LAPORAN PRAKTIKUM
STRUKTUR DATA

MODUL XIII

MULTI LINKED LIST

Disusun Oleh :
Muhammad Fathammubina
NIM : 103112430188

Dosen
FAHRUDIN MUKTI WIBOWO

PROGRAM STUDI STRUKTUR DATA
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY PURWOKERTO
2025

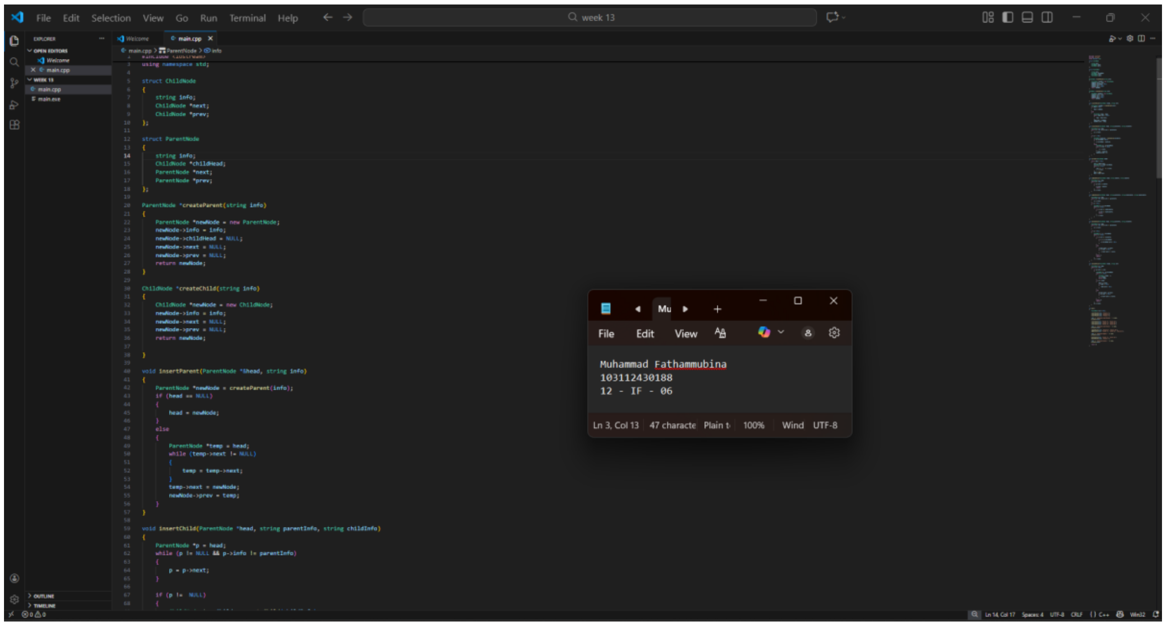
A. Dasar Teori

Multi Linked List adalah struktur data bertingkat yang menghubungkan satu daftar induk (parent) dengan banyak daftar anak (child). Parent dan child disimpan dalam node dinamis yang saling terhubung menggunakan pointer, sehingga membentuk relasi 1-to-many (satu parent punya banyak child). Pada program kamu, parent menggunakan doubly linked list (punya next dan prev) untuk navigasi dua arah, sedangkan child juga memakai doubly linked list yang diakses lewat childHead pada tiap parent. Operasi penting dalam ADT ini meliputi alokasi/dealokasi memori, insert parent/child, traversal print, update data, dan delete termasuk menghapus child otomatis saat parent dihapus. Struktur ini cocok untuk data hierarki seperti folder-file, induk-cabang organisasi, atau kategori-item, karena tiap level dapat ditelusuri dan dikelola tanpa mengganggu level lainnya.

B. Guided

Guided 1

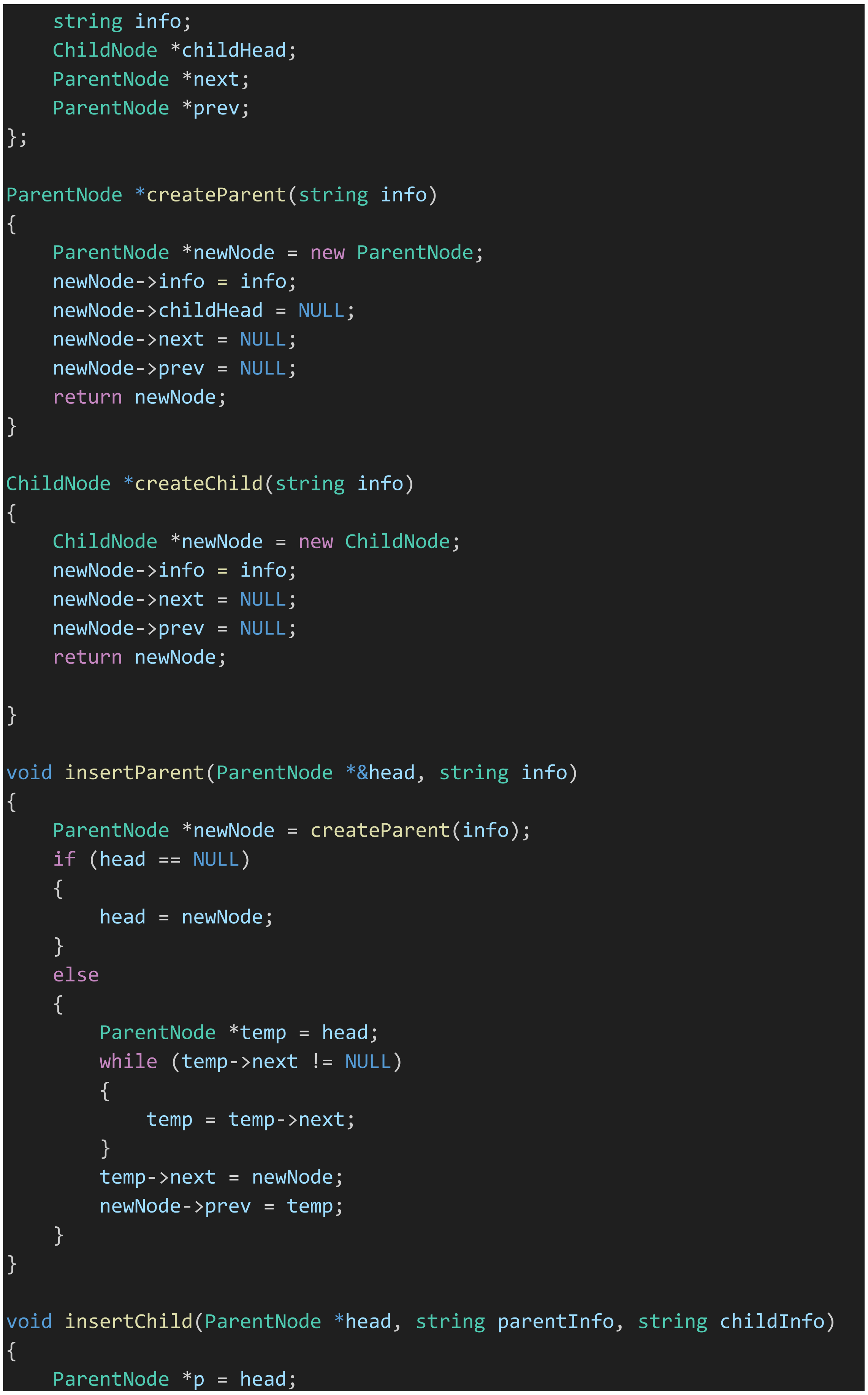
main.cpp


File Edit Selection View Go Run Terminal Help
week 13
Welcome
main.cpp
main.cpp > ParentNode > info

