

**LAPORAN PRAKTIKUM
STRUKTUR DATA**

**MODUL XIII
MULTI LINKED LIST**



Disusun Oleh :
NAMA : Azzahra Farelika Esti Ning Tyas
NIM : 103112430023

Dosen
FAHRUDIN MUKTI WIBOWO

**PROGRAM STUDI STRUKTUR DATA
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY PURWOKERTO
2025**

A. Dasar Teori

Multi linked list merupakan struktur data pengembangan dari linked list yang memungkinkan satu node memiliki lebih dari satu pointer untuk menghubungkan ke node lain. Struktur ini digunakan untuk merepresentasikan hubungan data yang kompleks seperti hubungan satu ke banyak atau banyak ke banyak. Dalam bahasa pemrograman C++, multi linked list bersifat dinamis karena menggunakan pointer sehingga tidak memerlukan alokasi memori secara berurutan. Struktur ini sering diterapkan pada sistem yang memiliki keterkaitan data, seperti sistem akademik, struktur organisasi, dan basis data sederhana, meskipun implementasinya lebih kompleks dibandingkan linked list biasa.

B. Guided (berisi screenshot source code & output program disertai penjelasannya)

Guided 1

```

namespace X
{
    void f() { G(); }
}

class Y
{
    void g() { f(); }
}

using namespace std;
using namespace X;

struct ChildNode {
    string info;
    string parent;
    ChildNode *next;
    ChildNode *prev;
};

struct ParentNode {
    string info;
    ChildNode *child;
    ParentNode *next;
    ParentNode *prev;
};

ParentNode *createParent(string info) {
    ParentNode *parent = new ParentNode();
    parent->info = info;
    parent->child = NULL;
    parent->next = NULL;
    parent->prev = NULL;
    return parent;
}

ParentNode *createChild(string info) {
    ChildNode *node = new ChildNode();
    node->info = info;
    node->parent = " ";
    node->next = NULL;
    node->prev = NULL;
    return node;
}

void insertParent(ParentNode *head, string info) {
    if (head == createParent(info)) {
        head = head->next;
    }
    while (head->next != NULL) {
        head = head->next;
    }
    ParentNode *temp = head;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = node;
    node->prev = temp;
}
}

ParentNode *createParent(string head, string parentInfo, string childInfo) {
    ParentNode *parent = head;
    while (p != NULL && p->info != parentInfo) {
        p = p->next;
    }
    if (p != NULL) {
        ChildNode *child = createChild(childInfo);
        if (p->child == NULL) {
            p->child = child;
        } else {
            ChildNode *c = p->child;
            while (c->next != NULL) {
                c = c->next;
            }
            c->next = child;
            child->prev = c;
        }
    }
}

void printAll(ParentNode *head) {
    cout << head->info;
    ChildNode *c = head->child;
    while (c != NULL) {
        cout << " - " << c->info;
        c = c->next;
    }
    cout << endl;
    head = head->next;
}

void updateParent(ParentNode *head, string childInfo, string newNode) {
    while (p != NULL) {
        if (p->info == childInfo) {
            p->info = newNode;
            return;
        }
        p = p->next;
    }
}

void updateChild(ParentNode *head, string parentInfo,
                 string childInfo, string newChildInfo) {
    ParentNode *parent = head;
    while (p != NULL && p->info != parentInfo) {
        p = p->next;
    }
    if (p != NULL) {
        ChildNode *c = p->child;
        while (c != NULL) {
            if (c->info == childInfo) {
                c->info = newChildInfo;
                return;
            }
            c = c->next;
        }
    }
}

ParentNode *createParent(string head, string parentInfo, string childInfo) {
    ParentNode *parent = head;
    while (p != NULL && p->info != parentInfo) {
        p = p->next;
    }
    if (p != NULL) {
        ChildNode *c = p->child;
        while (c != NULL) {
            if (c->info == childInfo) {
                c->info = childInfo;
                c->parent = head;
                c->prev = NULL;
                c->next = NULL;
                c->parent->next = c;
                if (c->parent->next == NULL)
                    c->parent->next = c->parent;
                else
                    c->parent->next->prev = c->parent;
                delete c;
                return;
            }
            c = c->next;
        }
    }
}

void f() { G(); }

class Y
{
    void g() { f(); }
}

using namespace std;
using namespace X;

ParentNode *list = NULL;
insertParent(list, "Parent A");
insertParent(list, "Parent B");
insertParent(list, "Parent C");

cout << "insertAll insertParent:\n";
printAll(list);

insertChild(list, "Parent A", "Child A");
insertChild(list, "Parent A", "Child B");
insertChild(list, "Parent B", "Child A");
insertChild(list, "Parent B", "Child B");

cout << "insertAll insertChild:\n";
printAll(list);

updateParent(list, "Parent A", "Parent A");
updateChild(list, "Parent A", "Child A", "Child B");

cout << "updateAll update:\n";
printAll(list);

selectParent(list, "Parent A");
selectChild(list, "Parent A", "Child A");

cout << "selectAll select:\n";
printAll(list);
}
}

