****

**LAPORAN PRAKTIKUM**

* **Identitas Praktikum**

Nama MK : Struktur Data

Kode MK : CCK2AAB4

Bobot SKS : 4 SKS

Tempat : L-Program, Gedung DC, lantai 3

Hari, tanggal : Selasa, 24 Desember 2024

Jam : 12:30-15:30 WIB

Topik praktikum : Modul-13 MULTI LINKED LIST

* **Identitas Mahasiswa**

Nama lengkap : Afad Fath Musyarof Halim

NIM : 2211104030

Program Studi : S-1 Software Engineering

* **Hasil Praktikum**

1. MULTI LINKED LIST
   1. Insert

|  |
| --- |
| /\* buat dahulu elemen yang akan disisipkan \*/  address\_anak alokasiAnak(infotypeanak X){  STRUKTUR DATA 105  address\_anak p = alokasi(X);  next(p) = null;  prev(p) = null;  return p;  }  /\* mencari apakah ada elemen pegawai dengan info X \*/  address findElm(listinduk L, infotypeinduk X){  address cariInduk = head(L);  do{  if(cariInduk.info == X){  return cariInduk;  }else{  cariInduk = next(cariInduk);  }  }while(cariInduk.info!=X || cariInduk!=last(L))  }  /\* menyisipkan anak pada akhir list anak \*/  void insertLastAnak(listanak &Lanak, address\_anak P){  address\_anak ! = head(&Lanak);  do{  Q = next(Q);  }while(next(&Lanak)!=NULL)  next(Q) = P;  prev(P) = Q;  next(P) = NULL;  } |

