**TUGAS ANALISIS DOUBLE LINKED LIST**

**PRAKTIKUM STRUKTUR DATA**

****

**Disusun Oleh:**

**MUHAMMAD RAFKI**

**BP/NIM : 2023/23343078**

**Hari/Tanggal : Selasa/ 05-03-2024**

**Sesi/Jam : 202323430157/08:50 - 10:30 WIB**

**Dosen Pengampu :**

**Randi Proska Sandra, M.Sc.**

**NIP. 221048**

**FAKULTAS TEKNIK**

**DEPARTEMEN TEKNIK ELEKTRONIKA**

**PROGRAM STUDI S1 TEKNIK INFORMATIKA**

**UNIVERSITAS NEGERI PADANG**

**2024**

**1. Insertion At Front**

|  |  |  |  |
| --- | --- | --- | --- |
| Nomor  Program | Baris Program | Petikan Source Code | Penjelasan |
| 1 | 5 - 9 | struct Node {  int data;  struct Node \*next;  struct Node \*prev;  }; | Kode tersebut mendefinisikan struktur untuk simpul dalam linked list berganda. Setiap simpul menyimpan data dan memiliki pointer ke simpul berikutnya dan sebelumnya, memungkinkan penelusuran maju dan mundur dalam daftar. |
| 2 | 11 - 24 | void push(struct Node\*\* head\_ref, int new\_data) {    struct Node\* new\_node = (struct Node\*)malloc(sizeof(struct Node));    new\_node->data = new\_data;    new\_node->next = (\*head\_ref);  new\_node->prev = NULL;    if ((\*head\_ref) != NULL)  (\*head\_ref)->prev = new\_node;    (\*head\_ref) = new\_node;  } | Fungsi `push` digunakan untuk menambahkan simpul baru ke depan linked list berganda. Pertama, sebuah simpul baru dialokasikan memori dengan menggunakan fungsi `malloc`. Nilai dari `new\_data` kemudian dimasukkan ke dalam simpul baru tersebut. Pointer `next` dari simpul baru diarahkan ke simpul yang saat ini menjadi kepala (head) dari linked list, dan pointer `prev` diatur menjadi `NULL`, menunjukkan bahwa simpul baru tersebut akan menjadi kepala. Jika linked list tidak kosong, simpul sebelumnya dari kepala saat ini diatur agar menunjuk ke simpul baru. Akhirnya, kepala dari linked list diubah untuk menunjuk ke simpul baru yang baru saja ditambahkan. |
| 3 | 26 – 39 | void printList(struct Node\* node) {  struct Node\* last;  printf("\nTraversal in forward direction \n");  while (node != NULL) {  printf(" %d ", node->data);  last = node;  node = node->next;  }  printf("\nTraversal in reverse direction \n");  while (last != NULL) {  printf(" %d ", last->data);  last = last->prev;  }  } | Fungsi `printList` digunakan untuk mencetak isi dari linked list berganda dalam dua arah: maju dan mundur. Pertama, dalam penelusuran maju, fungsi tersebut menggunakan loop while untuk mencetak nilai dari setiap simpul yang dimulai dari simpul pertama hingga terakhir, dengan setiap iterasi menunjuk ke simpul berikutnya. Kemudian, dalam penelusuran mundur, nilai-nilai tersebut dicetak dari simpul terakhir hingga simpul pertama menggunakan pointer `prev` dari setiap simpul, menunjuk ke simpul sebelumnya. Hal ini memungkinkan pengguna untuk melihat isi linked list baik secara maju maupun mundur. |
| 4 | 41-51 | int main() {  struct Node\* head = NULL;  push(&head, 8);  push(&head, 6);  push(&head, 4);  printf("Created DLL is: ");  printList(head);  getchar();  return 0;  } | Dalam fungsi `main`, program dimulai dengan linked list berganda kosong, yang diwakili oleh pointer `head` yang diatur menjadi `NULL`. Tiga nilai (8, 6, dan 4) kemudian ditambahkan ke linked list menggunakan fungsi `push`, dengan nilai-nilai ini dimasukkan ke depan linked list. Setelah itu, isi linked list dicetak baik secara maju maupun mundur dengan menggunakan fungsi `printList`. Hasilnya adalah pencetakan nilai-nilai dalam urutan dari simpul pertama hingga terakhir, dan kemudian dari simpul terakhir hingga pertama. Akhirnya, program menunggu input dari pengguna sebelum berakhir. |