```

Screenshots Output

```
PS D:\StrukturData> cd "d:\StrukturData\Modul13\GUIDED\" ; if ($?) { g++ multilist.cpp -o multilist } ; if ($?) { .\multilist }

Setelah InsertParent:
Parent A
Parent B
Parent C

Setelah InsertChild:
Parent A -> Child A1 -> Child A2
Parent B -> Child B1
Parent C

Setelah Update:
Parent A -> Child A1* -> Child A2
Parent B* -> Child B1
Parent C

Setelah Delete:
Parent A -> Child A1*
Parent B* -> Child B1
```

Deskripsi:

Kode program tersebut mengimplementasikan struktur data multi linked list parentchild menggunakan bahasa C++. Setiap parent disusun dalam double linked list dan memiliki pointer ke linked list child yang juga bertipe double linked list. Program ini menyediakan operasi untuk menambah, menampilkan, memperbarui, dan menghapus data parent maupun child, di mana penghapusan parent juga menghapus seluruh child yang terkait. Pada fungsi main, seluruh operasi tersebut diuji untuk menunjukkan cara kerja multi linked list secara sederhana dan terstruktur.

C. Unguided/Tugas (berisi screenshot source code & output program disertai penjelasannya)

Unguided 1

```
multilist.h
Modul13 > UNGUIDED > h multilist.h > ...
1  #ifndef MULTILIST_H_INCLUDED
2  #define MULTILIST_H_INCLUDED
3  #define Nil NULL
4
5  typedef int infotypeanak;
6  typedef int infotypeinduk;
7  typedef struct elemen_list_Induk *address;
8  typedef struct elemen_list_anak *address_anak;
9
10 struct elemen_list_anak {
11     infotypeanak info;
12     address_anak next;
13     address_anak prev;
14 };
15
16 struct listanak {
17     address_anak first;
18     address_anak last;
19 };
20
21 struct elemen_list_Induk {
22     infotypeinduk info;
23     listanak lanak;
24     address next;
25     address prev;
26 };
27
28 struct listinduk {
29     address first;
30     address last;
31 };
32
Tabnine | Edit | Test | Explain | Document
33 bool ListEmpty(listinduk L);
Tabnine | Edit | Test | Explain | Document
34 bool ListEmptyAnak(listanak L);
35
Tabnine | Edit | Test | Explain | Document
36 void CreateList(listinduk &L);
Tabnine | Edit | Test | Explain | Document
37 void CreateListAnak(listanak &L);
38
Tabnine | Edit | Test | Explain | Document
39 address alokasi(infotypeinduk P);
Tabnine | Edit | Test | Explain | Document
40 address_anak alokasiAnak(infotypeanak P);
Tabnine | Edit | Test | Explain | Document
41 void dealokasi(address P);
Tabnine | Edit | Test | Explain | Document
42 void dealokasiAnak(address_anak P);
43
Tabnine | Edit | Test | Explain | Document
44 address findElm(listinduk L, infotypeinduk X);
Tabnine | Edit | Test | Explain | Document
45 address_anak findElmAnak(listanak Lanak, infotypeanak X);
Tabnine | Edit | Test | Explain | Document
46 bool ffindElm(listinduk L, address P);
Tabnine | Edit | Test | Explain | Document
47 bool ffindElmanak(listanak Lanak, address_anak P);
Tabnine | Edit | Test | Explain | Document
48 address findBefore(listinduk L, address P);
Tabnine | Edit | Test | Explain | Document
49 address_anak findBeforeAnak(listanak Lanak, address_anak P);
50
Tabnine | Edit | Test | Explain | Document
51 void insertFirst(listinduk &L, address P);
Tabnine | Edit | Test | Explain | Document
52 void insertAfter(listinduk &L, address P, address Prec);
Tabnine | Edit | Test | Explain | Document
53 void insertLast(listinduk &L, address P);
Tabnine | Edit | Test | Explain | Document
54 void insertFirstAnak(listanak &L, address_anak P);
Tabnine | Edit | Test | Explain | Document
55 void insertAfterAnak(listanak &L, address_anak P, address_anak Prec);
Tabnine | Edit | Test | Explain | Document
56 void insertLastAnak(listanak &L, address_anak P);
Tabnine | Edit | Test | Explain | Document
57 void delFirst(listinduk &L, address &P);
Tabnine | Edit | Test | Explain | Document
58 void dellast(listinduk &L, address &P);
Tabnine | Edit | Test | Explain | Document
59 void delAfter(listinduk &L, address &P, address Prec);
Tabnine | Edit | Test | Explain | Document
60 void delP(listinduk &L, infotypeinduk X);
Tabnine | Edit | Test | Explain | Document
61 void delFirstAnak(listanak &L, address_anak &P);
Tabnine | Edit | Test | Explain | Document
62 void dellastAnak(listanak &L, address_anak &P);
Tabnine | Edit | Test | Explain | Document
63 void delAfterAnak(listanak &L, address_anak &P, address_anak Prec);
Tabnine | Edit | Test | Explain | Document
64 void delAnak(listanak &L, infotypeanak X);