* + 1. Delete

|  |
| --- |
| /\*file : multilist .h\*/  /\* contoh ADT list berkait dengan representasi fisik pointer\*/  /\* representasi address dengan pointer\*/  /\* info tipe adalah integer \*/  #ifndef MULTILIST\_H\_INCLUDED  #define MULTILIST\_H\_INCLUDED  #include <stdio.h>  #define Nil NULL  #define info(P) (P)->info  #define next(P) (P)->next  #define first(L) ((L).first)  #define last(L) ((L).last)  typedef int infotypeanak;  typedef int infotypeinduk;  typedef struct elemen\_list\_induk \*address;  typedef struct elemen\_list\_anak \*address\_anak;  /\* define list : \*/  /\* list kosong jika first(L)=Nil  setiap elemen address P dapat diacu info(P) atau next(P)  elemen terakhir list jika addressnya last, maka next(last) = Nil \*/  struct elemen\_list\_anak{  /\* struct ini untuk menyimpan elemen anak dan pointer penunjuk  elemen tetangganya \*/  infotypeanak info;  address\_anak next;  address\_anak prev;  };  struct listanak {  /\* struct ini digunakan untuk menyimpan list anak itu sendiri \*/  address\_anak first;  address\_anak last;  };  struct elemen\_list\_induk{  /\* struct ini untuk menyimpan elemen induk dan pointer penunjuk  elemen tetangganya \*/  infotypeanak info;  listanak lanak;  address next;  address prev;  };  struct listinduk {  /\* struct ini digunakan untuk menyimpan list induk itu sendiri \*/  address first;  address last;  };  /\*\*\*\*\*\*\*\*\* pengecekan apakah list kosong \*\*\*\*\*\*\*\*\*\*\*/  boolean ListEmpty(listinduk L);  /\*mengembalikan nilai true jika list induk kosong\*/  boolean ListEmptyAnak(listanak L);  /\*mengembalikan nilai true jika list anak kosong\*/  /\*\*\*\*\*\*\*\*\* pembuatan list kosong \*\*\*\*\*\*\*\*\*/  void CreateList(listinduk &L);  /\* I.S. sembarang  F.S. terbentuk list induk kosong\*/  void CreateListAnak(listanak &L);  /\* I.S. sembarang  F.S. terbentuk list anak kosong\*/  /\*\*\*\*\*\*\*\*\* manajemen memori \*\*\*\*\*\*\*\*\*/  address alokasi(infotypeinduk P);  /\* mengirimkan address dari alokasi sebuah elemen induk  jika alokasi berhasil, maka nilai address tidak Nil dan jika gagal  nilai address Nil \*/  address\_anak alokasiAnak(infotypeanak P);  /\* mengirimkan address dari alokasi sebuah elemen anak  jika alokasi berhasil, maka nilai address tidak Nil dan jika gagal  nilai address\_anak Nil \*/  void dealokasi(address P);  /\* I.S. P terdefinisi  F.S. memori yang digunakan P dikembalikan ke sistem \*/  void dealokasiAnak(address\_anak P);  /\* I.S. P terdefinisi  F.S. memori yang digunakan P dikembalikan ke sistem \*/  /\*\*\*\*\*\*\*\*\* pencarian sebuah elemen list \*\*\*\*\*\*\*\*\*/  address findElm(listinduk L, infotypeinduk X);  /\* mencari apakah ada elemen list dengan info(P) = X  jika ada, mengembalikan address elemen tab tsb, dan Nil jika sebaliknya  \*/  address\_anak findElm(listanak Lanak, infotypeanak X);  /\* mencari apakah ada elemen list dengan info(P) = X  jika ada, mengembalikan address elemen tab tsb, dan Nil jika sebaliknya  \*/  boolean fFindElm(listinduk L, address P);  /\* mencari apakah ada elemen list dengan alamat P  mengembalikan true jika ada dan false jika tidak ada \*/  boolean fFindElmanak(listanak Lanak, address\_anak P);  /\* mencari apakah ada elemen list dengan alamat P  mengembalikan true jika ada dan false jika tidak ada \*/  address findBefore(listinduk L, address P);  /\* mengembalikan address elemen sebelum P  jika P berada pada awal list, maka mengembalikan nilai Nil \*/  address\_anak findBeforeAnak(listanak Lanak, infotypeinduk X, address\_anak  P);  /\* mengembalikan address elemen sebelum P dimana info(P) = X  jika P berada pada awal list, maka mengembalikan nilai Nil \*/  /\*\*\*\*\*\*\*\*\* penambahan elemen \*\*\*\*\*\*\*\*\*\*/  void insertFirst(listinduk &L, address P);  /\* I.S. sembarang, P sudah dialokasikan  F.S. menempatkan elemen beralamat P pada awal list \*/  void insertAfter(listinduk &L, address P, address Prec);  /\* I.S. sembarang, P dan Prec alamt salah satu elemen list  F.S. menempatkan elemen beralamat P sesudah elemen beralamat Prec \*/  void insertLast(listinduk &L, address P);  /\* I.S. sembarang, P sudah dialokasikan  F.S. menempatkan elemen beralamat P pada akhir list \*/  void insertFirstAnak(listanak &L, address\_anak P);  /\* I.S. sembarang, P sudah dialokasikan  F.S. menempatkan elemen beralamat P pada awal list \*/  void insertAfterAnak(listanak &L, address\_anak P, address\_anak Prec);  /\* I.S. sembarang, P dan Prec alamt salah satu elemen list  F.S. menempatkan elemen beralamat P sesudah elemen beralamat Prec \*/  void insertLastAnak(listanak &L, address\_anak P);  /\* I.S. sembarang, P sudah dialokasikan  F.S. menempatkan elemen beralamat P pada akhir list \*/  /\*\*\*\*\*\*\*\*\* penghapusan sebuah elemen \*\*\*\*\*\*\*\*\*/  void delFirst(listinduk &L, address &P);  /\* I.S. list tidak kosong  F.S. adalah alamat dari alamat elemen pertama list  sebelum elemen pertama list dihapus  elemen pertama list hilang dan list mungkin menjadi kosong  first elemen yang baru adalah successor first elemen yang lama \*/  void delLast(listinduk &L, address &P);  /\* I.S. list tidak kosong  F.S. adalah alamat dari alamat elemen terakhir list  sebelum elemen terakhir list dihapus  elemen terakhir list hilang dan list mungkin menjadi kosong  last elemen yang baru adalah successor last elemen yang lama \*/  void delAfter(listinduk &L, address &P, address Prec);  /\* I.S. list tidak kosng, Prec alamat salah satu elemen list  F.S. P adalah alamatdari next(Prec), menghapus next(Prec) dari list \*/  void delP (listinduk &L, infotypeinduk X);  /\* I.S. sembarang  F.S. jika ada elemen list dengan alamat P, dimana info(P)=X, maka P  dihapus  dan P di-dealokasi, jika tidak ada maka list tetap  list mungkin akan menjadi kosong karena penghapusan \*/  void delFirstAnak(listanak &L, address\_anak &P);  /\* I.S. list tidak kosong  F.S. adalah alamat dari alamat elemen pertama list  sebelum elemen pertama list dihapus  elemen pertama list hilang dan list mungkin menjadi kosong  first elemen yang baru adalah successor first elemen yang lama \*/  void delLastAnak(listanak &L, address\_anak &P);  /\* I.S. list tidak kosong  F.S. adalah alamat dari alamat elemen terakhir list  sebelum elemen terakhir list dihapus  elemen terakhir list hilang dan list mungkin menjadi kosong  last elemen yang baru adalah successor last elemen yang lama \*/  void delAfterAnak(listanak &L, address\_anak &P, address\_anak Prec);  /\* I.S. list tidak kosng, Prec alamat salah satu elemen list  F.S. P adalah alamatdari next(Prec), menghapus next(Prec) dari list \*/  void delPAnak (listanak &L, infotypeanak X);  /\* I.S. sembarang  F.S. jika ada elemen list dengan alamat P, dimana info(P)=X, maka P  dihapus  dan P di-dealokasi, jika tidak ada maka list tetap  list mungkin akan menjadi kosong karena penghapusan \*/  /\*\*\*\*\*\*\*\*\*\* proses semau elemen list \*\*\*\*\*\*\*\*\*/  void printInfo(list L);  /\* I.S. list mungkin kosong  F.S. jika list tidak kosong menampilkan semua info yang ada pada list  \*/  int nbList(list L);  /\* mengembalikan jumlah elemen pada list \*/  void printInfoAnak(listanak Lanak);  /\* I.S. list mungkin kosong  F.S. jika list tidak kosong menampilkan semua info yang ada pada list  \*/  int nbListAnak(listanak Lanak);  /\* mengembalikan jumlah elemen pada list anak \*/  /\*\*\*\*\*\*\*\*\*\* proses terhadap list \*\*\*\*\*\*\*\*\*\*/  void delAll(listinduk &L);  /\* menghapus semua elemen list dan semua elemen di-dealokasi \*/  #endif |

