MODUL 8

MULTILIST

Tujuan :

1. Praktikan mampu memahami konsep dari Multilist
2. Praktikan mampu mengimplementasikan konsep Multilist pada Bahasa pemrograman C
3. Praktikan mampu mengimplementasikan konsep Multilist untuk menyelesaikan masalah yang ada

Teori :

Multilist (Multi Linked List) merupakan pengembangan lebih lanjut dari struktur data Linklist (Linked List). Apabila Linklist hanya memiliki 1 node penunjuk ke node lainnya, Multilist memiliki 2 node, dimana satu node menunjuk ke Node Parent Lainnya dan Satu Node lainnya akan menunjuk ke Node Anak. Sehingga, Multilist memiliki 2 bagian, yang bisa disebut sebagai Parent dan Child

Untuk lebih jelas, berikut Detail dari Parent dan Child yang ada pada Multilist :

1. Parent
   Parent merupakan Node yang memiliki 2 pointer. 1 pointer untuk ke Parent lainnya, dan 1 pointer lagi untuk menunjuk ke Child.

2. Child
   Child merupakan Node yang hanya memiliki 1 Pointer. Pointer tersebut akan menunjuk ke Child lainnya. Child dasarnya sama seperti Linklist yang telah dipelajari di modul sebelumnya. Hanya saja, Child harus bergantung ke sebuah parent dan tidak bisa berdiri sendiri.

Ilustrasi dari Multilist adalah sebagai berikut :

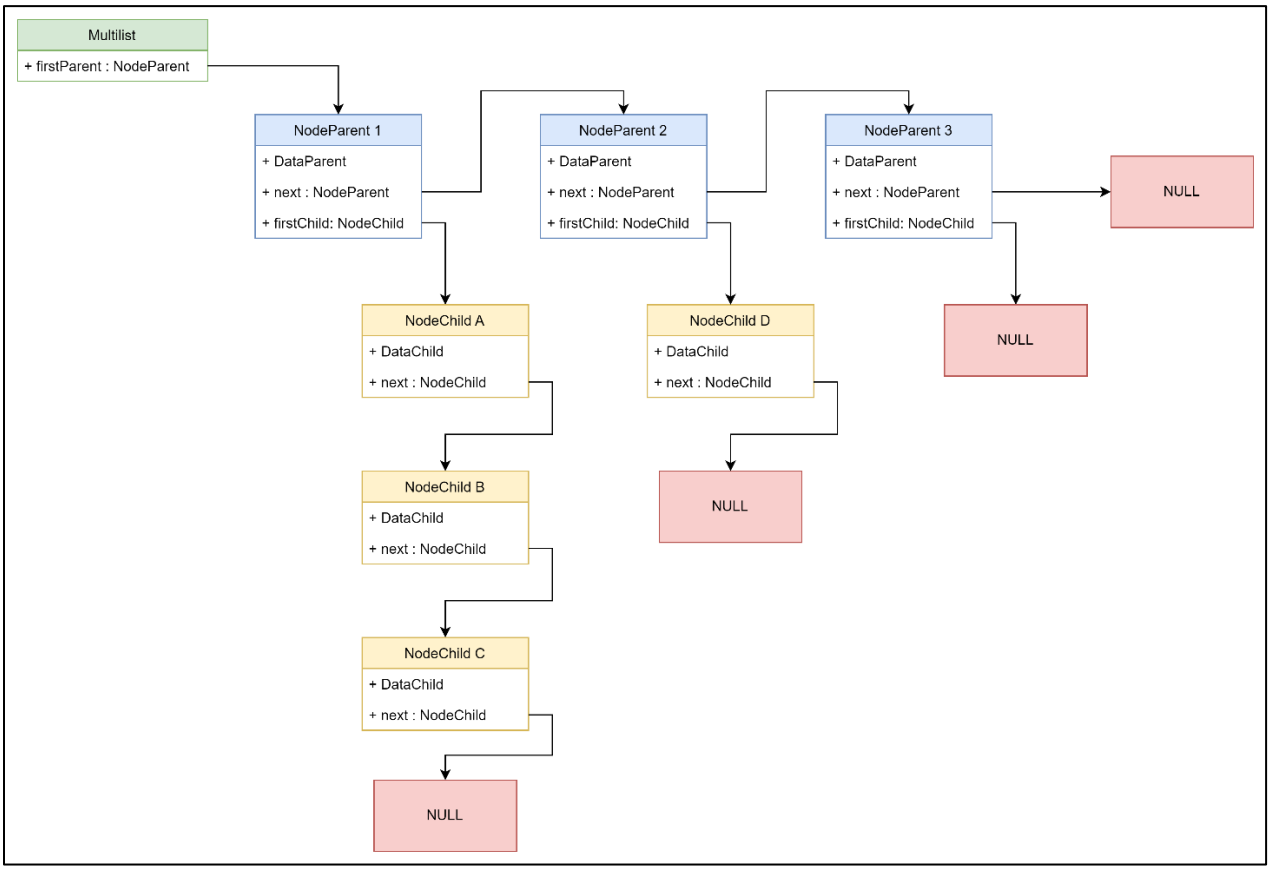
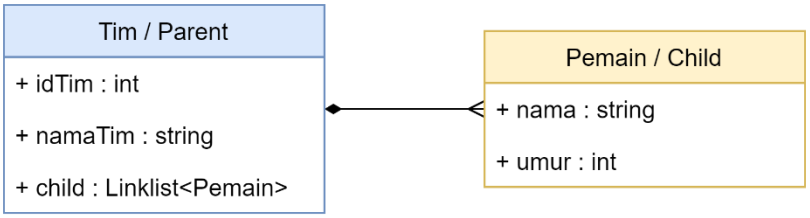


Diagram Multilist Utama

Implementasi Multilist pada Bahasa Pemrograman C

Untuk memudahkan teman-teman dalam belajar Multilist, berikut adalah *breakdown* untuk setiap code yang dibutuhkan untuk membuat struktur data Multilist.

Pada Guided, kita akan membuat sebuah program pendataan tim sederhana. Entitasnya adalah sebagai berikut.



Berikut adalah daftar fungsi-fungsi yang dibutuhkan untuk membuat program kita menggunakan struktur data Multilist.

A. Umum

1. Struct Data Parent

```
1
2 typedef struct DataParent{
3
4     int idTim;
5     string namaTim;
6
7 }DataParent;
8
```

Kita akan membuat struct untuk **DataParent**. Disini kita membuat sebuah struct baru yang akan menampung **DataParent** yang ingin kita simpan.

Semua atribut dari **DataParent** kita simpan disini. Dapat terlihat kita menyimpan **int idTim** dan **string namaTim**.

2. Struct Data Child

```
1
2 typedef struct DataChild{
3
4     string nama;
5     int umur;
6
7 }DataChild;
8
```

Sama hal-nya dengan **DataParent**, kita membuat struct baru untuk menyimpan **DataChild**.

Semua atribut dari **DataChild** kita simpan disini. Dapat dilihlat kita menyimpan **string nama** dan **int umur**.

3. Node Multilist

```
typedef struct{

    AddressParent firstParent;

}Multilist;
```

Tidak ada yang spesial.

Disini kita membuat **Sturct Multilist** yang akan digunakan untuk menampung **Struct Parent Utama** kita. Sama halnya seperti yang telah teman-teman pelajari di Linklist

4. Node Parent

```
typedef struct Parent* AddressParent;

typedef struct Parent{

    DataParent dataParent;
    AddressParent next;

    AddressChild firstChild;

}NodeParent;
```

Ini adalah **NodeParent**. Bisa dilhat bahwa **NodeParent** akan memiliki **2 buah pointer**

Pointer **next** yang merupakan **AddressParent**. Digunakan untuk menunjuk **Parent** selanjutnya

Pointer **firstChild** merupakan **AddressChild**. Digunakan untuk menunjuk **Child Pertama** yang dimiliki.

Perhatikan bahwa **NodeParent** memiliki **DataParent** sebagai data yang disimpan

5. Node Child

```
typedef struct Child* AddressChild;

typedef struct Child{

    DataChild dataChild;
    AddressChild next;

}NodeChild;
```

Ini adalah **NodeChild**. Benar-benar mirip dengan Linklist, karena pada dasarnya **Child** merupakan Linklist yang telah teman-teman pelajari

Terdapat Pointer **next** merupakan **AddressChild**. Digunakan untuk menunjuk **Child** selanjutnya.

NodeChild juga memiliki **DataChild** sebagai data yang disimpan.