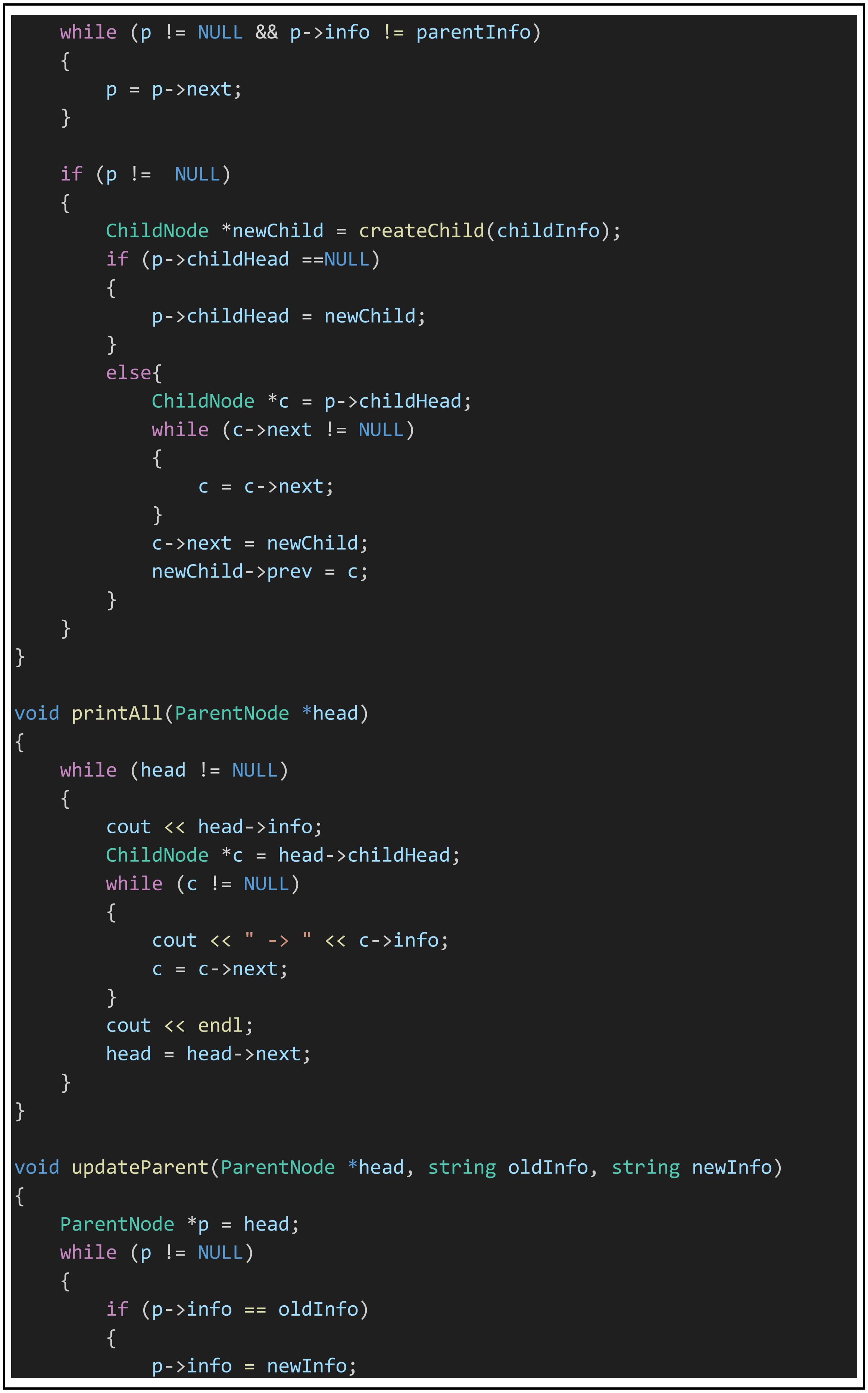
```cpp
using namespace std;

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

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

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

ChildNode *createChild(string info)
{
    ChildNode *newNode = new ChildNode;
    newNode->info = info;
    newNode->next = NULL;
    newNode->prev = NULL;
    return newNode;

}

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

void insertChild(ParentNode *head, string parentInfo, string childInfo)
{
    ParentNode *p = head;
    while (p != NULL && p->info != parentInfo)
    {
        p = p->next;
    }

    if (p !=  NULL)
    {
```

```
Muhammad Fathammubina
103112430188
12 - IF - 06
```

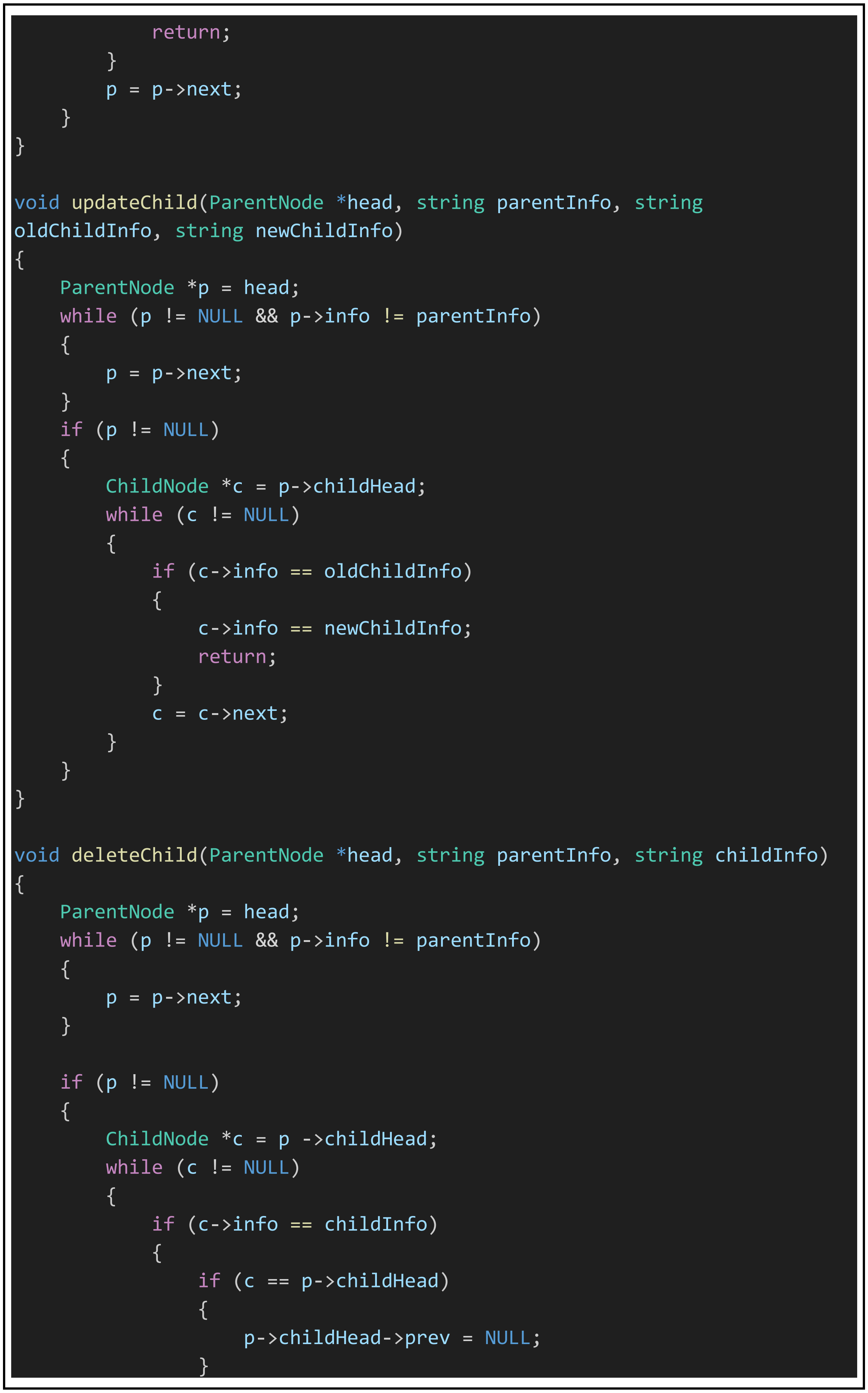
```cpp
#include <iostream>
#include <string>
using namespace std;

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

struct ParentNode
{
```