* + 1. Praktikum

|  |  |
| --- | --- |
| #include <iostream>  #include <string>  using namespace std;    // Struktur untuk mata kuliah  struct Course {  string courseName;  Course\* nextCourse;  };    // Struktur untuk mahasiswa  struct Student {  string studentName;  Course\* courseList;  Student\* nextStudent;  };    // Fungsi untuk menambahkan mahasiswa baru  Student\* addStudent(Student\* head, string name) {  Student\* newStudent = new Student{name, nullptr, head};  return newStudent;  }    // Fungsi untuk menambahkan mata kuliah ke mahasiswa  void addCourseToStudent(Student\* student, string courseName) {  Course\* newCourse = new Course{courseName, student->courseList};  student->courseList = newCourse;  }    // Fungsi untuk menampilkan daftar mahasiswa dan mata kuliah yang diambil  void displayStudents(Student\* head) {  while (head) {  cout << "Student: " << head->studentName << endl;  cout << " Courses: ";  Course\* course = head->courseList;  while (course) {  cout << course->courseName << " ";  course = course->nextCourse;  }  cout << endl;  head = head->nextStudent;  }  }    int main() {  // Membuat daftar mahasiswa  Student\* students = nullptr;    // Menambahkan mahasiswa  students = addStudent(students, "Alice");  students = addStudent(students, "Bob");    // Menambahkan mata kuliah untuk mahasiswa  addCourseToStudent(students, "Mathematics"); // Bob  addCourseToStudent(students, "Physics"); // Bob    addCourseToStudent(students->nextStudent, "Programming"); // Alice  addCourseToStudent(students->nextStudent, "English"); // Alice    // Menampilkan data mahasiswa  displayStudents(students);    return 0;  } |  |