6. Make Data Parent

```
1
2 DataParent makeDataParent(int id, string namaTim){
3
4     DataParent data;
5
6     data.idTim = id;
7     strcpy(data.namaTim, namaTim);
8
9     return data;
10 }
11
```

MakeDataParent adalah fungsi untuk membuat **DataParent.**
Jadi, ketika teman-teman sudah menginputkan atribut-atribut yang dibutuhkan, gunakanlah fungsi ini untuk membuat data-nya.

Jangan lupa untuk menginisialisasikan semua atribut yang diperlukan. Dan jangan lupa me-**return**-kan hasil yang sudah dibuat (***Line 9***)

7. Make Data Child

```
1
2 DataChild makeDataChild(string nama, int umur){
3
4     DataChild data;
5
6     data.umur = umur;
7     strcpy(data.nama, nama);
8
9     return data;
10 }
11
```

Mirip seperti *MakeDataParent*, hanya saja kita akan membuat **DataChild**.

Jangan sampai terkecoh dengan atribut-atribut yang diinisialisasi. Apabila terdapat **tambahan tinggi badan** misal, maka tambahkanlah di **parameter** pada fungsi dan **inisialisasikan nilai bersangkutan**.

B. Parent

Berikut ini adalah fungsi-fungsi dasar yang bersangkutan dengan Node Parent. Silahkan dicermati

1. Create Empty (Inisialisasi)

```
1
2 void createEmpty(Multilist *l){
3
4     l->firstParent = NULL;
5
6 }
7
```

CreateEmpty akan menginisialisasi **Struct Multilist**. Simplenya kita akan membuat **FirstParent** menjadi **NULL**.

Perlu diingat bahwa FirstParent bertipe data **AddressParent**, dimana **AddressParent** merupakan **Pointer of Struct Parent**. Sehingga kita bisa menginisialisasi nilainya dengan **NULL**

Jangan lupa menggunakan pointer :)

2. Is Empty (?)

```
1
2 bool isEmpty(Multilist l){
3
4     return l.firstParent == NULL;
5
6 }
7
```

Tugas IsEmpty sama seperti sebelum-sebelumnya. Dia bakalan memeriksa apakah **Multilist** itu kosong atau tidak (ada isinya).

Return **True** apabila Kosong
Return **False** apabila Tidak Kosong

3. Have Child (?)

```
1
2 bool haveChild(AddressParent ap){
3
4     return ap->firstChild != NULL;
5
6 }
7
```

Tugas haveChild sangat simple, jangan dipikir pusing yoo…
Intinya adalah dia bakalan memeriksa apakah **Parent** bersangkutan memiliki **Child** atau tidak.

Cth: lihat *Diagram Multilist Utama*

haveChild (**NodeParent1**) -> Return **True**
haveChild (**NodeParent2**) -> Return **True**
haveChild (**NodeParent3**) -> Return **False**

4. Alokasi Parent

```
1
2 AddressParent alokasiParent(Pelanggan data){
3
4     AddressParent ap;
5
6     ap = (AddressParent) malloc(sizeof(NodeParent));
7
8     ap->next = NULL;
9     ap->firstChild = NULL;
10    ap->dataParent = data;
11
12    return ap;
13 }
14
```

AlokasiParent akan membuat sebuah **Node Parent**. Sama hal-nya seperti pada Linklist.

Perbedaan yang paling jelas adalah kita harus men-set **2 buah** pointer. yakni : **Pointer Next** dan **Pointer FirstChild** menjadi **NULL**

Selebihnya penggunaan malloc tinggal disesuaikan dengan Tipe data-nya, yani **AddressParent** sebagai tipe casting-nya dan **NodeParent** sebagai Size Node-nya

5. Find Parent

```
1
2  AddressParent findParent(Multilist l, int cariId){
3
4      AddressParent parentBantu = NULL;
5
6      parentBantu = l.firstParent;
7
8      while(parentBantu != NULL){
9          if(parentBantu->dataParent.idTim == cariId){
10             return parentBantu;
11         }
12         parentBantu = parentBantu->next;
13     }
14
15     return NULL;
16 }
17
```

findParent merupakan fungsi yang akan mencari **NodeParent** berdasarkan atribut yang dicari.
Pada contoh, kita mencari **NodeParent** berdasarkan **IdTim**

Line 8-13 akan memastikan kita mencari semua Node hingga Node tersebut **NULL** (habis).

Line 9-11 akan memeriksa apakah Node bersangkutan merupakan Node yang kita cari atau bukan. Jika **iya**, maka kita akan langsung me-return-kan **Node** tersebut.
Apabila **tidak di dapat**, maka kita akan me-return-kan **NULL** (***Line 15***)

*Jangan Monoton!. Kita bisa aja mencari berdasarkan Nama Tim. Ubahlah kondisi pada if (**Line 9**) untuk mencari berdasarkan atribut lain. Jangan lupa gunakan strcmpi untuk string. Silahkan di explore yaaa…*

6. Insert First Parent

```
1
2  void insertFirstParent(Multilist *l, DataParent data){
3
4      AddressParent dataParent = alokasiParent(data);
5
6      dataParent->next = l->firstParent;
7      l->firstParent = dataParent;
8
9  }
10
```

Sama hal-nya seperti Linklist, insertFirstParent akan menginputkan Node Pertama ke dalam **Multilist**. Tidak ada perbedaan signifikan terkecuali tipe data yang digunakan.

Untuk langkah ataupun kegunaan ***Line 6-7*** sudah sangat dijelaskan pada Modul Linklist 1 dan 2, silahkan dibaca ulang modulnya untuk lebih mengerti alur program.

7. Insert Last Parent

```
1
2  void insertLastParent(Multilist *l, DataParent data){
3
4      AddressParent dataParent = alokasiParent(data);
5
6      if(isEmpty(*l)){
7
8          insertFirstParent(l, data);
9
10     }else{
11
12         AddressParent temp = l->firstParent;
13         while(temp->next != NULL){
14             temp = temp->next;
15         }
16
17         temp->next = dataParent;
18     }
19 }
20
```

InsertLastParent juga mirip konsepnya dengan Linklist.
Intinya apabila Node **masih kosong**, maka kita harus menggunakan **InsertFirstParent**.
Apabila sudah terisi, maka carilah Node Terakhir (***Line 12-16***) lalu inputlah data-nya sebagai NodeTerakhir->next (***Line 18***)

*Jangan lupa untuk menggunakan **Pointer!***
*Perhatikan pula pada **Line 8**, kita passing parameter menggunakan l dan bukan &(*l).*

Bisa demikian dikarenakan l merupakan Multilist, lalu parameter yang dibutuhkan InsertFirstParent juga Multilist*. Sehingga kita bisa langsung passing l ke dalam parameter InsertFirstParent.*

*Tetapi menggunakan &(*l) juga benar. Gunakan senyamannya :)*

8. Insert (*At/After*) Parent

```
1
2 //sebenarnya lebih tepat disebut insertAfter
3 void insertAtParent(Multilist *l, int idParent, DataParent data){
4
5     AddressParent dataBaru = alokasiParent(data);
6
7     if(!isEmpty(*l)){
8
9         AddressParent temp = findParent(*l, idParent);
10
11        if(temp != NULL){
12
13            dataBaru->next = temp->next;
14            temp->next = dataBaru;
15
16        }
17
18    }
19
20 }
21
```

InsertAtParent akan menginputkan **NodeParent** setelah **Node Bersangkutan.**

Konsepnya sama seperti insertAfter pada Linklist 2. Tetapi jika teman-teman perhatikan, cara yang saya gunakan sedikit berbeda.

Disini kita menggunakan bantuan *FindParent* untuk memudahkan kita mencari **NodeParent** yang akan disisipkan **NodeBaru**.

Apabila **ditemukan** (temp != NULL, ***Line 11***). Barulah kita akan insert Node yang telah disiapkan.

9. Delete First Parent

```
1
2 void deleteFirstParent(Multilist *l){
3
4     AddressParent temp = l->firstParent;
5
6     if(!isEmpty(*l)){
7         deleteAllChild(temp);
8
9         l->firstParent = l->firstParent->next;
10        free(temp);
11    }
12
13 }
14
```

deleteFirstParent. Tidak ada perbedaan dengan konsep Linklist.

Tambahan yang ada berada di ***Line 7***. Yakni penambahan prosedur *deleteAllChild*. deleteAllChild akan dibahas di bawah.

Intinya adalah ketika kita ingin men-delete **NodeParent**. Maka kita harus mendelete dulu semua **Child** di bawahnya, agar **Child** tersebut tidak "tersesat" di memori.

10. Delete Last Parent

```
1
2 void deleteLastParent(Multilist *l){
3
4     AddressParent temp = l->firstParent;
5
6     if(!isEmpty(*l)){
7
8         if(temp->next == NULL){
9             deleteFirstParent(l);
10        }else{
11
12            while(temp->next->next != NULL){
13                temp = temp->next;
14            }
15
16            deleteAllChild(temp->next);
17            free(temp->next);
18
19            temp->next = NULL;
20        }
21
22    }
23
24 }
25
```

deleteLastParent. Juga tidak ada perbedaan dengan konsep Linklist.