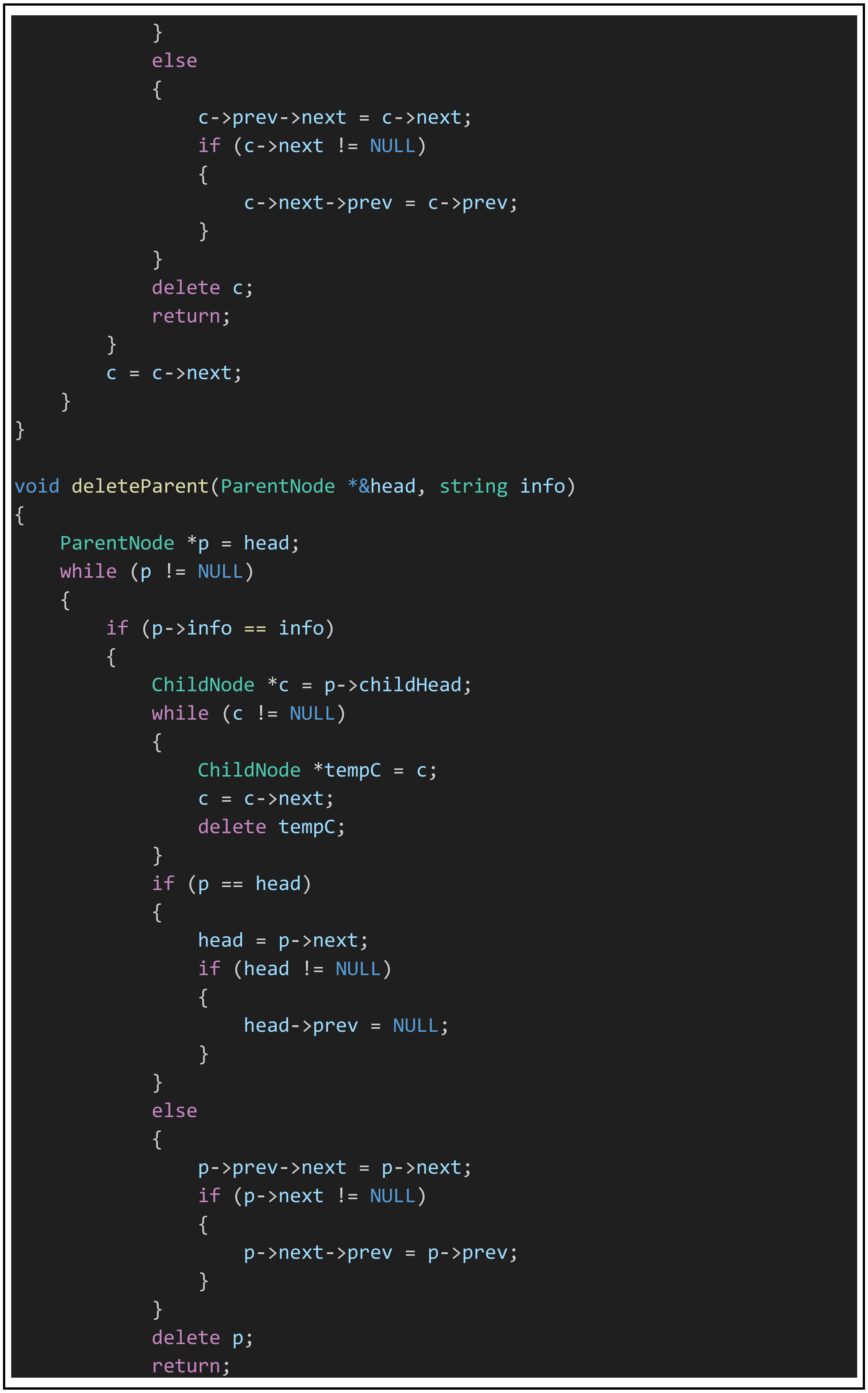
```cpp
    string info;
    ChildNode *childHead;
    ParentNode *next;
    ParentNode *prev;
};

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

ChildNode *createChild(string info)
{
    ChildNode *newNode = new ChildNode;
    newNode->info = info;
    newNode->next = NULL;
    newNode->prev = NULL;
    return newNode;

}

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

void insertChild(ParentNode *head, string parentInfo, string childInfo)
{
    ParentNode *p = head;
```



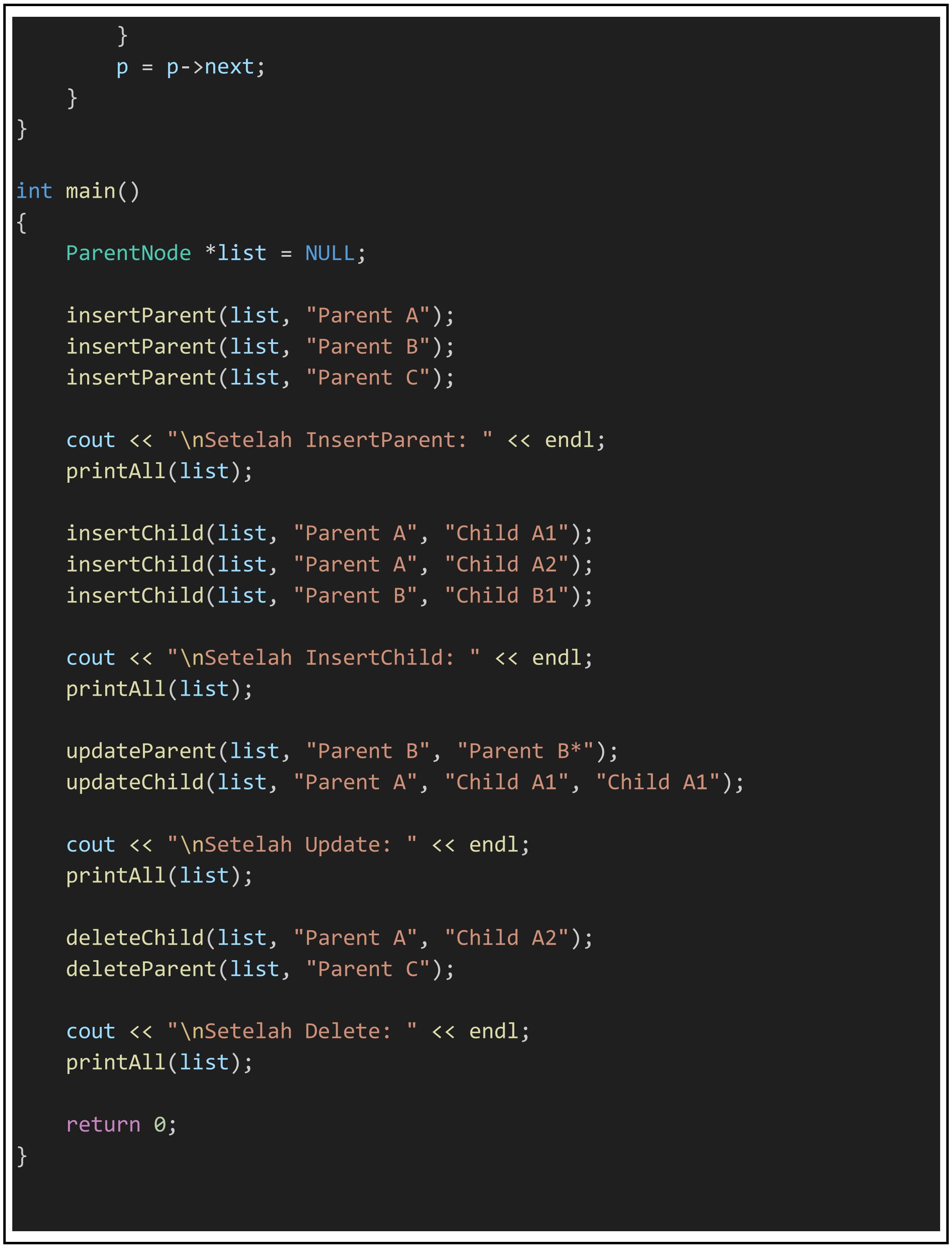
```cpp
    while (p != NULL && p->info != parentInfo)
    {
        p = p->next;
    }

    if (p !=  NULL)
    {
        ChildNode *newChild = createChild(childInfo);
        if (p->childHead ==NULL)
        {
            p->childHead = newChild;
        }
        else{
            ChildNode *c = p->childHead;
            while (c->next != NULL)
            {
                c = c->next;
            }
            c->next = newChild;
            newChild->prev = c;
        }
    }
}

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

void updateParent(ParentNode *head, string oldInfo, string newInfo)
{
    ParentNode *p = head;
    while (p != NULL)
    {
        if (p->info == oldInfo)
        {
            p->info = newInfo;
```



```
            return;
        }
        p = p->next;
    }
}

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

void deleteChild(ParentNode *head, string parentInfo, string childInfo)
{
    ParentNode *p = head;
    while (p != NULL && p->info != parentInfo)
    {
        p = p->next;
    }

    if (p != NULL)
    {
        ChildNode *c = p ->childHead;
        while (c != NULL)
        {
            if (c->info == childInfo)
            {
                if (c == p->childHead)
                {
                    p->childHead->prev = NULL;
                }
```