Tabnine | Edit | Test | Explain | Document
65 void printInfo(listinduk L);
Tabnine | Edit | Test | Explain | Document
66 int nbList(listinduk L);
Tabnine | Edit | Test | Explain | Document
67 void printInfoAnak(listanak Lanak);
Tabnine | Edit | Test | Explain | Document
68 int nbListAnak(listanak Lanak);
Tabnine | Edit | Test | Explain | Document
69 void delAll(listinduk &L);
Tabnine | Edit | Test | Explain | Document
70 void delAllAnak(listanak &L);
71
72 #endif
```

```
multilist.cpp X
Modul13 > UNGUIDED > C++ multilist.cpp > ↗ alokasi(infotypeinduk)
1 #include "multilist.h"
2 #include <iostream>
3 using namespace std;
4
5 Tabnine | Edit | Test | Explain | Document
6 bool ListEmpty(listinduk L) {
7     return L.first == Nil;
8 }
9
10 Tabnine | Edit | Test | Explain | Document
11 bool ListEmptyAnak(listanak L) {
12     return L.first == Nil;
13 }
14
15 Tabnine | Edit | Test | Explain | Document
16 void CreateList(listinduk &L) {
17     L.first = Nil;
18     L.last = Nil;
19 }
20
21 Tabnine | Edit | Test | Explain | Document
22 void CreateListAnak(listanak &L) {
23     L.first = Nil;
24     L.last = Nil;
25 }
26
27 Tabnine | Edit | Test | Explain | Document
28 address alokasi(infotypeinduk X) {
29     address P = new elemen_list_induk;
30     if (P != Nil) {
31         P->info = X;
32         P->next = Nil;
33         P->prev = Nil;
34         CreateListAnak(P->lanak);
35     }
36     return P;
37 }
38
39 Tabnine | Edit | Test | Explain | Document
40 address_anak alokasiAnak(infotypeanak X) {
41     address_anak P = new elemen_list_anak;
42     if (P != Nil) {
43         P->info = X;
44         P->next = Nil;
45         P->prev = Nil;
46     }
47     return P;
48 }
49
50 Tabnine | Edit | Test | Explain | Document
51 void dealokasi(address P) {
52     delete P;
53 }
54
55 Tabnine | Edit | Test | Explain | Document
56 void dealokasiAnak(address_anak P) {
57     delete P;
58 }
59
60 Tabnine | Edit | Test | Explain | Document
61 address findElm(listinduk L, infotypeinduk X) {
62     address P = L.first;
63     while (P != Nil) {
64         if (P->info == X) {
65             return P;
66         }
67         P = P->next;
68     }
69     return Nil;
70 }
71
72 Tabnine | Edit | Test | Explain | Document
73 address_anak findElmAnak(listanak Lanak, infotypeanak X) {
74     address_anak P = Lanak.first;
75     while (P != Nil) {
76         if (P->info == X) {
77             return P;
78         }
79         P = P->next;
80     }
81     return Nil;
82 }
83
84 Tabnine | Edit | Test | Explain | Document
85 bool ffindElm(listinduk L, address P) {
86     address Q = L.first;
87     while (Q != Nil) {
88         if (Q == P) {
89             return true;
90         }
91         Q = Q->next;
92     }
93     return false;
94 }
95
96 Tabnine | Edit | Test | Explain | Document
97 bool ffindElmAnak(listanak Lanak, address_anak P) {
98     address_anak Q = Lanak.first;
99     while (Q != Nil) {
100         if (Q == P) {
101             return true;
102         }
103         Q = Q->next;
104     }
105     return false;
106 }
107
108 Tabnine | Edit | Test | Explain | Document
109 address_findElm(listinduk L, address P) {
110 }
```

```
multilist.cpp ×
Modul13 > UNGUIDED > C++ multilist.cpp > ...
95 Tabnine | Edit | Test | Explain | Document
96 address findBefore(listinduk L, address P) {
97     address Q = L.first;
98     if (Q == P) {
99         return Nil;
100    }
101    while (Q != Nil && Q->next != P) {
102        Q = Q->next;
103    }
104    return Q;
105 }
106 Tabnine | Edit | Test | Explain | Document
107 address_anak findBeforeAnak(listanak Lanak, address_anak P) {
108     address_anak Q = Lanak.first;
109     if (Q == P) {
110         return Nil;
111     }
112     while (Q != Nil && Q->next != P) {
113         Q = Q->next;
114     }
115     return Q;
116 }
117 Tabnine | Edit | Test | Explain | Document
118 void insertFirst(listinduk &L, address P) {
119     if (ListEmpty(L)) {
120         L.first = P;
121         L.last = P;
122         P->next = Nil;
123         P->prev = Nil;
124     } else {
125         P->next = L.first;
126         P->prev = Nil;
127         L.first->prev = P;
128         L.first = P;
129     }
130 }
131 Tabnine | Edit | Test | Explain | Document
132 void insertAfter(listinduk &L, address P, address Prec) {
133     if (Prec != Nil) {
134         P->next = Prec->next;
135         P->prev = Prec;