Tambahan yang ada berada di ***Line 16***. Yakni penambahan prosedur *deleteAllChild*.

Konsepnya sama, apabila hanya ada 1 **Parent**, maka *deleteFirstParent*. Apabila lebih dari 1 **Parent**, maka carilah **Parent** ke-2 terakhir lalu delete **Node Terakhir**.

Jangan lupa kita harus menghapus semua **Child** terlebih dahulu barulah kita *free*(**Node**) bersangkutan.

Urutannya **jangan sampai terbalik**. Apabila urutannya terbalik, maka *DeleteAllChild* tidak bisa digunakan karena Node sudah menjadi **NULL** (***Line 16 & 17***)

11. Delete At Parent

```
1
2  void deleteAtParent(Multilist *l, int idParent){
3
4      AddressParent temp = l->firstParent;
5      AddressParent hapus;
6
7      if(!isEmpty(*l)){
8
9          if(temp->dataParent.idTim == idParent){
10             deleteFirstParent(l);
11         }else{
12
13             while(temp->next != NULL){
14
15                 if(temp->next->dataParent.idTim == idParent){
16                     hapus = temp->next;
17                     temp->next = temp->next->next;
18
19                     deleteAllChild(hapus);
20                     free(hapus);
21                     break;
22                 }
23
24                 temp = temp->next;
25             }
26         }
27     }
28 }
29
```

deleteAtParent memiliki konsep yang mirip seperti deleteAt pada Linklist. tetapi code ini mungkin terlihat lebih kompleks, padahal tidak.

Perbedaan pada segi code adalah deleteAtParent disini tetap men-*passing* parameter **Multilist** dan kita akan menggunakan **idParent** untuk mencari **Node** yang akan dihapus.

Line 9 memeriksa apakah **Node pertama** adalah **Node** yang ingin kita **hapus**. Apabila **iya**, maka *deleteFirstParent*

Line 13 digunakan untuk *traversing* ke seluruh **Node** dan memastikan tidak ada **infinite Loop** semisal node yang dicari tidak ditemukan.

Line 15 digunakan untuk memeriksa apakah **Node** tersebut yang ingin kita hapus atau tidak. Perlu diingat bahwa untuk men-delete **Node** bersangkutan. kita harus mencari **Node Sebelum**, itulah mengapa kita menggunakan **temp->next**

12. Delete All Child

```
1
2  void deleteAllChild(AddressParent parent){
3
4      AddressChild temp;
5
6      while(haveChild(parent)){
7
8          temp = parent->firstChild;
9          parent->firstChild = parent->firstChild->next;
10
11         free(temp);
12     }
13
14 }
15
```

deleteAllChild memiliki tugas untuk manghapus semua **Child** yang dimiliki oleh **Parent** tertentu.

Line 6 menjadi kunci utama-nya***, Line 6*** memastikan bahwa ketika **Parent** masih memiliki **Child**, maka kita akan men-*delete* **Child** tersebut (Line 8-11).

13. Print Parent (***tambahan***)

```
1
2 void printParent(AddressParent parent){
3
4     printf("\n\t=== Parent ===");
5     printf("\n\t[-] Id Tim   : %d", parent->dataParent.idTim);
6     printf("\n\t[-] Nama Tim : %s", parent->dataParent.namaTim);
7
8 }
9
```

printParent akan mem-print semua **DataParent** dari **NodeParent** tertentu. Kebetulan pada contoh kita hanya memiliki 2 buah atribut, yakni **idTim** dan **namaTim**

printParent akan berguna apabila kita harus terus menerus print **DataParent**. Sehingga kita tinggal memanggil prosedurnya saja dan *wooossh*…

14. Print All Parent (***tambahan***)

```
1
2 void printAllParent(Multilist l){
3
4     AddressParent temp = l.firstParent;
5
6     while(temp != NULL){
7         printParent(temp);
8
9         printf("\n");
10        temp = temp->next;
11    }
12 }
13
```

printAllParent ditugaskan untuk mem-print semua **Parent** yang ada tanpa mem-*print* **Child**

Dapat terlihat kita menggunakan *printParent* untuk mem-*print* **DataParent** yang diperlukan

15. Print All

```
1
2 void printAll(Multilist l){
3
4     AddressParent temp = l.firstParent;
5
6     while(temp != NULL){
7         printParent(temp);
8
9         printAllChild(temp);
10        printf("\n");
11
12        temp = temp->next;
13    }
14
15 }
16
```

printAll merupakan prosedur untuk mem-*print* semua **Node** yang ada, baik dari **Parent** hingga **Child**.

Disini kita menggunakan prosedur *printParent* untuk mem-*print* **DataParent** dan *printAllChild* untuk mem-*print* keseluruhan **Child** yang dimilki oleh **Node bersangkutan**

C. Child

1. Alokasi Child

```
AddressChild alokasiChild(DataChild data){

    AddressChild ac;

    ac = (AddressChild) malloc(sizeof(NodeChild));

    ac->next = NULL;
    ac->dataChild = data;

    return ac;
}
```

Sama aja ya gaess.. :)

Membuat sebuah **NodeChild** yang dasarnya merupakan Linklist. mirip kayak Modul Linklist

2. Insert First Child

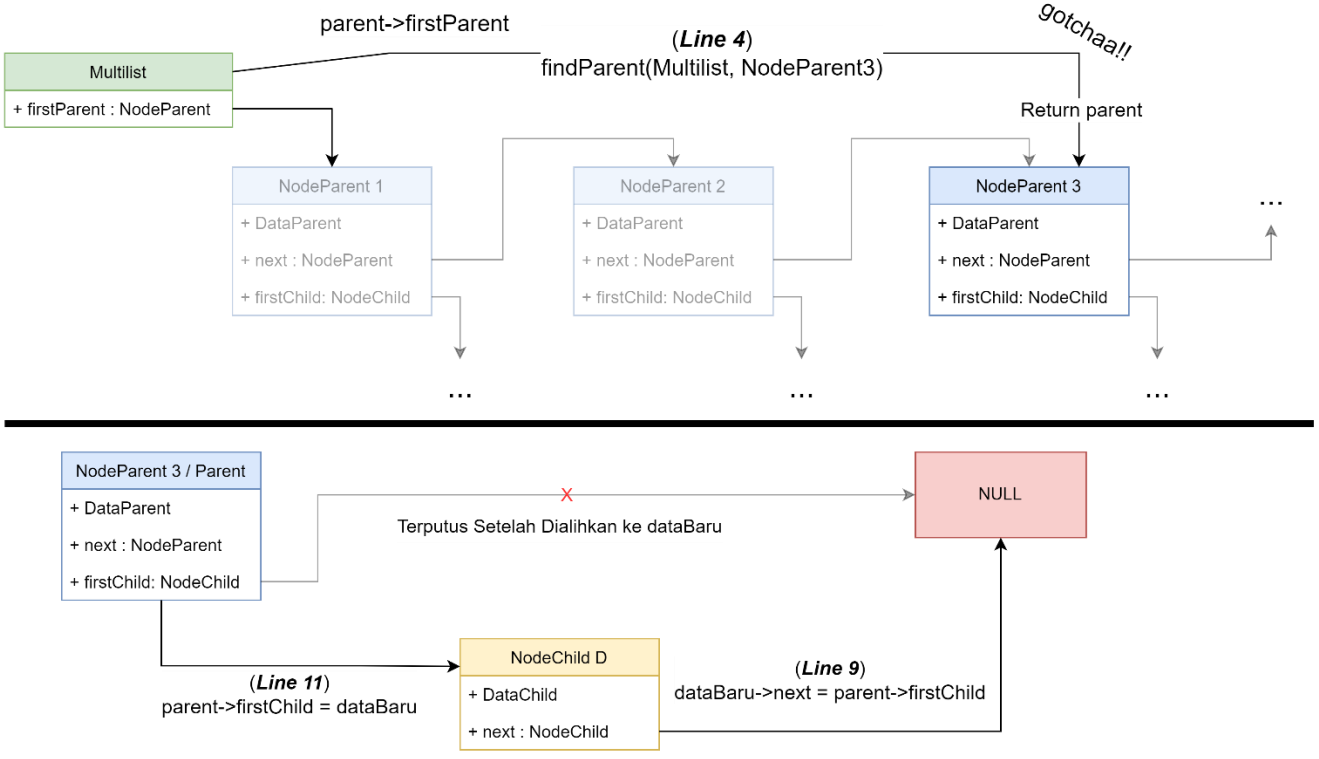
```
void insertFirstChild(Multilist l, int idParent, DataChild data){

    AddressParent parent = findParent(l, idParent);

    if(parent != NULL){

        AddressChild dataBaru = alokasiChild(data);
        dataBaru->next = parent->firstChild;

        parent->firstChild = dataBaru;

    }
}
```

InsertFirstChild memerlukan langkah spesial…

insertFirstChild harus mengetahui terlebih dahulu di **Parent** mana dia akan disimpan. Oleh karena itu, kita harus men-*passing* **Multilist** kita dan **idParent** dimana dia akan disimpan.

