContinue();
607                 break;
608             }
```

```
609             case 11: {
610                 clearScreen();
611                 cout << "=== CHANGE CUSTOMER FOR MOVIE (Change Parent) ===\n";
612                 string movieCode, newCustomerId;
613                 cout << "Movie Code: ";
614                 getline(cin, movieCode);
615                 cout << "New Customer ID: ";
616                 getline(cin, newCustomerId);
617 
618                 //Find the movie
619                 elementMovie* movie = findMovieByCode(LM, movieCode);
620                 if (!movie) {
621                     cout << "Movie not found!\n";
622                     pressEnterToContinue();
623                     break;
624                 }
625 
626                 //Find the new customer
627                 elementCustomer* newCustomer = findCustomerByID(LC, newCustomerId);
628                 if (!newCustomer) {
629                     cout << "New customer not found!\n";
630                     pressEnterToContinue();
631                     break;
632                 }
633 
634                 //Check if new customer already has this movie
635                 if (findRelCustMovieByMovie(newCustomer, movieCode)) {
636                     cout << "New customer already has this movie!\n";
637                     pressEnterToContinue();
638                     break;
639                 }
640 
641                 //Find the old customer (current parent)
642                 elementCustomer* oldCustomer = findCustomerByMovieCode(LC, movieCode);
643 
644                 if (!oldCustomer) {
645                     cout << "Movie is not currently rented by any customer!\n";
646                     pressEnterToContinue();
647                     break;
648                 }
649 
650                 //Find the specific relation in old customer
651                 relCustMovie* oldRel = findRelCustMovieByMovie(oldCustomer, movieCode);
652 
653                 if (!oldRel) {