136         if (Prec->next != Nil) {
137             Prec->next->prev = P;
138         } else {
139             L.last = P;
140         }
141         Prec->next = P;
142     }
143 }
144 Tabnine | Edit | Test | Explain | Document
145 void insertLast(listinduk &L, address P) {
146     if (ListEmpty(L)) {
147         L.first = P;
148         L.last = P;
149         P->next = Nil;
150         P->prev = Nil;
151     } else {
152         P->prev = L.last;
153         P->next = Nil;
154         L.last->next = P;
155         L.last = P;
156     }
157 }
158 Tabnine | Edit | Test | Explain | Document
159 void insertFirstAnak(listanak &L, address_anak P) {
160     if (ListEmptyAnak(L)) {
161         L.first = P;
162         L.last = P;
163         P->next = Nil;
164         P->prev = Nil;
165     } else {
166         P->next = L.first;
167         P->prev = Nil;
168         L.first->prev = P;
169         L.first = P;
170     }
171 }
172 Tabnine | Edit | Test | Explain | Document
173 void insertAfterAnak(listanak &L, address_anak P, address_anak Prec) {
174     if (Prec != Nil) {
175         P->next = Prec->next;
176         P->prev = Prec;
177         if (Prec->next != Nil) {
178             Prec->next->prev = P;
179         } else {
180             L.last = P;
181         }
182         Prec->next = P;
183     }
184 }
185 Tabnine | Edit | Test | Explain | Document
186 void insertLastAnak(listanak &L, address_anak P) {
```

```

+ multilist.cpp X
Modul13 > UNGUIDED > C++ multilist.cpp > ↗ insertAfter(listinduk &, address, address)
173 void insertAfterAnak(listanak &L, address_anak P, address_anak Prec) {
174     if (Prec != Nil) {
175         P->next = Prec;
176         Prec->prev = P;
177     }
178 }
179
Tabnine | Edit | Test | Explain | Document
186 void insertLastAnak(listanak &L, address_anak P) {
187     if (!ListEmptyAnak(L)) {
188         L.first = P;
189         L.last = P;
190         P->next = Nil;
191         P->prev = Nil;
192     } else {
193         P->prev = L.last;
194         P->next = Nil;
195         L.last->next = P;
196         L.last = P;
197     }
198 }
199
Tabnine | Edit | Test | Explain | Document
200 void delFirst(listinduk &L, address &P) {
201     if (!ListEmpty(L)) {
202         P = L.first;
203         if (L.first == L.last) {
204             L.first = Nil;
205             L.last = Nil;
206         } else {
207             L.first = L.first->next;
208             L.first->prev = Nil;
209             P->next = Nil;
210         }
211     }
212 }
213
Tabnine | Edit | Test | Explain | Document
214 void dellast(listinduk &L, address &P) {
215     if (!ListEmpty(L)) {
216         P = L.last;
217         if (L.first == L.last) {
218             L.first = Nil;
219             L.last = Nil;
220         } else {
221             L.last = L.last->prev;
222             L.last->next = Nil;
223             P->prev = Nil;
224         }
225     }
226 }
227
Tabnine | Edit | Test | Explain | Document
228 void delAfter(listinduk &L, address &P, address Prec) {
229     if (Prec != Nil && Prec->next != Nil) {
230         P = Prec->next;
231         Prec->next = P->next;
232         if (P->next != Nil) {
233             P->next->prev = Prec;
234         } else {
235             L.last = Prec;
236         }
237         P->next = Nil;
238         P->prev = Nil;
239     }
240 }
241
Tabnine | Edit | Test | Explain | Document
242 void delP(listinduk &L, infotypeinduk X) {
243     address P = findElm(L, X);
244     if (P != Nil) {
245
246         delAllAnak(P->lanak);
247
248         if (P == L.first) {
249             delFirst(L, P);
250         } else if (P == L.last) {
251             dellast(L, P);
252         } else {
253             address Prec = P->prev;
254             delAfter(L, P, Prec);
255         }
256         dealokasi(P);
257     }
258 }
259
Tabnine | Edit | Test | Explain | Document
260 void delFirstAnak(listanak &L, address_anak &P) {
261     if (!ListEmptyAnak(L)) {
262         P = L.first;
263         if (L.first == L.last) {
264             L.first = Nil;
265             L.last = Nil;
266         } else {
267             L.first = L.first->next;
268             L.first->prev = Nil;
269             P->next = Nil;
270         }
271     }
272 }
273
Tabnine | Edit | Test | Explain | Document
274 void dellastAnak(listanak &L, address_anak &P) {
275     if (!ListEmptyAnak(L)) {
276