Setelah itu, kita menggunakan bantuan *findParent* untuk mendapatkan **Parent** mana dia akan disimpan (***Line 4***).

Lalu kita insert **Node Baru**-nya seperti biasa terhadap **NodeParent** yang sudah didapatkan.



3. Insert Last Child

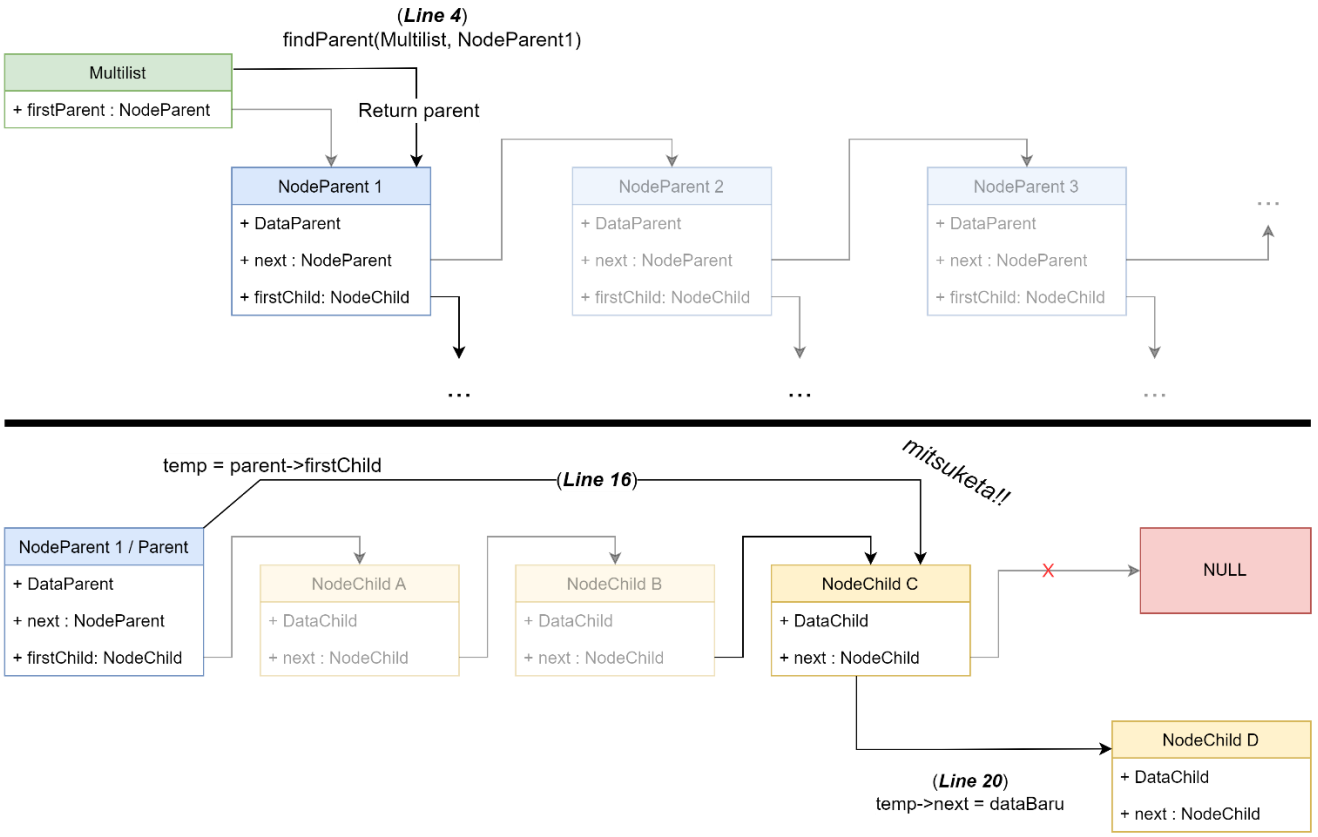
```
1
2  void insertLastChild(Multilist l, int idParent, DataChild data){
3
4      AddressParent parent = findParent(l, idParent);
5
6      if(parent != NULL){
7
8          if(!haveChild(parent)){
9              insertFirstChild(l, idParent, data);
10
11         }else{
12
13             AddressChild temp = parent->firstChild;
14             AddressChild dataBaru = alokasiChild(data);
15
16             while(temp->next != NULL){
17                 temp = temp->next;
18             }
19
20             temp->next = dataBaru;
21         }
22
23     }
24
25 }
26
```

Sama hal-nya seperti *insertFirstChild*. insertLastChild juga perlu mengetahui di **Parent** mana dia akan disimpan.

Setelah mengetahui di **Parent** mana dia akan disimpan. Seperti biasa kita perlu mencari **Node Terakhir** lalu men-*insert* **Node Baru** tersebut (***Line 13-20***)

Jangan lupa untuk men-*set* nilai dari **AddressChild bantu** dengan **Parent->firstChild**.

Hal ini dikarenakan **firstChild** merupakan **NodeChild Pertama** yang dipegang oleh **Parent** bersangkutan (***Line 13***)



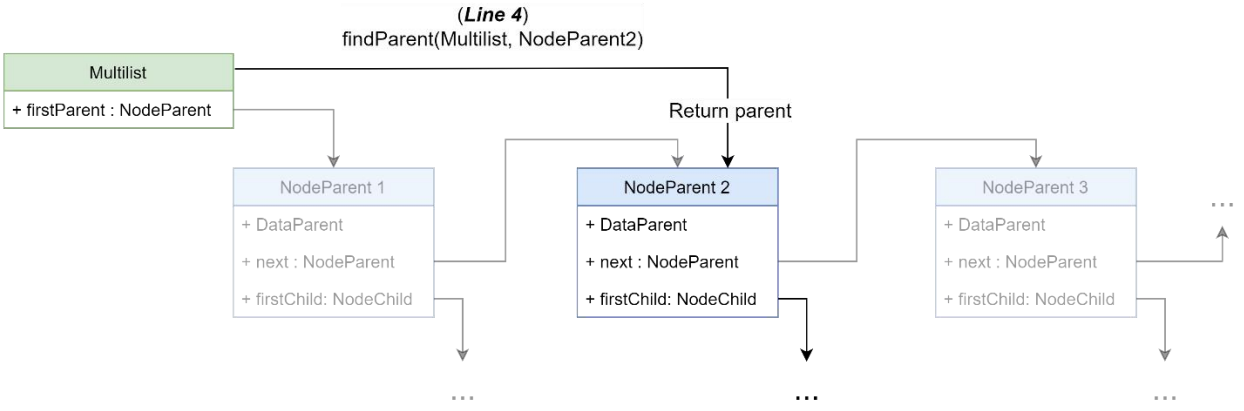
4. Delete First Child

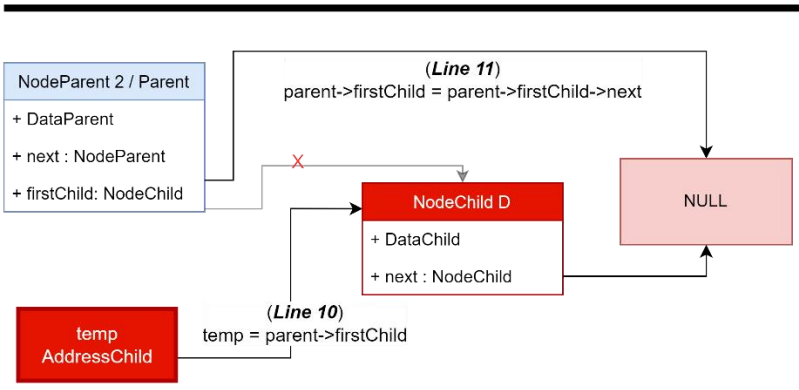
```
1
2  void deleteFirstChild(Multilist l, int idParent){
3
4      AddressParent parent = findParent(l, idParent);
5
6      if(parent != NULL){
7
8          if(haveChild(parent)){
9
10             AddressChild temp = parent->firstChild;
11             parent->firstChild = parent->firstChild->next;
12
13             free(temp);
14         }
15
16     }
17
18 }
19
```

Tetap dengan hal yang sama. Kita akan mencari terlebih dahulu **NodeParent** yang ingin kita hapus **FirstChild**-nya.