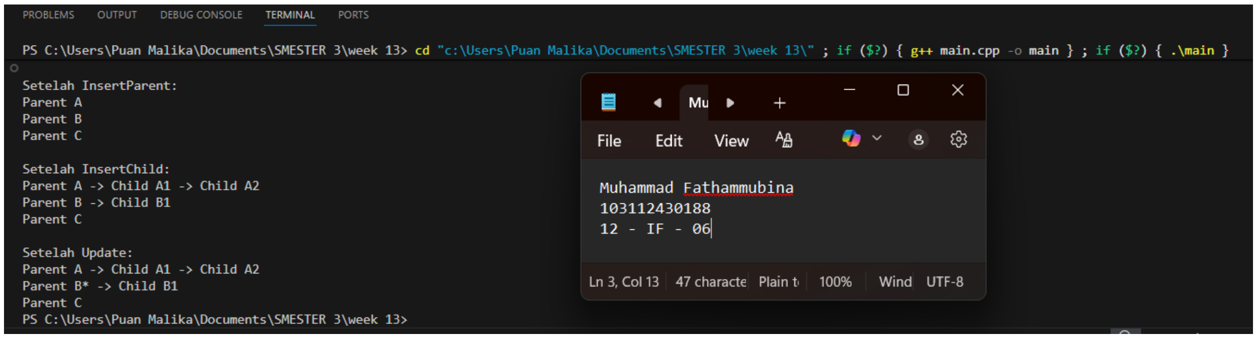


```
            }
            else
            {
                c->prev->next = c->next;
                if (c->next != NULL)
                {
                    c->next->prev = c->prev;
                }
            }
            delete c;
            return;
        }
        c = c->next;
    }
}

void deleteParent(ParentNode *&head, string info)
{
    ParentNode *p = head;
    while (p != NULL)
    {
        if (p->info == info)
        {
            ChildNode *c = p->childHead;
            while (c != NULL)
            {
                ChildNode *tempC = c;
                c = c->next;
                delete tempC;
            }
            if (p == head)
            {
                head = p->next;
                if (head != NULL)
                {
                    head->prev = NULL;
                }
            }
            else
            {
                p->prev->next = p->next;
                if (p->next != NULL)
                {
                    p->next->prev = p->prev;
                }
            }
            delete p;
            return;
```



```cpp
        }
        p = p->next;
    }
}

int main()
{
    ParentNode *list = NULL;

    insertParent(list, "Parent A");
    insertParent(list, "Parent B");
    insertParent(list, "Parent C");

    cout << "\nSetelah InsertParent: " << endl;
    printAll(list);

    insertChild(list, "Parent A", "Child A1");
    insertChild(list, "Parent A", "Child A2");
    insertChild(list, "Parent B", "Child B1");

    cout << "\nSetelah InsertChild: " << endl;
    printAll(list);

    updateParent(list, "Parent B", "Parent B*");
    updateChild(list, "Parent A", "Child A1", "Child A1");

    cout << "\nSetelah Update: " << endl;
    printAll(list);

    deleteChild(list, "Parent A", "Child A2");
    deleteParent(list, "Parent C");

    cout << "\nSetelah Delete: " << endl;
    printAll(list);

    return 0;
}
```

Screenshots Output



```
PS C:\Users\Puan Malika\Documents\SMESTER 3\week 13> cd "c:\Users\Puan Malika\Documents\SMESTER 3\week 13\" ; if ($?) { g++ main.cpp -o main } ; if ($?) { .\main }

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
PS C:\Users\Puan Malika\Documents\SMESTER 3\week 13>
```

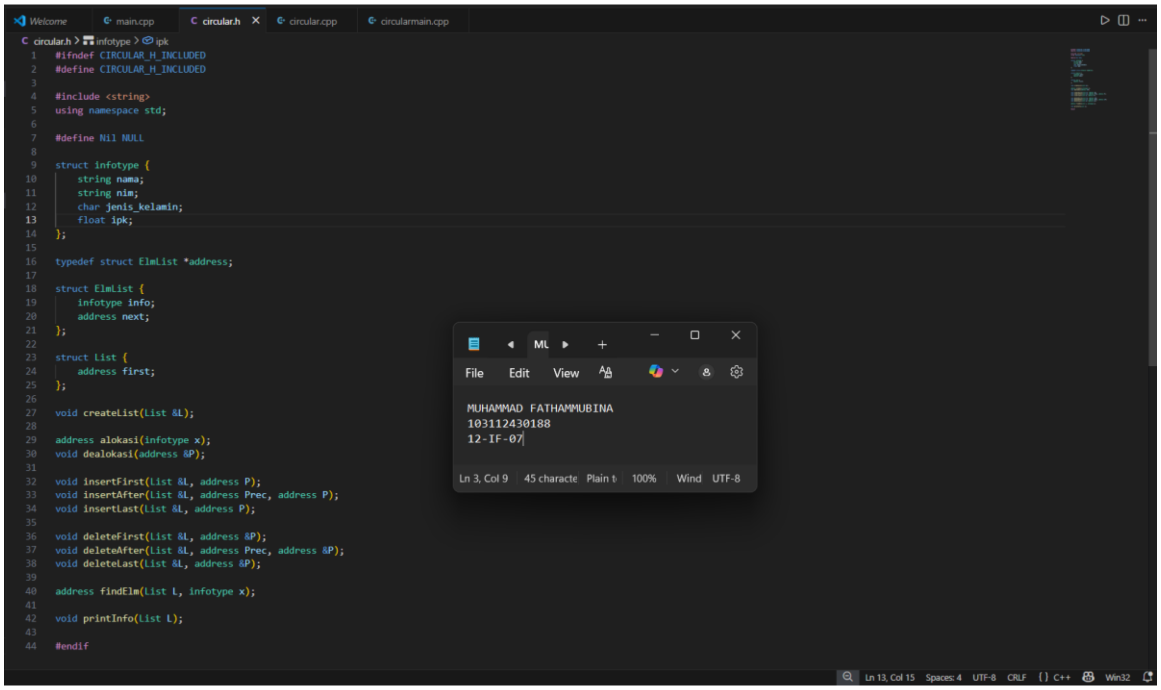
```
Muhammad Fathammubina
103112430188
12 - IF - 06
```

Deskripsi:

Program di atas di atas membuat struktur data multilevel doubly linked list (orang-tua & anak). ParentNode tersambung secara double link (punya next & prev), dan setiap parent bisa menyimpan daftar child sendiri lewat childHead yang juga berbentuk doubly linked list. Ada fungsi untuk membuat, menambah, menampilkan, mengupdate, dan menghapus parent maupun child secara dinamis dengan new dan delete. Alur program di main() mengisi 3 parent, lalu menambahkan beberapa child, kemudian melakukan update nama, dan terakhir menghapus 1 child serta 1 parent beserta semua child-nya. Namun ada bug kecil: di updateChild operator == tertulis c->info == newChildInfo; (seharusnya =), dan logika deleteChild juga kurang tepat karena penempatan else bisa salah menghapus node. Secara inti, program ini meniru hubungan hierarki data (induk–anak) dengan pointer yang saling terhubung agar bisa ditelusuri maju-mundur di setiap level.