```

```

C++ multilist.cpp
Modul13 > UNGUIDED > C++ multilist.cpp > delP(listanak &L, infotypeinduk)
268 void delFirstAnak(listanak &L, address_anak &P) {
269 }
270
271 Tabnine | Edit | Test | Explain | Document
272 void delLastAnak(listanak &L, address_anak &P) {
273     if (!ListEmptyAnak(L)) {
274         P = L.last;
275         if (L.first == L.last) {
276             L.first = Nil;
277             L.last = Nil;
278         } else {
279             L.last = L.last->prev;
280             L.last->next = Nil;
281             P->prev = Nil;
282         }
283     }
284 }
285
286 }
287
288 Tabnine | Edit | Test | Explain | Document
289 void delAfterAnak(listanak &L, address_anak &P, address_anak Prec) {
290     if (Prec != Nil && Prec->next != Nil) {
291         P = Prec->next;
292         Prec->next = P->next;
293         if (P->next != Nil) {
294             P->next->prev = Prec;
295         } else {
296             L.last = Prec;
297         }
298         P->next = Nil;
299         P->prev = Nil;
300     }
301 }
302
303 Tabnine | Edit | Test | Explain | Document
304 void delPAnak(listanak &L, infotypeanak X) {
305     address_anak P = findElmAnak(L, X);
306     if (P != Nil) {
307         if (P == L.first) {
308             delFirstAnak(L, P);
309         } else if (P == L.last) {
310             delLastAnak(L, P);
311         } else {
312             address_anak Prec = P->prev;
313             delAfterAnak(L, P, Prec);
314         }
315     }
316 }
317
318 Tabnine | Edit | Test | Explain | Document
319 void printInfo(listinduk L) {
320     address P = L.first;
321     cout << "Data Induk dan Anak" << endl;
322     while (P != Nil) {
323         cout << "Induk: " << P->info << endl;
324         cout << " Anak: ";
325         printInfoAnak(P->lanak);
326         cout << endl;
327         P = P->next;
328     }
329 }
330
331 Tabnine | Edit | Test | Explain | Document
332 void printInfoAnak(listanak Lanak) {
333     address_anak P = Lanak.first;
334     if (!ListEmptyAnak(Lanak)) {
335         cout << "(kosong)";
336     } else {
337         while (P != Nil) {
338             cout << P->info;
339             if (P->next != Nil) {
340                 cout << ", ";
341             }
342             P = P->next;
343         }
344     }
345 }
346
347 Tabnine | Edit | Test | Explain | Document
348 int nbList(listinduk L) {
349     int count = 0;
350     address P = L.first;
351     while (P != Nil) {
352         count++;
353         P = P->next;
354     }
355     return count;
356 }
357
358 Tabnine | Edit | Test | Explain | Document
359 int nbListAnak(listanak Lanak) {
360     int count = 0;
361     address_anak P = Lanak.first;
362     while (P != Nil) {
363         count++;
364         P = P->next;
365     }
366     return count;
367 }
368
369 Tabnine | Edit | Test | Explain | Document
370 void delAll(listinduk &L) {
371     address P;
372     while (!ListEmpty(L)) {
373         delFirst(L, P);
374         // Hapus semua anak
375         delAllAnak(P->lanak);
376         dealokasi(P);
377     }
378 }
379