Kita juga harus memeriksa terlebih dahulu apakah **Parent** bersangkutan memiliki **Child** atau tidak. Gunakanlah *HaveChild* untuk mengetahui apakah **Parent** memiliki **Child** atau tidak (Line 8)

Setelah itu, hapuslah **Node** seperti biasa.





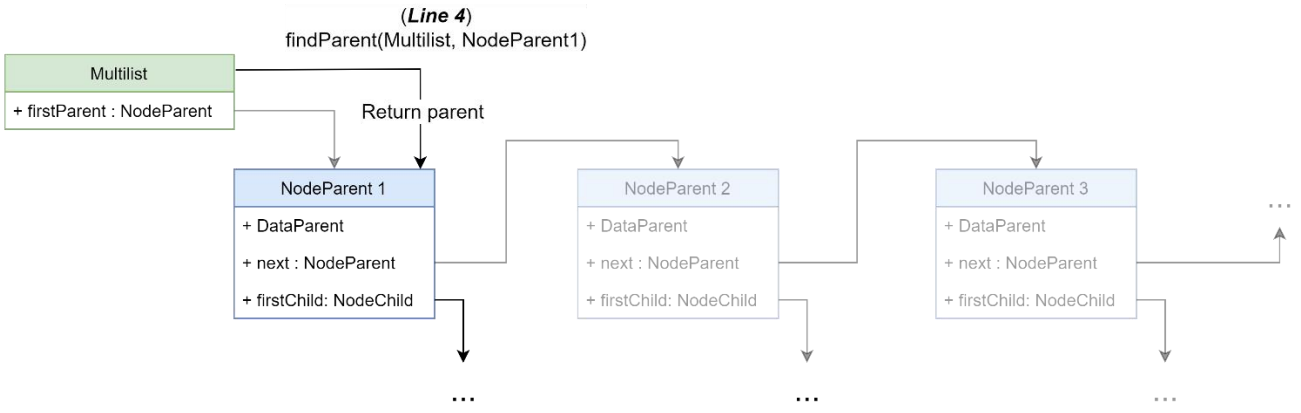
5. Delete Last Child

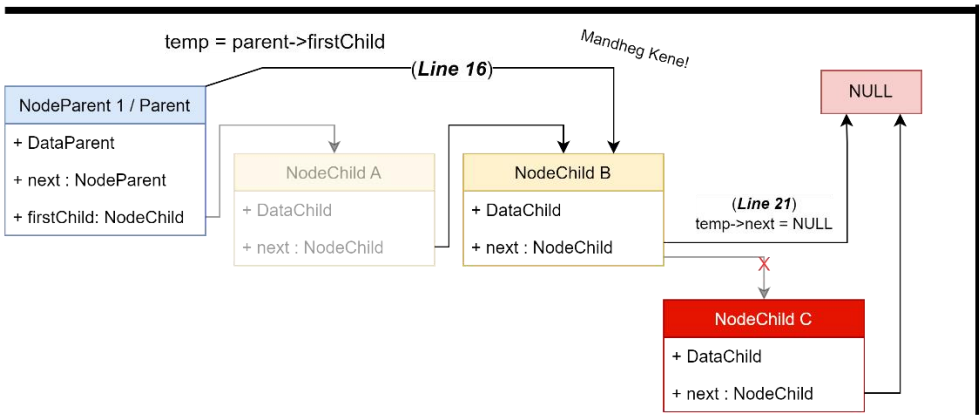
```
1
2  void deleteLastChild(Multilist l, int idParent){
3
4      AddressParent parent = findParent(l, idParent);
5
6      if(parent != NULL){
7
8          if(haveChild(parent)){
9
10             if(parent->firstChild->next == NULL){
11                 deleteFirstChild(l, idParent);
12             }else{
13
14                 AddressChild temp = parent->firstChild;
15
16                 while(temp->next->next != NULL){
17                     temp = temp->next;
18                 }
19
20                 free(temp->next);
21                 temp->next = NULL;
22             }
23         }
24     }
25 }
26
```

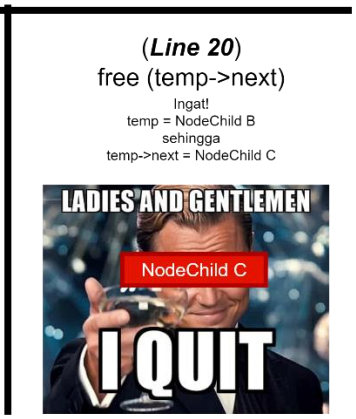
Langkah-langkahnya tidak berubah :)
Cari **Parent** mana yang ingin dihapus **Child**-nya. lalu gunakanlah **Parent**->**firstChild** sebagai **Node Acuan**

Kunci utamanya tetap di mencari **Parent**. Periksa dahulu apakah **Parent** punya **Child**? (***Line 8***)

Lalu apakah **Child**-nya hanya 1?
jika **iya**, maka *deleteFirst* (***Line 10-12***).
Jika **Tidak**, barulah cari **NodeChild** **ke-2 Terakhir**. Lalu Hapuslah **Node Terakhirnya**.







6. Print Child (***tambahan***)

```
1
2 void printChild(AddressChild child){
3
4     printf("\n\n\t\t=== Child ===");
5     printf("\n\t\t[*] Nama  : %s", child->dataChild.nama);
6     printf("\n\t\t[*] Umur  : %d", child->dataChild.umur);
7
8 }
9
```

Fungsi tambahan yang memudahkan kita untuk mem-*print* **DataChild** bersangkutan.
Sama seperti *printParent*, printChild akan sangat berguna apabila kita sering mem-*print* **DataChild**

Perhatikan bahwa parameternya menggunakan **AddressChild**, dan bukan **AddressParent**.

7. Print All Child

```
1
2 void printAllChild(AddressParent parent){
3
4     AddressChild temp = parent->firstChild;
5
6     while(temp != NULL){
7
8         printChild(temp);
9
10        temp = temp->next;
11    }
12 }
13
```

Mirip seperti *PrintAllParent*. Perbedaanya adalah kita akan mem-*print* semua **Child** yang dimiliki oleh **Parent** **Tertentu.**

Itu mengapa parameternya berupa **AddressParent**.

Kita hanya perlu *traversing* ke tiap **NodeChild** lalu gunakan prosedur *printChild* untuk mem-*print* **DataChild**.

GUIDED

Pada Guided, saya menggunakan 3 File Source. Yakni : sourceParent.c ; sourceChild.c ; dan source.c. tetapi apabila teman-teman hanya ingin menggunakan 1 file source, Maka itu juga diperbolehkan.

Tujuan saya menggunakan 3 file source ialah tidak lain untuk memudahkan mencari fungsi-fungsi yang telah dibuat. Karena Multilist menggunakan banyak sekali fungsi untuk Parent, Child, dan fungsi-fungsi tambahan lainnya untuk keperluan program yang akan dibuat.

Silahkan dibuat senyamannya :)

header.h

```
1.  #ifndef NORMAL_HEADER
2.  #define NORMAL_HEADER
3.
4.  #include <stdio.h>
5.  #include <stdlib.h>
6.  #include <stdbool.h>
7.  #include <string.h>
8.
9.  typedef char string[50];
10. typedef struct Child* AddressChild;
11. typedef struct Parent* AddressParent;
12.
13. //data parent dan data child
14. typedef struct DataChild{
15.
16.   string nama;
17.   int umur;
18.
19. }DataChild;
20.
21.
22. typedef struct DataParent{
23.
24.   int idTim;
25.   string namaTim;
26.
27. }DataParent;
28.
29.
30. typedef struct Child{
31.
32.   DataChild dataChild;
33.   AddressChild next;
34.
35. }NodeChild;
36.
37.
38. typedef struct Parent{
39.
40.   DataParent dataParent;
41.   AddressParent next;
42.
43.   AddressChild firstChild;
44.
45. }NodeParent;
46.
47.
48. typedef struct{
49.
50.   AddressParent firstParent;
51.
52. }Multilist;
53.
```

```
55. //sourceParent;
56. void createEmpty(Multilist *l);
57. void insertFirstParent(Multilist *l, DataParent data);
58. void insertAtParent(Multilist *l, int idParent, DataParent data);
59. void insertLastParent(Multilist *l, DataParent data);
60.
61. void deleteFirstParent(Multilist *l);
62. void deleteAtParent(Multilist *l, int idParent);
63. void deleteLastParent(Multilist *l);
64.
65. void deleteAllChild(AddressParent parent);
66.
67. void printParent(AddressParent parent);
68. void printAll(Multilist l);
69. void printAllParent(Multilist l);
70.
71. AddressParent alokasiParent(DataParent data);
72. AddressParent findParent(Multilist l, int cariId);
73.
74. bool isEmpty(Multilist l);
75. bool haveChild(AddressParent ap);
76.
77.
78. //sourceChild;
79. AddressChild alokasiChild(DataChild data);
80.
81. void insertFirstChild(Multilist l, int idParent, DataChild data);
82. void insertLastChild(Multilist l, int idParent, DataChild data);
83.
84. void deleteFirstChild(Multilist l, int idParent);
85. void deleteLastChild(Multilist l, int idParent);
86.
87. void printChild(AddressChild child);
88. void printAllChild(AddressParent parent);
89.
90.
91. //Umum
92. DataParent makeDataParent(int id, string namaTim);
93. DataChild makeDataChild(string nama, int umur);
94. bool isUnik(Multilist l, int id);
95.
96.
97. #endif
98.
```