654                     cout << "Error: Relation not found in old customer!\n";
655                     pressEnterToContinue();
656                     break;
```



```
657                 }
658
659                 //Remove movie from old customer
660                 relCustMovie* current = oldCustomer->firstRel;
661                 relCustMovie* prev = nullptr;
662
663                 while (current && current != oldRel) {
664                     prev = current;
665                     current = current->nextRel;
666                 }
667
668                 relCustMovie* toDelete;
669                 if (prev) {
670                     deleteAfterRelCustMovie(prev, toDelete);
671                 } else {
672                     deleteFirstRelCustMovie(oldCustomer, toDelete);
673                 }
674
675                 //Add movie to new customer
676                 insertElementRelCustMovie(newCustomer, movie);
677
678                 //Clean up
679                 delete toDelete;
680
681                 cout << "\nSuccessfully transferred movie!\n";
682                 cout << "Movie '" << movie->info.title << "' moved from:\n";
683                 cout << "Old Customer: " << oldCustomer->info.name << " (" << oldCustomer->info.id << ")\n";
684                 cout << "New Customer: " << newCustomer->info.name << " (" << newCustomer->info.id << ")\n";
685
686                 pressEnterToContinue();
687                 break;
688             }
689
690         }
691     } while (choice != 0);
692 }
693
```

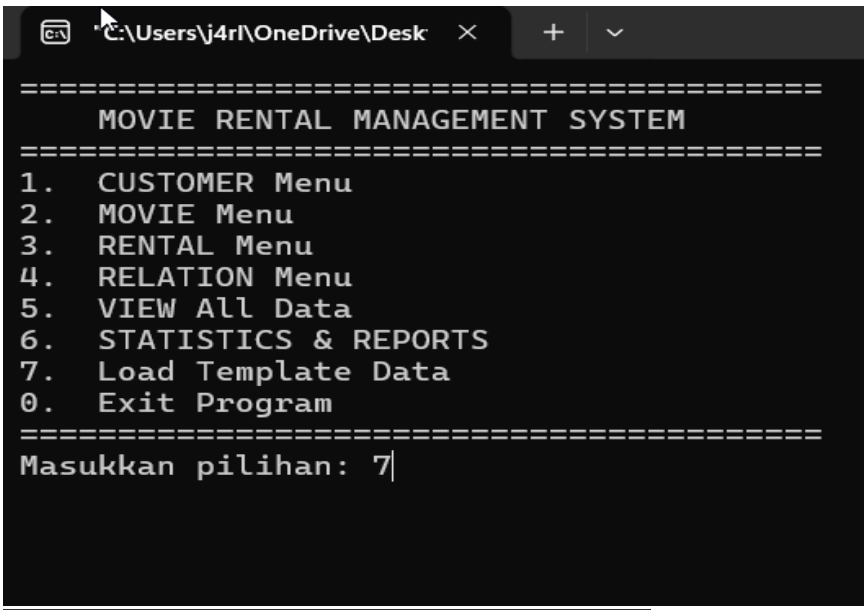
Penjelasan :

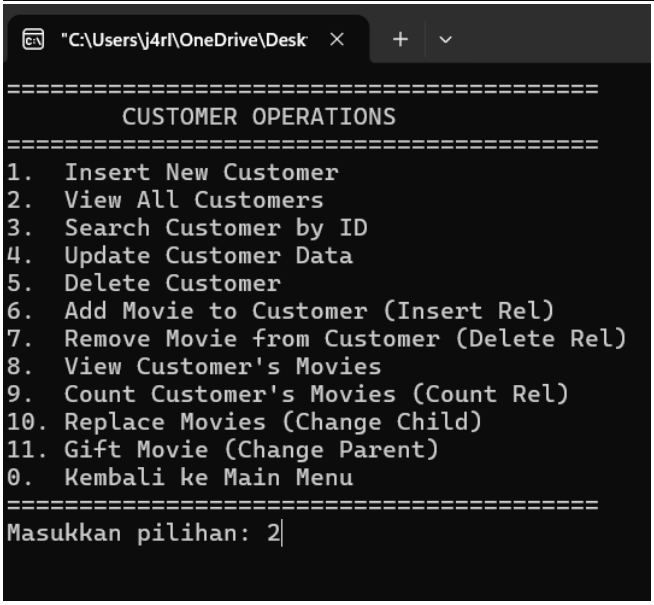
Pada implementasi di handleCustomerMenu() case 10, proses dimulai dengan validasi keberadaan customer, movie lama, dan movie baru. Dilakukan traversal linear pada list relasi customer untuk menemukan relasi dengan movie lama. Apabila ditemukan, relasi tersebut dipotong dari linked list relasi customer dengan mengatur pointer nextRel dari node sebelumnya atau menggunakan fungsi delete yang sesuai jika