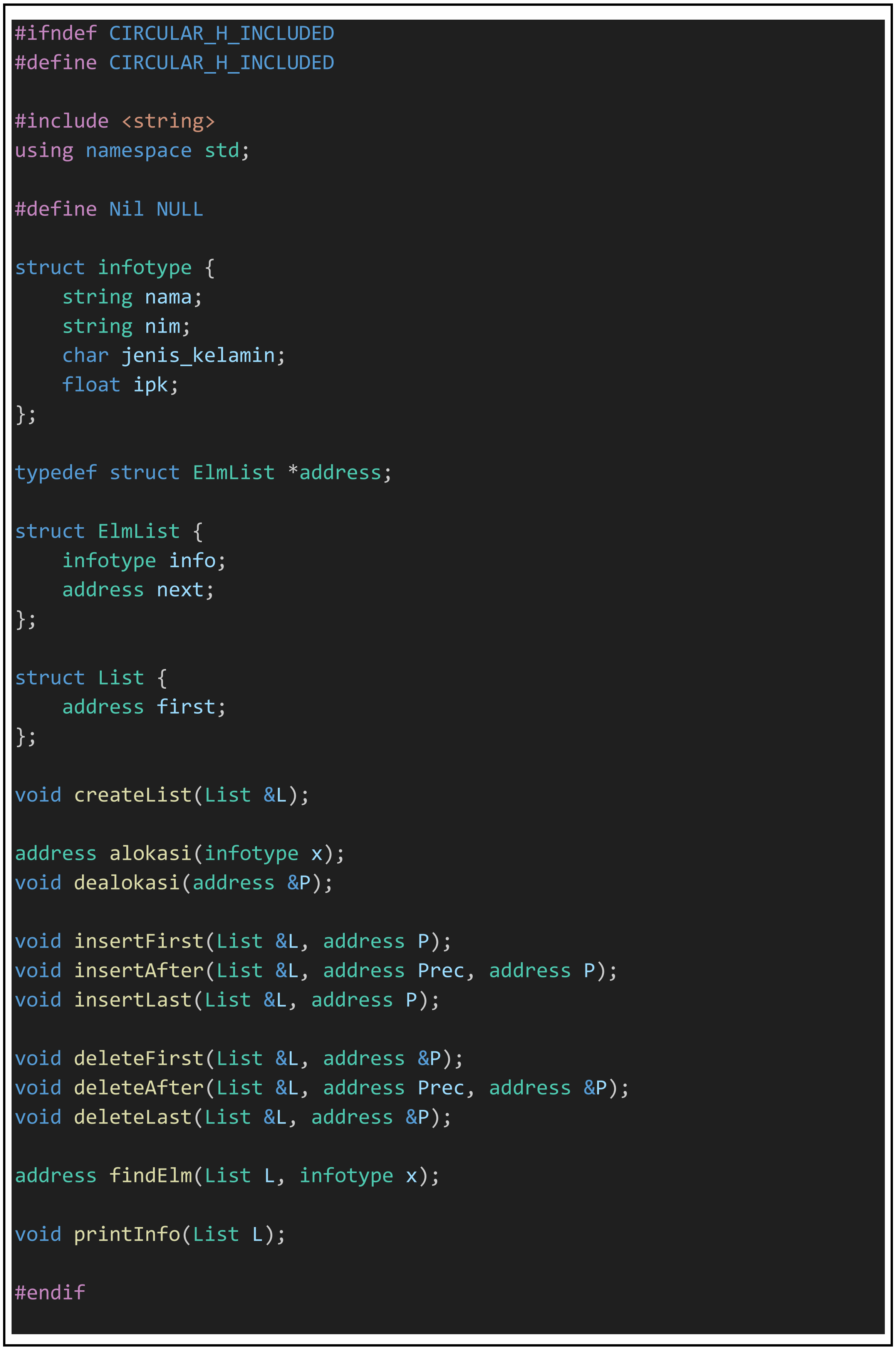
Unguided 1

circular.h



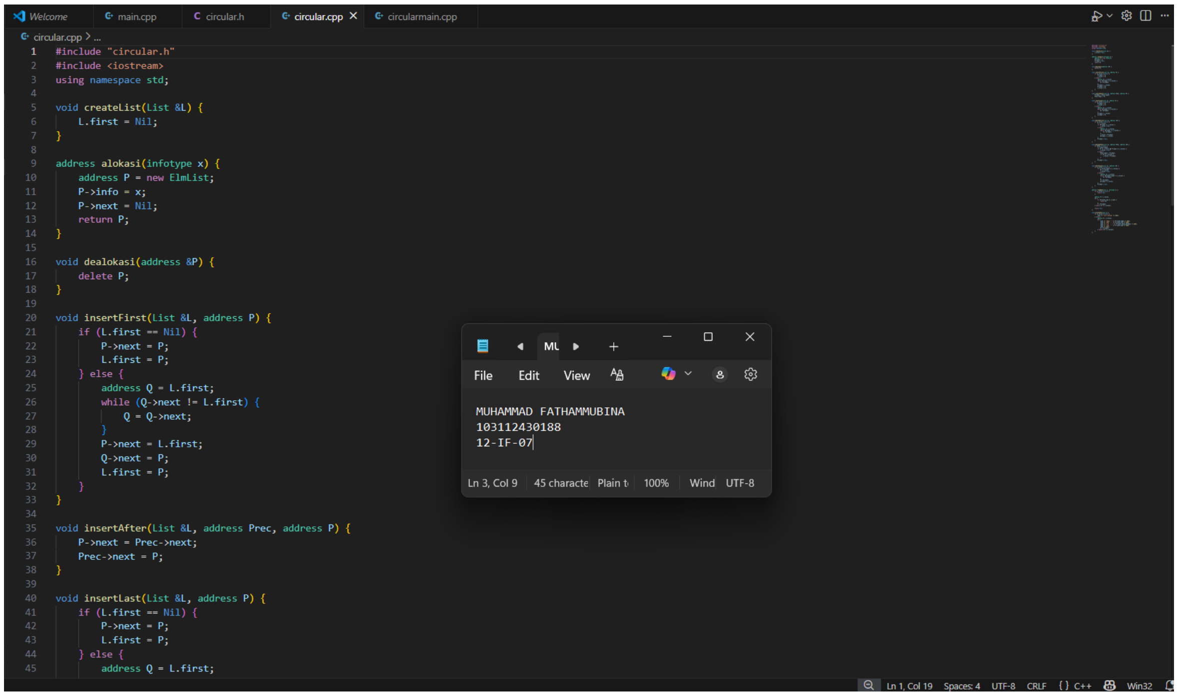
```cpp
#ifndef CIRCULAR_H_INCLUDED
#define CIRCULAR_H_INCLUDED

#include <string>
using namespace std;

#define Nil NULL

struct infotype {
    string nama;
    string nim;
    char jenis_kelamin;
    float ipk;
};

typedef struct ElmList *address;

struct ElmList {
    infotype info;
    address next;
};

struct List {
    address first;
};

void createList(List &L);

address alokasi(infotype x);
void dealokasi(address &P);

void insertFirst(List &L, address P);
void insertAfter(List &L, address Prec, address P);
void insertLast(List &L, address P);

void deleteFirst(List &L, address &P);
void deleteAfter(List &L, address Prec, address &P);
void deleteLast(List &L, address &P);

address findElm(List L, infotype x);

void printInfo(List L);

#endif
```

```
MUHAMMAD FATHAMMUBINA
103112430188
12-IF-07
```



```cpp
#ifndef CIRCULAR_H_INCLUDED
#define CIRCULAR_H_INCLUDED

#include <string>
using namespace std;

#define Nil NULL

struct infotype {
    string nama;
    string nim;
    char jenis_kelamin;
    float ipk;
};

typedef struct ElmList *address;

struct ElmList {
    infotype info;
    address next;
};

struct List {
    address first;
};

void createList(List &L);

address alokasi(infotype x);
void dealokasi(address &P);

void insertFirst(List &L, address P);
void insertAfter(List &L, address Prec, address P);
void insertLast(List &L, address P);

void deleteFirst(List &L, address &P);
void deleteAfter(List &L, address Prec, address &P);
void deleteLast(List &L, address &P);

address findElm(List L, infotype x);

void printInfo(List L);

#endif
```