* + 1. LATIHAN
* Circularlist.h

|  |
| --- |
| #ifndef CIRCULARLIST\_H  #define CIRCULARLIST\_H  #include <string>  struct Mahasiswa {  std::string nama;  std::string nim;  char jenis\_kelamin;  float ipk;  };  typedef Mahasiswa infotype;  struct ElmList {  infotype info;  ElmList\* next;  };  typedef ElmList\* address;  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 // CIRCULARLIST\_H |

* Circularlist.cpp

|  |
| --- |
| #include "circularlist.h"  #include <iostream>  void CreateList(List &L) {  L.first = nullptr;  }  address alokasi(infotype x) {  address P = new ElmList;  P->info = x;  P->next = nullptr;  return P;  }  address createData(std::string nama, std::string nim, char jenis\_kelamin, float ipk) {  infotype x;  x.nama = nama;  x.nim = nim;  x.jenis\_kelamin = jenis\_kelamin;  x.ipk = ipk;  return alokasi(x);  }  void dealokasi(address &P) {  delete P;  P = nullptr;  }  void insertFirst(List &L, address P) {  if (L.first == nullptr) {  L.first = P;  P->next = P;  } else {  address last = L.first;  while (last->next != L.first) {  last = last->next;  }  P->next = L.first;  L.first = P;  last->next = L.first;  }  }  void insertAfter(List &L, address Prec, address P) {  if (Prec != nullptr) {  P->next = Prec->next;  Prec->next = P;  }  }  void insertLast(List &L, address P) {  if (L.first == nullptr) {  insertFirst(L, P);  } else {  address last = L.first;  while (last->next != L.first) {  last = last->next;  }  last->next = P;  P->next = L.first;  }  }  void deleteFirst(List &L, address &P) {  if (L.first != nullptr) {  P = L.first;  if (L.first->next == L.first) {  L.first = nullptr;  } else {  address last = L.first;  while (last->next != L.first) {  last = last->next;  }  L.first = P->next;  last->next = L.first;  }  P->next = nullptr;  }  }  void deleteAfter(List &L, address Prec, address &P) {  if (Prec != nullptr && Prec->next != L.first) {  P = Prec->next;  Prec->next = P->next;  P->next = nullptr;  }  }  void deleteLast(List &L, address &P) {  if (L.first != nullptr) {  if (L.first->next == L.first) {  P = L.first;  L.first = nullptr;  } else {  address last = L.first;  address precLast = nullptr;  while (last->next != L.first) {  precLast = last;  last = last->next;  }  P = last;  precLast->next = L.first;  }  P->next = nullptr;  }  }  address findElm(List L, infotype x) {  address P = L.first;  if (P != nullptr) {  do {  if (P->info.nim == x.nim) {  return P;  }  P = P->next;  } while (P != L.first);  }  return nullptr;  }  void printInfo(List L) {  address P = L.first;  if (P != nullptr) {  do {  std::cout << "Nama: " << P->info.nama << "\n"  << "NIM : " << P->info.nim << "\n"  << "L/P : " << P->info.jenis\_kelamin << "\n"  << "IPK : " << P->info.ipk << "\n"  << std::endl;  P = P->next;  } while (P != L.first);  }  } |

* Main.cpp

|  |
| --- |
| #include <iostream>  #include "circularlist.cpp"  using namespace std;  int main() {  List L, A, B, L2;  address P1 = NULL;  address P2 = NULL;    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", 'p', 3.3);  insertAfter(L, P1, P2);  x.nim = "04";  P1 = findElm(L, x);  P2 = createData("Eli", "05", 'p', 3.4);  insertAfter(L, P1, P2);  printInfo(L);  return 0;  } |

* Output

A screenshot of a computer

Description automatically generated