sourceParent.c

```
1.  #include "header.h"
2.
3.  void createEmpty(Multilist *l){
4.      l->firstParent = NULL;
5.  }
6.
7.
8.  bool isEmpty(Multilist l){
9.      return l.firstParent == NULL;
10. }
11.
12.
13. bool haveChild(AddressParent ap){
14.     return ap->firstChild != NULL;
15. }
16.
17.
18. AddressParent alokasiParent(DataParent data){
19.
20.     AddressParent ap;
21.
22.     ap = (AddressParent) malloc(sizeof(NodeParent));
23.
24.     ap->next = NULL;
25.     ap->firstChild = NULL;
26.     ap->dataParent = data;
27.
28.     return ap;
29. }
30.
31. AddressParent findParent(Multilist l, int cariId){
32.
33.     AddressParent parentBantu = NULL;
34.
35.     parentBantu = l.firstParent;
36.
37.     while(parentBantu != NULL){
38.         if(parentBantu->dataParent.idTim == cariId){
39.             return parentBantu;
40.         }
41.
42.         parentBantu = parentBantu->next;
43.     }
44.
45.     return NULL;
46. }
47.
48.
49.
50. void insertFirstParent(Multilist *l, DataParent data){
51.
52.     AddressParent dataParent = alokasiParent(data);
53.
54.     dataParent->next = l->firstParent;
55.     l->firstParent = dataParent;
56. }
57.
58.
```

```
1.  //sebenarnya lebih tepat disebut insertAfter
2.  void insertAtParent(Multilist *l, int idParent, DataParent data){
3.
4.      AddressParent dataBaru = alokasiParent(data);
5.
6.      if(!isEmpty(*l)){
7.
8.          AddressParent temp = findParent(*l, idParent);
9.
10.         if(temp != NULL){
11.             dataBaru->next = temp->next;
12.             temp->next = dataBaru;
13.         }
14.     }
15. }
16.
17.
18. void insertLastParent(Multilist *l, DataParent data){
19.
20.     AddressParent dataParent = alokasiParent(data);
21.
22.     if(isEmpty(*l)){
23.         insertFirstParent(l, data);
24.     }else{
25.
26.         AddressParent temp = l->firstParent;
27.
28.         while(temp->next != NULL){
29.             temp = temp->next;
30.         }
31.         temp->next = dataParent;
32.     }
33. }
34.
35.
36. void deleteFirstParent(Multilist *l){
37.
38.     AddressParent temp = l->firstParent;
39.
40.     if(!isEmpty(*l)){
41.         deleteAllChild(temp);
42.
43.         l->firstParent = l->firstParent->next;
44.         free(temp);
45.     }
46.
47. }
48.
```

```
1.  void deleteAtParent(Multilist *l, int idParent){
2.
3.      AddressParent temp = l->firstParent;
4.      AddressParent hapus;
5.
6.      if(!isEmpty(*l)){
7.
8.          if(temp->dataParent.idTim == idParent){
9.              deleteFirstParent(l);
10.         }else{
11.
12.             while(temp->next != NULL){
13.
14.                 if(temp->next->dataParent.idTim == idParent){
15.                     hapus = temp->next;
16.                     temp->next = temp->next->next;
17.
18.                     deleteAllChild(hapus);
19.                     free(hapus);
20.                     break;
21.                 }
22.
23.                 temp = temp->next;
24.             }
25.         }
26.     }
27. }
28.
29.
30. void deleteLastParent(Multilist *l){
31.
32.     AddressParent temp = l->firstParent;
33.
34.     if(!isEmpty(*l)){
35.
36.         if(temp->next == NULL){
37.             deleteFirstParent(l);
38.         }else{
39.
40.             while(temp->next->next != NULL){
41.                 temp = temp->next;
42.             }
43.
44.             deleteAllChild(temp->next);
45.             free(temp->next);
46.
47.             temp->next = NULL;
48.         }
49.
50.     }
51.
52. }
53.
```

```
1.  void deleteAllChild(AddressParent parent){
2.
3.      AddressChild temp;
4.
5.      while(haveChild(parent)){
6.
7.          temp = parent->firstChild;
8.          parent->firstChild = parent->firstChild->next;
9.
10.         free(temp);
11.     }
12. }
13.
14.
15. void printParent(AddressParent parent){
16.
17.     printf("\n\t=== Parent ===");
18.     printf("\n\t[-] Id Tim   : %d", parent->dataParent.idTim);
19.     printf("\n\t[-] Nama Tim : %s", parent->dataParent.namaTim);
20.
21. }
22.
23. void printAllParent(Multilist l){
24.
25.     AddressParent temp = l.firstParent;
26.
27.     while(temp != NULL){
28.         printParent(temp);
29.
30.         printf("\n");
31.         temp = temp->next;
32.     }
33. }
34.
35.
36. void printAll(Multilist l){
37.
38.     AddressParent temp = l.firstParent;
39.
40.     while(temp != NULL){
41.         printParent(temp);
42.
43.         printAllChild(temp);
44.         printf("\n");
45.
46.         temp = temp->next;
47.     }
48.
49. }
50.
```

sourceChild.c

```c
#include "header.h"

AddressChild alokasiChild(DataChild data){

    AddressChild ac;
    ac = (AddressChild) malloc(sizeof(NodeChild));

    ac->next = NULL;
    ac->dataChild = data;

    return ac;
}


void insertFirstChild(Multilist l, int idParent, DataChild data){

    AddressParent parent = findParent(l, idParent);

    if(parent != NULL){
        AddressChild dataBaru = alokasiChild(data);
        dataBaru->next = parent->firstChild;

        parent->firstChild = dataBaru;
    }
}


void insertLastChild(Multilist l, int idParent, DataChild data){

    AddressParent parent = findParent(l, idParent);

    if(parent != NULL){

        if(!haveChild(parent)){
            insertFirstChild(l, idParent, data);
        }else{

            AddressChild temp = parent->firstChild;
            AddressChild dataBaru = alokasiChild(data);

            while(temp->next != NULL){
                temp = temp->next;
            }

            temp->next = dataBaru;
        }
    }
}


void deleteFirstChild(Multilist l, int idParent){

    AddressParent parent = findParent(l, idParent);

    if(parent != NULL){

        if(haveChild(parent)){

            AddressChild temp = parent->firstChild;
            parent->firstChild = parent->firstChild->next;

            free(temp);
        }
    }
}

```

```
1.
2.  void deleteLastChild(Multilist l, int idParent){
3.
4.      AddressParent parent = findParent(l, idParent);
5.
6.      if(parent != NULL){
7.
8.          if(haveChild(parent)){
9.
10.             if(parent->firstChild->next == NULL){
11.                 deleteFirstChild(l, idParent);
12.             }else{
13.
14.                 AddressChild temp = parent->firstChild;
15.
16.                 while(temp->next->next != NULL){
17.                     temp = temp->next;
18.                 }
19.
20.                 free(temp->next);
21.                 temp->next = NULL;
22.             }
23.         }
24.     }
25. }
26.
27.
28. void printChild(AddressChild child){
29.
30.     printf("\n\n\t\t=== Child ===");
31.     printf("\n\t\t[*] Nama  : %s", child->dataChild.nama);
32.     printf("\n\t\t[*] Umur  : %d", child->dataChild.umur);
33.
34. }
35.
36.
37. void printAllChild(AddressParent parent){
38.
39.     AddressChild temp = parent->firstChild;
40.
41.     while(temp != NULL){
42.
43.         printChild(temp);
44.
45.         temp = temp->next;
46.     }
47. }
48.
```

source.c

```
#include "header.h"

DataParent makeDataParent(int id, string namaTim){

    DataParent data;

    data.idTim = id;
    strcpy(data.namaTim, namaTim);

    return data;
}


DataChild makeDataChild(string nama, int umur){

    DataChild data;

    data.umur = umur;
    strcpy(data.nama, nama);

    return data;
}

bool isUnik(Multilist l, int id){

    AddressParent temp= l.firstParent;

    while(temp != NULL){
        if(temp->dataParent.idTim == id){
            return false;
        }
        temp = temp->next;
    }
    return true;
}