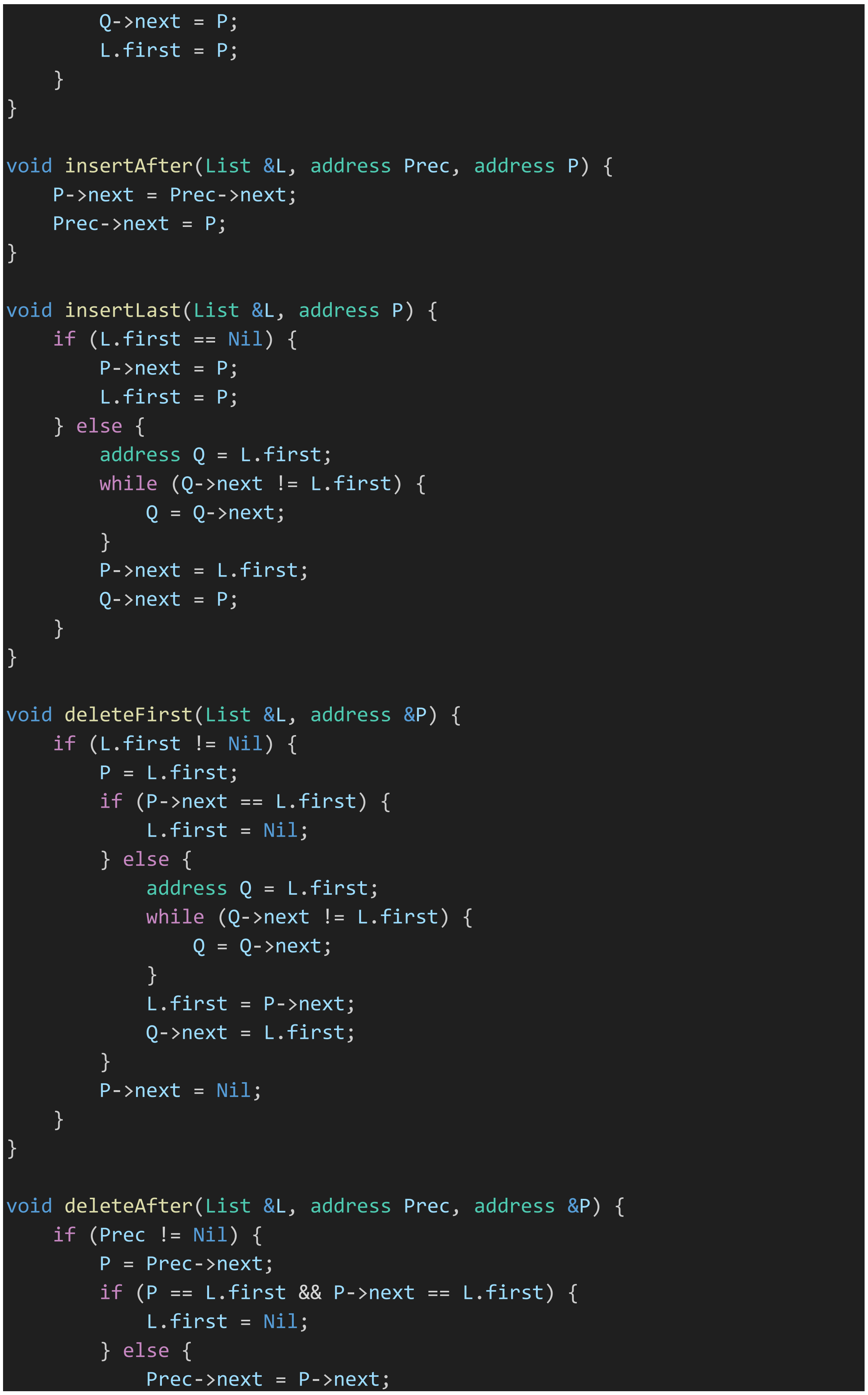
circular.cpp



```cpp
#include "circular.h"
#include <iostream>
using namespace std;

void createList(List &L) {
    L.first = Nil;
}

address alokasi(infotype x) {
    address P = new ElmList;
    P->info = x;
    P->next = Nil;
    return P;
}

void dealokasi(address &P) {
    delete P;
}

void insertFirst(List &L, address P) {
    if (L.first == Nil) {
        P->next = P;
        L.first = P;
    } else {
        address Q = L.first;
        while (Q->next != L.first) {
            Q = Q->next;
        }
        P->next = L.first;
        Q->next = P;
        L.first = P;
    }
}

void insertAfter(List &L, address Prec, address P) {
    P->next = Prec->next;
    Prec->next = P;
}

void insertLast(List &L, address P) {
    if (L.first == Nil) {
        P->next = P;
        L.first = P;
    } else {
        address Q = L.first;
```

```
MUHAMMAD FATHAMMUBINA
103112430188
12-IF-07
```