```

```
Tabnine | Edit | Test | Explain | Document
364 void delAll(ListInduk &L) {
365     address P;
366     while (!ListEmpty(L)) {
367         delFirst(L, P);
368         // Hapus semua anak
369         delAllAnak(P->lanak);
370         dealokasi(P);
371     }
372 }
373
Tabnine | Edit | Test | Explain | Document
374 void delAllAnak(ListAnak &L) {
375     address_anak PA;
376     while (!ListEmptyAnak(L)) {
377         delFirstAnak(L, PA);
378         dealokasiAnak(PA);
379     }
380 }
```

```
C++ main.cpp X
Modul13 > UNGUIDED > C++ main.cpp > main()
1 #include "multilist.h"
2 #include "multilist.cpp"
3 #include <iostream>
4 using namespace std;
5
Tabnine | Edit | Test | Explain | Document
6 int main() {
7     listInduk L;
8     address P_Induk;
9     address_anak P_anak;
10
11    CreateList(L);
12
13    cout << "Menambah data Parent" << endl;
14    P_Induk = alokasi(1);
15    insertFirst(L, P_Induk);
16
17    P_Induk = alokasi(2);
18    insertLast(L, P_Induk);
19
20    P_Induk = alokasi(3);
21    insertLast(L, P_Induk);
22
23    cout << "Menambah anak untuk parent 1" << endl;
24    P_Induk = findElm(L, 1);
25    if (P_Induk != Nil) {
26        P_anak = alokasiAnak(101);
27        insertFirstAnak(P_Induk->lanak, P_anak);
28
29        P_anak = alokasiAnak(102);
30        insertLastAnak(P_Induk->lanak, P_anak);
31
32        P_anak = alokasiAnak(103);
33        insertLastAnak(P_Induk->lanak, P_anak);
34    }
35
36    cout << "Menambah anak untuk parent 2" << endl;
37    P_Induk = findElm(L, 2);
38    if (P_Induk != Nil) {
39        P_anak = alokasiAnak(201);
40        insertFirstAnak(P_Induk->lanak, P_anak);
41
42        P_anak = alokasiAnak(202);
43        insertLastAnak(P_Induk->lanak, P_anak);
44    }
45
46    cout << "Menambah anak untuk parent 3" << endl;
47    P_Induk = findElm(L, 3);
48    if (P_Induk != Nil) {
49        P_anak = alokasiAnak(301);
50        insertFirstAnak(P_Induk->lanak, P_anak);
51    }
52
53    cout << endl;
54    printInfo(L);
55
56    cout << endl << "Menghapus anak 102 dari parent 1..." << endl;
57    P_Induk = findElm(L, 1);
58    if (P_Induk != Nil) {
59        delPAnak(P_Induk->lanak, 102);
60    }
61
62    printInfo(L);
63
64    cout << endl << "Menghapus parent 2 beserta semua anaknya..." << endl;
65    delP(L, 2);
66
67    printInfo(L);
68
69    cout << endl << "Jumlah parent: " << nbList(L) << endl;
70
71    delAll(L);
72
73    return 0;
74 }
```