```

main.c

```
#include "header.h"

int main() {

    Multilist l;
    AddressParent tempParent;
    AddressChild tempChild;
    createEmpty(&l);

    string konfirmasi;

    int idParent;

    int menu;
    int id;
    int umur;
    string nama;

    int urut = 1;
    do{
        system("cls");

        printf("\n\t=== GUIDED ===");
        printf("\n\n\t=== Menu Parent ===");
        printf("\n[%d]\tInsert First Parent" , urut++);
        printf("\n[%d]\tInsert After Parent" , urut++);
        printf("\n[%d]\tInsert Last Parent"  , urut++);
        printf("\n[%d]\tDelete First Parent" , urut++);
        printf("\n[%d]\tDelete At Parent"    , urut++);
        printf("\n[%d]\tDelete Last Parent"  , urut++);
        printf("\n[%d]\tFind Parent"         , urut++);
        printf("\n[%d]\tShow Parent"         , urut++);

        printf("\n\n\t=== Menu Child ===");
        printf("\n[%d]\tInsert First Child" , urut++);
        printf("\n[%d]\tInsert Last Child"  , urut++);
        printf("\n[%d]\tDelete First Child" , urut++);
        printf("\n[%d]\tDelete Last Child"  , urut++);
        printf("\n[%d]\tShow Child"         , urut++);

        printf("\n\n\t=== Show All ===");
        printf("\n[%d]\tShow All" , urut++);
        printf("\n[0]\tExit");

        urut = 1;

        printf("\n\n>>>>> "); scanf("%d", &menu);


```

```
1.            switch(menu){
2.
3.                //insert first parent
4.                case 1 :
5.                    printf("\n\t\t--- Input First Parent ---\n");
6.                    do{
7.                        printf("\t\tInput Id Tim   : "); scanf("%d", &id);
8.
9.                        if(isUnik(l, id)){
10.                           break;
11.                       }else{
12.                           printf("\t\t\t[!] ID Harus Unik\n\n");
13.                       }
14.                   }while(true);
15.
16.                   do{
17.                       printf("\t\tInput Nama Tim : "); fflush(stdin); gets(nama);
18.
19.                       if(strlen(nama) != 0){
20.                           break;
21.                       }else{
22.                           printf("\t\t\t[!] Nama Tidak Boleh Kosong\n\n");
23.                       }
24.                   }while(true);
25.
26.                   insertFirstParent(&l, makeDataParent(id, nama));
27.                   printf("\n\t\t\t[*] Input First Berhasil");
28.               break;   //break case 1
29.
30.               case 2 :
31.                   printf("\n\t\t--- Input At Parent --- \n");
32.
33.                   if(!isEmpty(l)){
34.                       printf("\n\t Input Id Parent : "); scanf("%d", &idParent);
35.
36.                       tempParent = findParent(l, idParent);
37.
38.                       if(tempParent == NULL){
39.                           printf("\n\t\t[!] Parent Tidak Ditemukan!");
40.                       }else{
41.                           printf("\n");
42.                           do{
43.                               printf("\t\tInput Id Tim   : "); scanf("%d", &id);
44.
45.                               if(isUnik(l, id)){
46.                                   break;
47.                               }else{
48.                                   printf("\t\t\t[!] ID Harus Unik\n\n");
49.                               }
50.                           }while(true);
51.
52.                           do{
53.                               printf("\t\tInput Nama Tim : "); fflush(stdin); gets(nama);
54.
55.                               if(strlen(nama) != 0){
56.                                   break;
57.                               }else{
58.                                   printf("\t\t\t[!] Nama Tidak Boleh Kosong\n\n");
59.                               }
60.                           }while(true);
61.
62.                           insertAtParent(&l, idParent, makeDataParent(id, nama));
63.                           printf("\n\t\t[*] Berhasil Insert At!");
64.                       }
65.
66.                   }else{
67.                       printf("\n\t\t[!] List Masih Kosong!");
68.                   }
69.
70.               break; //break case 2
71.
```

```
                //input last parent
                case 3 :

                    printf("\n\t\t--- Input Last Parent ---\n");
                    do{
                        printf("\t\tInput Id Tim   : "); scanf("%d", &id);

                        if(isUnik(l, id)){
                            break;
                        }else{
                            printf("\t\t\t[!] ID Harus Unik\n\n");
                        }
                    }while(true);

                    do{
                        printf("\t\tInput Nama Tim : "); fflush(stdin); gets(nama);

                        if(strlen(nama) != 0){
                            break;
                        }else{
                            printf("\t\t\t[!] Nama Tidak Boleh Kosong\n\n");
                        }
                    }while(true);

                    insertLastParent(&l, makeDataParent(id, nama));

                    printf("\n\t\t\t[*] Input Last Berhasil");
                break;   //break case 3



                case 4 :
                    printf("\n\t\t=== Delete First Parent ===");

                    if(!isEmpty(l)){

                        printParent(l.firstParent);
                        printAllChild(l.firstParent);

                        printf("\n\n\tApakah Yakin ingin Menghapus Parent? [Y/N] : "); fflush(stdin);
    gets(konfirmasi);

                        if(!strcmpi(konfirmasi, "y")){
                            deleteFirstParent(&l);
                            printf("\n\t\t\t[*] Delete First Berhasil");
                        }else if(!strcmpi(konfirmasi, "n")){
                            printf("\n\t\t\t[!] Delete First Dibatalkan");
                        }else{
                            printf("\n\t\t\t[!] Inputan Invalid");
                        }
                    }else{
                        printf("\n\t\t[!] List Masih Kosong");
                    }

                break;   //break case 4;
```

```
1.                case 5 :
2.                     printf("\n\t\t=== Delete At Parent ===");
3.
4.                     if(!isEmpty(l)){
5.
6.                          printf("\n\n\tInput Id Parent : "); scanf("%d", &idParent);
7.
8.                          tempParent = findParent(l, idParent);
9.
10.                         if(tempParent == NULL){
11.                              printf("\n\t\t[!] Parent Tidak Ditemukan!");
12.                         }else{
13.
14.                              printParent(tempParent);
15.                              printAllChild(tempParent);
16.
17.                              printf("\n\n\tApakah Yakin ingin Menghapus Parent? [Y/N] : "); fflush(stdin);
    gets(konfirmasi);
18.
19.                              if(!strcmpi(konfirmasi, "y")){
20.                                   deleteAtParent(&l, idParent);
21.                                   printf("\n\t\t\t[*] Delete At Berhasil");
22.                              }else if(!strcmpi(konfirmasi, "n")){
23.                                   printf("\n\t\t\t[!] Delete At Dibatalkan");
24.                              }else{
25.                                   printf("\n\t\t\t[!] Inputan Invalid");
26.                              }
27.
28.                         }
29.                    }else{
30.                         printf("\n\t\t[!] List Masih Kosong!");
31.                    }
32.               break;   //break case 5
33.
34.
35.               case 6 :
36.                    printf("\n\t\t=== Delete Last Parent ===");
37.
38.                    if(!isEmpty(l)){
39.
40.                         tempParent = l.firstParent;
41.
42.                         if(tempParent->next != NULL){
43.                              while(tempParent->next != NULL){
44.                                   tempParent = tempParent->next;
45.                              }
46.                         }
47.
48.                         printParent(tempParent);
49.                         printAllChild(tempParent);
50.
51.                         printf("\n\n\tApakah Yakin ingin Menghapus Parent? [Y/N] : "); fflush(stdin);
    gets(konfirmasi);
52.
53.                         if(!strcmpi(konfirmasi, "y")){
54.                              deleteLastParent(&l);
55.                              printf("\n\t\t\t[*] Delete Last Berhasil");
56.                         }else if(!strcmpi(konfirmasi, "n")){
57.                              printf("\n\t\t\t[!] Delete Last Dibatalkan");
58.                         }else{
59.                              printf("\n\t\t\t[!] Inputan Invalid");
60.                         }
61.                    }else{
62.                         printf("\n\t\t[!] List Masih Kosong");
63.                    }
64.
65.               break; //break case 6
66.
```

```
                case 7 :
                    printf("\n\t\t=== Find Parent ===");
                    printf("\n\n\tInput Id Parent : "); scanf("%d", &idParent);

                    tempParent = findParent(l, idParent);

                    if(tempParent == NULL){
                        printf("\n\t\t\t[!] Parent Tidak Ditemukan!");
                    }else{
                        printParent(tempParent);
                        printAllChild(tempParent);
                    }

                break;   //break case 7;

                case 8 :
                    printf("\n\t\t=== Show Parent ===");
                    if(!isEmpty(l)){
                        printAllParent(l);
                    }else{
                        printf("\n\t\t[!] List Masih Kosong");
                    }

                break;   //break case 8

                case 9 :
                    printf("\n\t\t=== Input First Child ===");
                    printf("\n\tInput Id Tim : "); scanf("%d", &idParent);

                    tempParent = findParent(l, idParent);

                    if(tempParent != NULL){
                        printParent(tempParent);

                        printf("\n\t-------------------------------\n");
                        do{
                            printf("\n\tInput Nama : "); fflush(stdin); gets(nama);
                            if(strlen(nama) == 0){
                                printf("\t\t[!] Nama Tidak Boleh Kosong\n");
                            }else{
                                break;
                            }

                        }while(true);

                        do{
                            printf("\tInput Umur : "); scanf("%d", &umur);
                            if(umur <= 0){
                                printf("\n\t\t[!] Umur Tidak Boleh <= 0\n");
                            }else{
                                break;
                            }
                        }while(true);

                        insertFirstChild(l, idParent, makeDataChild(nama, umur));
                        printf("\n\t\t\t[*] Input First Child Berhasil");

                    }else{
                        printf("\n\t\t[!] Parent Tidak Ditemukan");
                    }

                break;   //break case 9;