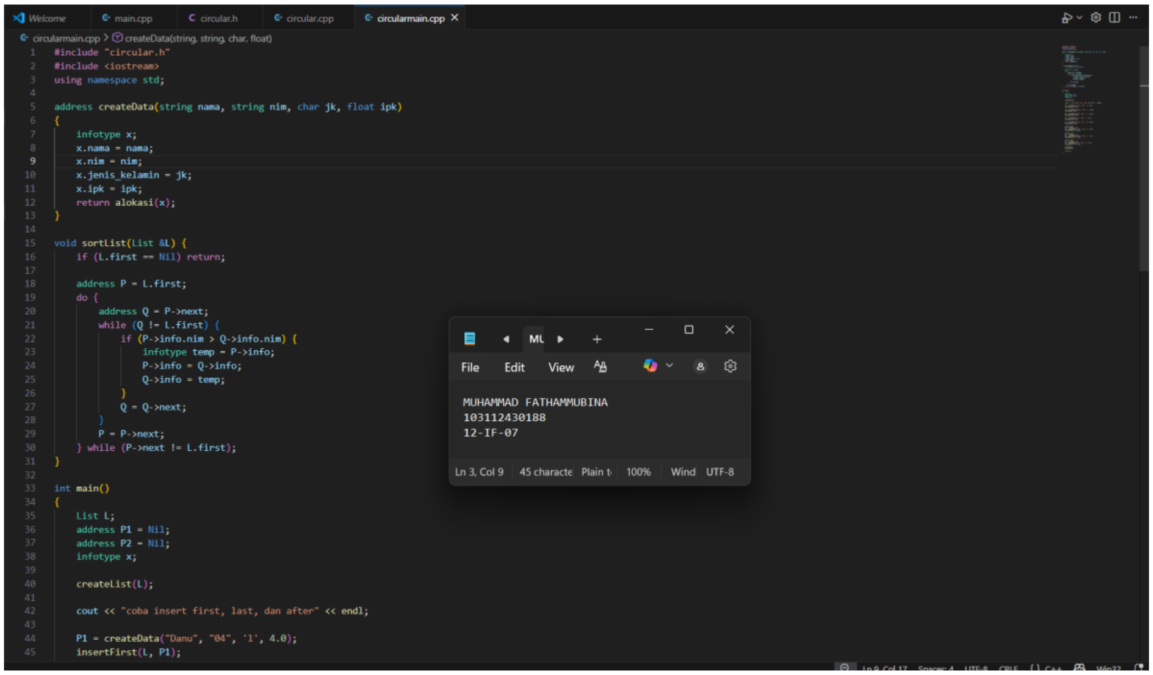
```cpp
#include "circular.h"
#include <iostream>
using namespace std;

void createList(List &L) {
    L.first = Nil;
}

address alokasi(infotype x) {
    address P = new ElmList;
    P->info = x;
    P->next = Nil;
    return P;
}

void dealokasi(address &P) {
    delete P;
}

void insertFirst(List &L, address P) {
    if (L.first == Nil) {
        P->next = P;
        L.first = P;
    } else {
        address Q = L.first;
        while (Q->next != L.first) {
            Q = Q->next;
        }
        P->next = L.first;
```



```
        Q->next = P;
        L.first = P;
    }
}

void insertAfter(List &L, address Prec, address P) {
    P->next = Prec->next;
    Prec->next = P;
}

void insertLast(List &L, address P) {
    if (L.first == Nil) {
        P->next = P;
        L.first = P;
    } else {
        address Q = L.first;
        while (Q->next != L.first) {
            Q = Q->next;
        }
        P->next = L.first;
        Q->next = P;
    }
}

void deleteFirst(List &L, address &P) {
    if (L.first != Nil) {
        P = L.first;
        if (P->next == L.first) {
            L.first = Nil;
        } else {
            address Q = L.first;
            while (Q->next != L.first) {
                Q = Q->next;
            }
            L.first = P->next;
            Q->next = L.first;
        }
        P->next = Nil;
    }
}

void deleteAfter(List &L, address Prec, address &P) {
    if (Prec != Nil) {
        P = Prec->next;
        if (P == L.first && P->next == L.first) {
            L.first = Nil;
        } else {
            Prec->next = P->next;
            if (P == L.first) {
                L.first = P->next;
            }
        }
        P->next = Nil;
    }
}

void deleteLast(List &L, address &P) {
    if (L.first != Nil) {
        if (L.first->next == L.first) {
            P = L.first;
            L.first = Nil;
        } else {
            address Q = L.first;
            while (Q->next->next != L.first) {
                Q = Q->next;
            }
            P = Q->next;
            Q->next = L.first;
        }
        P->next = Nil;
    }
}

address findElm(List L, infotype x) {
    if (L.first == Nil) {
        return Nil;
    }

    address P = L.first;
    do {
        if (P->info.nim == x.nim) {
            return P;
        }
        P = P->next;
    } while (P != L.first);

    return Nil;
}

void printInfo(List L) {
    if (L.first == Nil) {
        cout << "List kosong" << endl;
    } else {
        address P = L.first;
        do {
            cout << "Nama : " << P->info.nama << endl;
```

```cpp
            cout << "NIM  : " << P->info.nim << endl;
            cout << "L/P    : " << P->info.jenis_kelamin << endl;
            cout << "IPK  : " << P->info.ipk << endl;
            cout << endl;
            P = P->next;
        } while (P != L.first);
    }
}
```

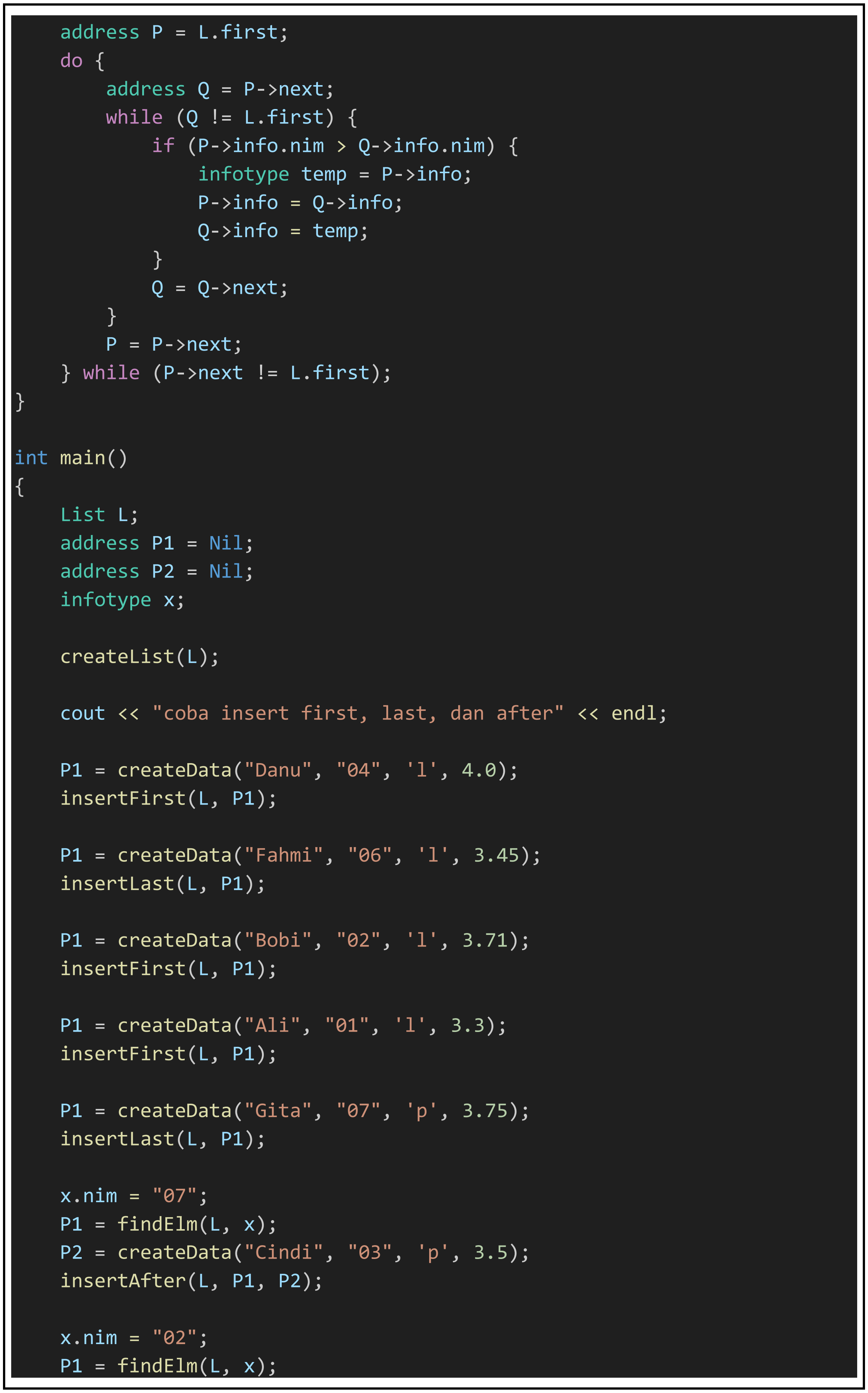
circularmain.cpp



```cpp
#include "circular.h"
#include <iostream>
using namespace std;

address createData(string nama, string nim, char jk, float ipk)
{
    infotype x;
    x.nama = nama;
    x.nim = nim;
    x.jenis_kelamin = jk;
    x.ipk = ipk;
    return alokasi(x);
}

void sortList(List &L) {
    if (L.first == Nil) return;

    address P = L.first;
    do {
        address Q = P->next;
        while (Q != L.first) {
            if (P->info.nim > Q->info.nim) {
                infotype temp = P->info;
                P->info = Q->info;
                Q->info = temp;
            }
            Q = Q->next;
        }
        P = P->next;
    } while (P->next != L.first);
}

int main()
{
    List L;
    address P1 = Nil;
    address P2 = Nil;
    infotype x;

    createList(L);

    cout << "coba insert first, last, dan after" << endl;

    P1 = createData("Danu", "04", 'l', 4.0);
    insertFirst(L, P1);
```