Screenshots Output

```
PS D:\StrukturData> cd "d:\StrukturData\Modul13\UNGUIDED\" ; if ($?) { g++ main.cpp -o main } ; if ($?) { .\main }

Menambah anak untuk parent 1
Menambah anak untuk parent 2
Menambah anak untuk parent 3

==== Data Induk dan Anak ====
Induk: 1
    Anak: 101, 102, 103
Induk: 2
    Anak: 201, 202
Induk: 3
    Anak: 301

Menghapus anak 102 dari parent 1...
==== Data Induk dan Anak ====
Induk: 1
    Anak: 101, 103
Induk: 2
    Anak: 201, 202
Induk: 3
    Anak: 301

Menghapus parent 2 beserta semua anaknya...
==== Data Induk dan Anak ====
Induk: 1
    Anak: 101, 103
Induk: 3
    Anak: 301

Jumlah parent: 2
```

Deskripsi:

Kode ADT tersebut mengimplementasikan circular singly linked list menggunakan bahasa C++. File circularlist.h mendefinisikan struktur data berupa infotype untuk menyimpan data mahasiswa, node ElmList, serta struktur List yang memiliki pointer first. File circularlist.cpp berisi implementasi operasi dasar circular linked list seperti inisialisasi list, alokasi dan dealokasi node, penyisipan data di awal, tengah, dan akhir list, penghapusan node, pencarian data berdasarkan NIM, serta penampilan seluruh isi list secara melingkar. Pada mainn.cpp, program digunakan untuk menguji ADT dengan membuat data mahasiswa, melakukan berbagai operasi insert dan pencarian, lalu menampilkan hasilnya untuk menunjukkan bahwa list bersifat melingkar, yaitu node terakhir selalu menunjuk kembali ke node pertama.

Unguided 2

```
h circularlist.h X
Modul13 > UNGUIDED > h circularlist.h > ...
1 #ifndef CIRCULARLIST_H_INCLUDED
2 #define CIRCULARLIST_H_INCLUDED
3 #define Nil NULL
4
5 #include <string>
6 using namespace std;
7
8 struct infotype {
9     string nama;
10    string nim;
11    char jenis_kelamin;
12    float ipk;
13 };
14
15 typedef struct ElmList *address;
16
17 struct ElmList {
18     infotype info;
19     address next;
20 };
21
22 struct List {
23     address first;
24 };
25
26 Tabnine | Edit | Test | Explain | Document
27 void createList(List &L);
28 Tabnine | Edit | Test | Explain | Document
29 address alokasi(infotype x);
30 Tabnine | Edit | Test | Explain | Document
31 void dealokasi(address P);
32 Tabnine | Edit | Test | Explain | Document
33 void insertFirst(List &L, address P);
34 Tabnine | Edit | Test | Explain | Document
35 void insertAfter(List &L, address Prec, address P);
36 Tabnine | Edit | Test | Explain | Document
37 void insertLast(List &L, address P);
38 Tabnine | Edit | Test | Explain | Document
39 void deleteFirst(List &L, address &P);
40 Tabnine | Edit | Test | Explain | Document
41 void deleteAfter(List &L, address Prec, address &P);
42 Tabnine | Edit | Test | Explain | Document
43 void deleteLast(List &L, address &P);
44 Tabnine | Edit | Test | Explain | Document
45 address findElm(List L, infotype x);
46 Tabnine | Edit | Test | Explain | Document
47 void printInfo(List L);
48
49 #endif
```

```

1 circularList.cpp *
Module 3 > UNLINKED > circularList.cpp > diskokasi[address]
1 #include <iostream>
2 #include "circularList.h"
3 using namespace std;
4
5 int main () {
6     cout << "Selamat datang di program Circular List" << endl;
7     cout << "Silakan pilih menu yang tersedia:" << endl;
8     cout << "1. Insert First" << endl;
9     cout << "2. Insert After" << endl;
10    cout << "3. Insert Last" << endl;
11    cout << "4. Delete First" << endl;
12    cout << "5. Delete After" << endl;
13    cout << "6. Delete Last" << endl;
14    cout << "7. Find Info" << endl;
15    cout << "8. Print Info" << endl;
16    cout << "9. Exit" << endl;
17
18    int choice;
19    cin >> choice;
20
21    switch (choice) {
22        case 1:
23            insertFirst();
24            break;
25        case 2:
26            insertAfter();
27            break;
28        case 3:
29            insertLast();
30            break;
31        case 4:
32            deleteFirst();
33            break;
34        case 5:
35            deleteAfter();
36            break;
37        case 6:
38            deleteLast();
39            break;
40        case 7:
41            findInfo();
42            break;
43        case 8:
44            printInfo();
45            break;
46        case 9:
47            exit(0);
48        default:
49            cout << "Pilihan tidak valid. Silakan coba lagi." << endl;
50    }
51
52    system("cls");
53}