```

```
                case 10 :
                    printf("\n\t\t=== Input Last Child ===");

                    printf("\n\tInput Id Tim : "); scanf("%d", &idParent);

                    tempParent = findParent(l, idParent);

                    if(tempParent != NULL){
                        printParent(tempParent);

                        printf("\n\t--------------------------------\n");

                        do{
                            printf("\n\tInput Nama : "); fflush(stdin); gets(nama);
                            if(strlen(nama) == 0){
                                printf("\t\t[!] Nama Tidak Boleh Kosong\n");
                            }else{
                                break;
                            }

                        }while(true);

                        do{
                            printf("\tInput Umur : "); scanf("%d", &umur);
                            if(umur <= 0){
                                printf("\n\t\t[!] Umur Tidak Boleh <= 0\n");
                            }else{
                                break;
                            }
                        }while(true);

                        insertLastChild(l, idParent, makeDataChild(nama, umur));

                        printf("\n\t\t\t[*] Input Last Child Berhasil");

                    }else{

                        printf("\n\t\t[!] Parent Tidak Ditemukan");

                    }

                break;   //break case 10;

```

```
1.            case 11 :
2.                 printf("\n\t\t=== Delete First Child ===");
3.                 printf("\n\tInput Id Tim : "); scanf("%d", &idParent);
4.
5.                 tempParent = findParent(l, idParent);
6.
7.                 if(tempParent != NULL){
8.                      printParent(tempParent);
9.
10.                     if(haveChild(tempParent)){
11.                          printChild(tempParent->firstChild);
12.                          printf("\n\tApakah Yakin Ingin Menghapus Child? [Y/N] : "); fflush(stdin);
    gets(konfirmasi);
13.
14.                          if(!strcmpi(konfirmasi, "y")){
15.                               printf("\n\t\t[*] Delete First Child Berhasil");
16.                               deleteFirstChild(l, idParent);
17.                          }else if(!strcmpi(konfirmasi, "n")){
18.                               printf("\n\t\t[*] Delete First Child Dibatalkan");
19.                          }else{
20.                               printf("\n\t\t[!] Inputan Invalid");
21.                          }
22.                     }else{
23.                          printf("\n\n\t\t[*] Parent Tidak Memiliki Child");
24.                     }
25.                     printf("\n\t--------------------------------\n");
26.                }else{
27.                     printf("\n\t\t[!] Parent Tidak Ditemukan");
28.                }
29.           break;    //break case 11
30.
31.           case 12 :
32.                printf("\n\t\t=== Delete Last Child ===");
33.                printf("\n\tInput Id Tim : "); scanf("%d", &idParent);
34.
35.                tempParent = findParent(l, idParent);
36.
37.                if(tempParent != NULL){
38.                     printParent(tempParent);
39.
40.                     if(haveChild(tempParent)){
41.
42.                          tempChild = tempParent->firstChild;
43.                          while(tempChild->next != NULL){
44.                               tempChild = tempChild->next;
45.                          }
46.
47.                          printChild(tempChild);
48.                          printf("\n\tApakah Yakin Ingin Menghapus Child? [Y/N] : "); fflush(stdin);
    gets(konfirmasi);
49.
50.                          if(!strcmpi(konfirmasi, "y")){
51.                               deleteLastChild(l, idParent);
52.                               printf("\n\t\t[*] Delete Last Child Berhasil");
53.                          }else if(!strcmpi(konfirmasi, "n")){
54.                               printf("\n\t\t[*] Delete Last Child Dibatalkan");
55.                          }else{
56.                               printf("\n\t\t[!] Inputan Invalid");
57.                          }
58.                     }else{
59.                          printf("\n\n\t\t[*] Parent Tidak Memiliki Child");
60.                     }
61.                     printf("\n\t--------------------------------\n");
62.                }else{
63.                     printf("\n\t\t[!] Parent Tidak Ditemukan");
64.                }
65.
66.           break;    //break case 12
67.
```

```
                case 13 :
                    printf("\n\t=== Show Child ===");

                    printf("\n\tInput Id Tim : "); scanf("%d", &idParent);

                    tempParent = findParent(l, idParent);

                    if(tempParent != NULL){
                        printAllChild(tempParent);
                    }else{
                        printf("\n\t\t[!] Parent Tidak Ditemukan");
                    }
                break;   //break case 13

                case 14 :

                    if(!isEmpty(l)){
                        printAll(l);
                    }else{
                        printf("\n\t\t[!] List Masih Kosong");
                    }

                break;   //break case 14;

                case 0 :
                    printf("\n\t\t[*] Keluar Menu");
                break;

                default :
                    printf("\n\t\t[!] Menu Tidak Ditemukan");
                break;   //break default;

            }

            getch();
        }while(menu != 0);

        return 0;
    }
```

<u>Ketentuan & Format Pengumpulan Guided:</u>

1. Untuk comment tidak menjadi masalah jika tidak ditulis di Guided
2. Ini udh benar-benar bisa di copas nih… jangan sampai compile error woy… :v
3. konsekuensi Guided yang tidak bisa di compile bisa berupa **pengurangan nilai** dan konsekuensi paling parah adalah **tidak boleh mengikuti** dan **mengerjakan** Pretest, UGD, Tugas, dan Laporan.
4. Teman-teman boleh berkreasi dengan Guided seperti mengganti nama variabel, membuat prosedur lain, mengganti logika program, dll. Dengan ketentuan :
    - Poin 2 dan 3 tetap harus dipatuhi yaa...
    - Fungsionalitas Program tetap terjaga (menu yang sudah ada jangan diubah-ubah)
    - Pastikan tidak ada error pada penambahan/perubahan code yang teman-teman lakukan ya, atau tetap akan dikenakan pengurangan nilai pada salah satu komponen penilaian :(
5. Saya memperbolehkan variasi **Multilist Rekursif** jika memang ada yang membuatnya. *(sebenarnya lebih mudah Multilist rekursif, tapi itu di modul selanjutnya, yakni Rekursif List)*
6. Jangan ragu-ragu untuk bertanya kepada saya ataupun asisten lain yoo… :)
7. Semua kode dimasukkan ke dalam sebuah folder dengan format penamaan: **GD8_X_YYYYY**
8. Folder tadi kemudian di Zip dengan format penamaan: **GD8_X_YYYYY.zip**
9. Keterangan:
    - X = Kelas
    - YYYYY = 5 Digit Terakhir NPM Praktikan

Yass benar, Guided bisa di copas langsung. Karena multilist memang banyak, jadi tentu ada keringan.

*Tetapi artinya UGD dan Tugas pasti bakalan **susah**... mohon dengan sangat dipelajari dengan baik yaa.. :v*

Hint UGD & Tugas

1. Tenang saja, tentu tidak semua akan keluar di UGD... palingan cuman beberapa menu dan tentunya bervariasi untuk tiap kelas.

2. Usahakan jangan mengubah apa-apa di fungsi pada saat UGD. Sebisa mungkin coba sesuaikan penamaan tipe data, fungsi, dll sesuai dengan Guided. Untuk meminimalkan perubahan pada Fungsi yang sudah stabil.

3. Usahakan mengerti code Guided dengan baik. Benar-benar usahakan dengan baik.. :v

4. Latihan CRUD seperti biasa, terutama cari data dan edit data karena belum ada di guided kan...

5. Jangan lupakan materi-materi sebelumnya yooo, ada yang saya ambil soale :)