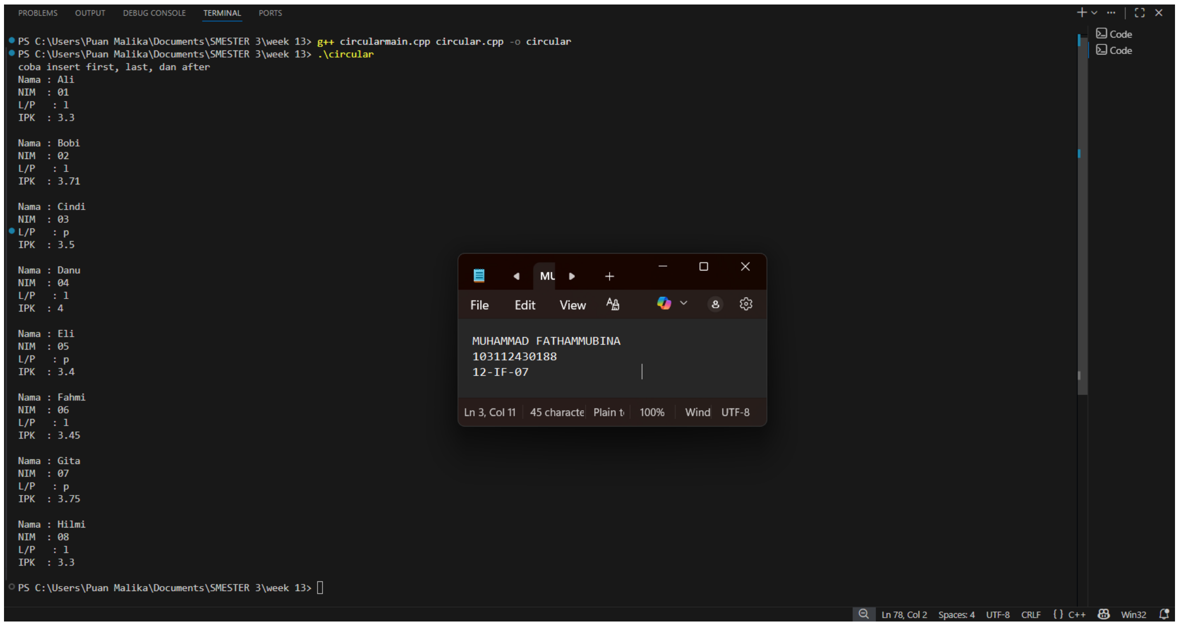
```
MUHAMMAD FATHAMMUBINA
103112430188
12-IF-07
```

```cpp
#include "circular.h"
#include <iostream>
using namespace std;

address createData(string nama, string nim, char jk, float ipk)
{
    infotype x;
    x.nama = nama;
    x.nim = nim;
    x.jenis_kelamin = jk;
    x.ipk = ipk;
    return alokasi(x);
}

void sortList(List &L) {
    if (L.first == Nil) return;
```



```cpp
    address P = L.first;
    do {
        address Q = P->next;
        while (Q != L.first) {
            if (P->info.nim > Q->info.nim) {
                infotype temp = P->info;
                P->info = Q->info;
                Q->info = temp;
            }
            Q = Q->next;
        }
        P = P->next;
    } while (P->next != L.first);
}

int main()
{
    List L;
    address P1 = Nil;
    address P2 = Nil;
    infotype x;

    createList(L);

    cout << "coba insert first, last, dan after" << endl;

    P1 = createData("Danu", "04", 'l', 4.0);
    insertFirst(L, P1);

    P1 = createData("Fahmi", "06", 'l', 3.45);
    insertLast(L, P1);

    P1 = createData("Bobi", "02", 'l', 3.71);
    insertFirst(L, P1);

    P1 = createData("Ali", "01", 'l', 3.3);
    insertFirst(L, P1);

    P1 = createData("Gita", "07", 'p', 3.75);
    insertLast(L, P1);

    x.nim = "07";
    P1 = findElm(L, x);
    P2 = createData("Cindi", "03", 'p', 3.5);
    insertAfter(L, P1, P2);

    x.nim = "02";
    P1 = findElm(L, x);
    P2 = createData("Hilmi", "08", 'l', 3.3);
    insertAfter(L, P1, P2);

    x.nim = "04";
    P1 = findElm(L, x);
    P2 = createData("Eli", "05", 'p', 3.4);
    insertAfter(L, P1, P2);

    sortList(L);
    printInfo(L);

    return 0;
}
```

Screenshots Output



```
PS C:\Users\Puan Malika\Documents\SMESTER 3\week 13> g++ circularmain.cpp circular.cpp -o circular
PS C:\Users\Puan Malika\Documents\SMESTER 3\week 13> .\circular
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 : Cindi
NIM  : 03
L/P   : p
IPK  : 3.5

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 : Hilmi
NIM  : 08
L/P   : l
IPK  : 3.3

PS C:\Users\Puan Malika\Documents\SMESTER 3\week 13>
```

```
MUHAMMAD FATHAMMUBINA
103112430188
12-IF-07
```

Deskripsi:

Program di atas adalah implementasi ADT Circular Singly Linked List untuk menyimpan data mahasiswa (nama, NIM, jenis kelamin, IPK). List bersifat circular, artinya node terakhir menunjuk kembali ke first, sehingga traversal pakai do-while. Terdapat operasi ADT lengkap: buat list, alokasi/dealokasi node, insert di awal/tengah/akhir, delete, search, print, dan sort. Di main() dibuat list, diisi beberapa data dengan kombinasi insertFirst, insertLast, dan insertAfter, lalu list diurutkan berdasarkan NIM menggunakan teknik bubble sort dengan swap isi info antar node (tanpa mengubah pointer). Program kemudian menampilkan seluruh isi list. Secara konsep sudah sesuai circular ADT, hanya ada bagian kurang rapi seperti fungsi dealokasi yang tidak mengeset pointer ke Nil, tetapi inti program adalah membangun dan mengelola list melingkar secara dinamis dengan pointer agar data bisa ditambah, dicari, diurut, dan dihapus

C. Kesimpulan

Praktikum ini menunjukkan bahwa Multi Linked List efektif untuk menyimpan data hierarki parent–child secara dinamis di memori. Program berhasil menerapkan operasi insert parent di akhir, insert child ke parent tertentu, traversal print semua level, update parent dan child, serta delete parent beserta seluruh child-nya menggunakan pointer bertingkat. Struktur doubly linked di kedua level memudahkan penelusuran dan manipulasi node. Kesalahan kecil pada operator update child (== seharusnya =) memberi pelajaran bahwa manipulasi isi node dan pointer harus ditulis tepat agar data berubah sesuai tujuan. Secara keseluruhan, program sudah menggambarkan konsep relasi 1 parent ke banyak child sesuai teori Multi Linked List di modul, dan membuktikan bahwa ADT ini fleksibel untuk mengelola data bertingkat.

D. Referensi

Muliono, R. (2017). Abstract Data Type (ADT). Universitas Multimedia Nusantara.

Alipiqri. (n.d.). *Jurnal Modul 10 – Multi Linked List* [PDF]. Scribd.

Ulum, M. B., & Kastina, M. (2018). Modul Praktikum Struktur Data C++ (UEU-Course-10190-7_0264). Universitas Esa Unggul