relasi berada di posisi pertama. Setelah relasi lama dihapus, dibuat relasi baru dengan movie target menggunakan fungsi insertElementRelCustMovie. Proses ini memungkinkan customer mengganti satu movie dengan movie lain dalam daftar penyewaannya tanpa mempengaruhi data customer atau movie itu sendiri. **Disini yang diubah childnya**

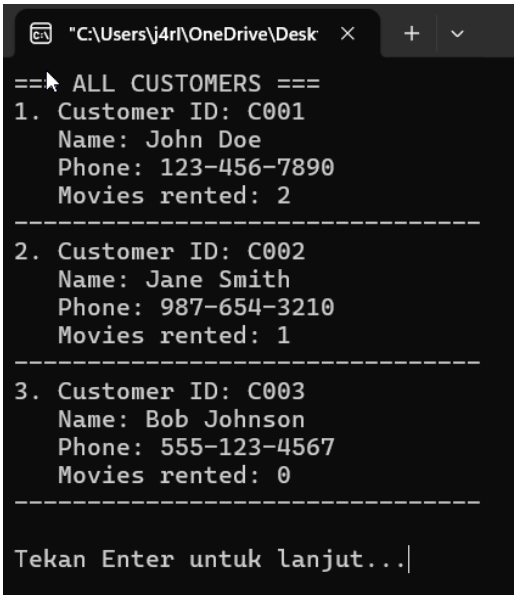
Pada implementasi di handleCustomerMenu() case 11, proses diawali dengan pencarian movie dan customer baru. Dilakukan pencarian customer lama yang memiliki relasi dengan movie target melalui traversal linear seluruh list customer. Setelah customer lama ditemukan,

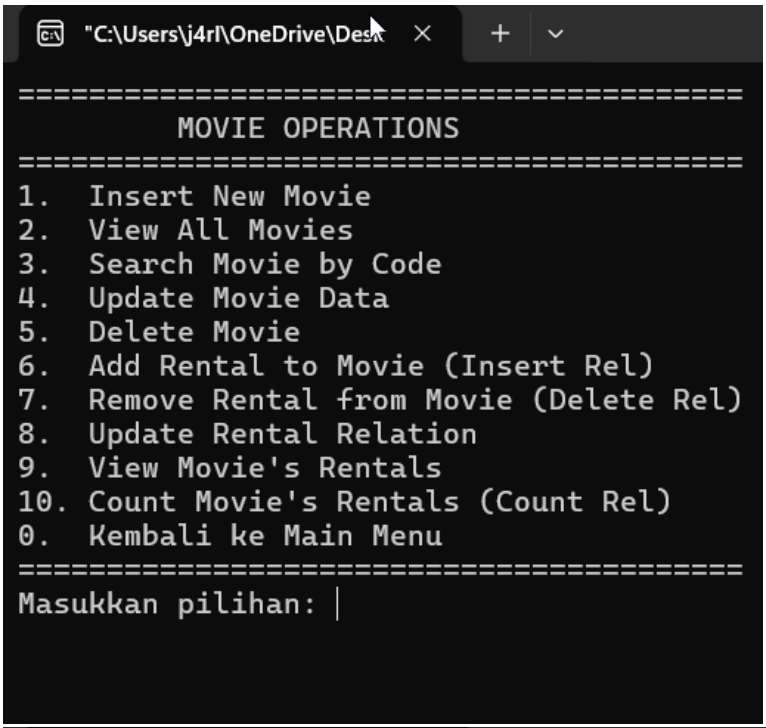
dilakukan traversal pada list relasi customer tersebut untuk menemukan relasi spesifik dengan movie target. Relasi ini kemudian dipotong dari linked list relasi customer lama dan ditambahkan ke linked list relasi customer baru. Implementasi ini memungkinkan transfer kepemilikan movie dari satu customer ke customer lain, mengubah hubungan parent tanpa mempengaruhi data movie itu sendiri. **Disini yang diubah parentnya.**

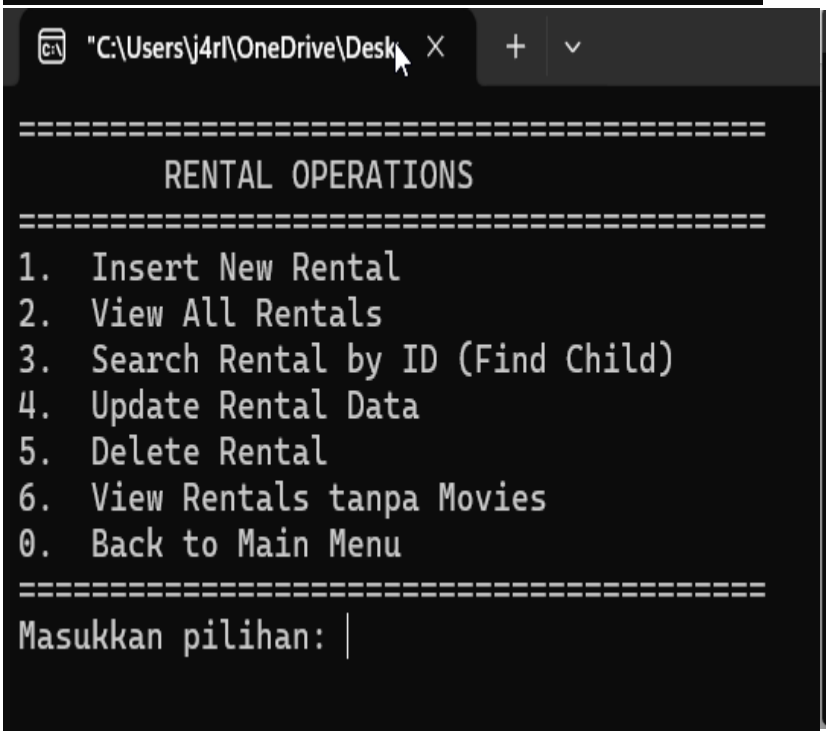
CONTOH OUTPUT:

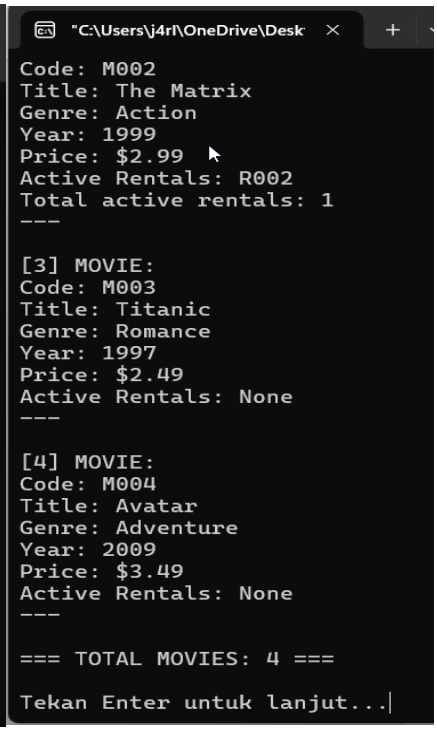







"C:\Users\j4rl\OneDrive\Desk
==========
MOVIE OPERATIONS
==========
1. Insert New Movie
2. View All Movies
3. Search Movie by Code
4. Update Movie Data
5. Delete Movie
6. Add Rental to Movie (Insert Rel)
7. Remove Rental from Movie (Delete Rel)
8. Update Rental Relation
9. View Movie's Rentals
10. Count Movie's Rentals (Count Rel)
0. Kembali ke Main Menu
==========
Masukkan pilihan:


"C:\Users\j4rl\OneDrive\Desk
==========
RENTAL OPERATIONS
==========
1. Insert New Rental
2. View All Rentals
3. Search Rental by ID (Find Child)
4. Update Rental Data
5. Delete Rental
6. View Rentals tanpa Movies
0. Back to Main Menu
==========
Masukkan pilihan:


"C:\Users\j4rl\OneDrive\Desk
Code: M002
Title: The Matrix
Genre: Action
Year: 1999
Price: $2.99
Active Rentals: R002
Total active rentals: 1
---
[3] MOVIE:
Code: M003
Title: Titanic
Genre: Romance
Year: 1997
Price: $2.49
Active Rentals: None
---
[4] MOVIE:
Code: M004
Title: Avatar
Genre: Adventure
Year: 2009
Price: $3.49
Active Rentals: None

=== TOTAL MOVIES: 4 ===
Tekan Enter untuk lanjut...

Persen kontribusi anggota dalam tim (total 100%) :

1. Josua Michael E. S (103092400006)
    - Menyelesaikan fungsi & prosedur create, insert, delete, find dan show untuk list rental.
        - b. insert element child (rental).
        - e. delete element child (rental).
        - k. show all data di list child (rental).
        - h. find element child (rental)
        - s. Edit relasi /mengganti child dari parent tertentu.
    - Membuat main.cpp dan prosedur display menu.
2. Kayana Hayu Candraningtyas (103092400017)
    - Menyelesaikan fungsi dan prosedur create, insert, delete, find dan show untuk list movie.
        - a. insert element parent (movie).
        - d. delete element parent (movie).
        - j. show all data di list parent (movie).
        - g. find element parent (movie)
3. M. Virza Akbar Hidayatulla (103092400013)
    - Menyelesaikan fungsi & prosedur create, insert, delete, find dan show untuk list customer.
        - a. insert element parent (customer).
        - d. delete element parent (customer).
        - j. show all data di list parent (customer).
        - g. find element parent (customer)
4. Kerjasama (Semua Anggota) :
    - Menyusun header adt mll semua list dan membuat list relasi.
    - Menambahkan prosedur show spesifik (l, m).
    - Membuat insert, delete, dan find elemen relasi (c,f,i).
    - Membuat Prosedur yang memenuhi requirement Fungsionalitas:
        - p. count relation dari setiap element parent.
        - q. count relation yang dimiliki oleh child tertentu.

- r. count elemen child yang tidak memiliki relasi.
- n. Show data child beserta data parent yang masing-masing child miliki.
- o. Show data parent yang berelasi dengan child tertentu.

Bukti responsi tugas besar bersama asdos, asprak, ataupun dosen :