```

C++ mainn.cpp X

```
Modul13 > UNGUIDED > C++ mainn.cpp > main()
1 #include <iostream>
2 #include "circularlist.h"
3 #include "circularlist.cpp"
4 using namespace std;
5
6 Tabnine | Edit | Test | Explain | Document
7 address createData(string nama, string nim, char jenis_kelamin, float ipk)
8 {
9     infotype x;
10    x.nama = nama;
11    x.nim = nim;
12    x.jenis_kelamin = jenis_kelamin;
13    x.ipk = ipk;
14    return alokasi(x);
15
16 Tabnine | Edit | Test | Explain | Document
17 int main()
18 {
19     List L;
20     address P1 = Nil;
21     address P2 = Nil;
22     infotype x;
23     createList(L);
24
25     cout<<"coba insert first, last, dan after"<<endl;
26     P1 = createData("Danu", "04", 'l', 4.0);
27     insertFirst(L,P1);
28
29     P1 = createData("Fahmi", "06", 'l', 3.45);
30     insertLast(L,P1);
31
32     P1 = createData("Bobi", "02", 'l', 3.71);
33     insertFirst(L,P1);
34
35     P1 = createData("Ali", "01", 'l', 3.3);
36     insertFirst(L,P1);
37
38     P1 = createData("Gita", "07", 'p', 3.75);
39     insertLast(L,P1);
40
41     x.nim = "07";
42     P1 = findElm(L,x);
43     P2 = createData("Cindi", "03", 'p', 3.5);
44     insertAfter(L, P1, P2);
45
46     x.nim = "02";
47     P1 = findElm(L,x);
48     P2 = createData("Hilmi", "08", 'p', 3.3);
49     insertAfter(L, P1, P2);
50
51     x.nim = "04";
52     P1 = findElm(L,x);
53     P2 = createData("Eli", "05", 'p', 3.4);
54     insertAfter(L, P1, P2);
55     printInfo(L);
56 }
```

Screenshots Output

```
PS D:\StrukturData> cd "d:\StrukturData\Modul13\UNGUIDED\" ; if ($?) { g++ mainn.cpp -o mainn } ; if ($?) { .\mainn }

● coba insert first, last, dan after
Nama : Ali
NIM : 01
L/P : l
IPK : 3.3

Nama : Bobi
NIM : 02
L/P : l
IPK : 3.71

Nama : Hilmi
NIM : 08
L/P : p
IPK : 3.3

Nama : Danu
NIM : 04
L/P : l
IPK : 4

Nama : Eli
NIM : 05
L/P : p
IPK : 3.4

Nama : Fahmi
NIM : 06
L/P : l
IPK : 3.45

Nama : Gita
NIM : 07
L/P : p
IPK : 3.75

Nama : Cindi
NIM : 03
L/P : p
IPK : 3.5
```

Deskripsi:

Kode ADT tersebut mengimplementasikan struktur data multi linked list bertipe parent child menggunakan bahasa C++ dengan konsep double linked list pada data induk maupun data anak. File multilist.h mendefinisikan struktur node induk yang memiliki pointer ke list anak, sedangkan multilist.cpp berisi implementasi operasi dasar seperti pembuatan list, alokasi dan dealokasi memori, pencarian, penyisipan, penghapusan parent dan anak, perhitungan jumlah elemen, serta penampilan seluruh data. Pada file main.cpp, ADT diuji dengan menambahkan beberapa parent dan anak, menampilkan hubungan parent-child, menghapus anak maupun parent beserta seluruh anaknya, sehingga menunjukkan bahwa struktur multi linked list bekerja secara terorganisir dan aman dalam pengelolaan data relasional.

D. Kesimpulan

Dari hasil pembuatan dan pengujian program, dapat disimpulkan bahwa struktur data linked list, baik circular linked list maupun multi linked list, dapat digunakan untuk mengelola data secara dinamis menggunakan bahasa C++. Multi linked list cocok untuk menyimpan data yang memiliki hubungan parent dan child karena setiap data induk dapat memiliki beberapa data anak. Sementara itu, circular linked list memungkinkan data saling terhubung secara melingkar sehingga dapat ditelusuri terus menerus tanpa batas akhir.

Penerapan ADT dengan pemisahan file header, source, dan main membuat program lebih rapi dan mudah dipahami. Operasi dasar seperti insert, delete, search, dan print dapat berjalan dengan baik sesuai dengan konsep struktur data yang digunakan. Dengan demikian, penggunaan linked list membantu memahami cara kerja pointer, manajemen memori, dan pengolahan data yang lebih fleksibel dibandingkan struktur data statis.

E. Referensi

<https://firmaninformatika.blogspot.com/2014/11/referensi-multiple-linked-list-multi.html>

<https://www.geeksforgeeks.org/multilevel-linked-list/>

<https://www.geeksforgeeks.org/data-structures/linked-list/>

<https://www.programiz.com/dsa/linked-list>