**2. Insertion After given Node**

|  |  |  |  |
| --- | --- | --- | --- |
| Nomor  Program | Baris Program | Petikan Source Code | Penjelasan |
| 1 | 4 – 8 | struct Node {  int data;  struct Node \*next;  struct Node \*prev;  }; | Kode tersebut mendefinisikan struktur untuk simpul dalam linked list berganda. Setiap simpul menyimpan data dan memiliki pointer ke simpul berikutnya dan sebelumnya, memungkinkan penelusuran maju dan mundur dalam daftar. |
| 2 | 11 – 24 | void push(struct Node\*\* head\_ref, int new\_data) {    struct Node\* new\_node = (struct Node\*)malloc(sizeof(struct Node));  new\_node->data = new\_data;  new\_node->next = (\*head\_ref);  new\_node->prev = NULL;  if ((\*head\_ref) != NULL)  (\*head\_ref)->prev = new\_node;  (\*head\_ref) = new\_node;  } | Fungsi `push` digunakan untuk menambahkan simpul baru ke depan linked list berganda. Pertama, fungsi mengalokasikan memori untuk simpul baru menggunakan `malloc`, kemudian nilai dari `new\_data` dimasukkan ke dalam simpul tersebut. Pointer `next` dari simpul baru diatur untuk menunjuk ke simpul yang saat ini menjadi kepala (head) dari linked list, sementara pointer `prev` diatur menjadi `NULL` untuk menunjukkan simpul baru tersebut akan menjadi kepala. Jika linked list tidak kosong, simpul sebelumnya dari kepala saat ini diatur untuk menunjuk ke simpul baru. Akhirnya, kepala dari linked list diperbarui agar menunjuk ke simpul baru yang baru saja ditambahkan. |
| 3 | 27 – 46 | void insertAfter(struct Node\* prev\_node, int new\_data) {    if (prev\_node == NULL) {  printf("node sebelumnya tidak boleh NULL");  return;  }    struct Node\* new\_node = (struct Node\*)malloc(sizeof(struct Node));    new\_node->data = new\_data;    new\_node->next = prev\_node->next;    prev\_node->next = new\_node;    new\_node->prev = prev\_node;  if (new\_node->next != NULL)  new\_node->next->prev = new\_node;  } | Fungsi `insertAfter` digunakan untuk menyisipkan simpul baru setelah simpul tertentu dalam linked list berganda. Pertama, fungsi memeriksa apakah simpul sebelumnya tidak boleh `NULL`, jika iya, maka akan mencetak pesan kesalahan dan mengembalikan eksekusi. Selanjutnya, fungsi mengalokasikan memori untuk simpul baru menggunakan `malloc`, kemudian nilai `new\_data` dimasukkan ke dalam simpul tersebut. Pointer `next` dari simpul baru diatur untuk menunjuk ke simpul yang berada setelah simpul sebelumnya. Kemudian, pointer `next` dari simpul sebelumnya diubah untuk menunjuk ke simpul baru. Pointer `prev` dari simpul baru diatur untuk menunjuk ke simpul sebelumnya. Terakhir, jika simpul baru tidak menjadi simpul terakhir dalam linked list, pointer `prev` dari simpul setelah simpul baru juga diubah untuk menunjuk ke simpul baru tersebut. |
| 4 | 49 – 62 | void printList(struct Node\* node) {  struct Node\* last;  printf("\nTraversal in forward direction \n");  while (node != NULL) {  printf(" %d ", node->data);  last = node;  node = node->next;  }  printf("\nTraversal in reverse direction \n");  while (last != NULL) {  printf(" %d ", last->data);  last = last->prev;  }  } | Fungsi `printList` digunakan untuk mencetak isi dari linked list berganda baik secara maju maupun mundur. Pertama, fungsi mencetak nilai dari setiap simpul dalam penelusuran maju, dimulai dari simpul pertama hingga terakhir, dengan setiap iterasi menunjuk ke simpul berikutnya. Selanjutnya, fungsi mencetak nilai dari setiap simpul dalam penelusuran mundur, dimulai dari simpul terakhir hingga simpul pertama, dengan setiap iterasi menunjuk ke simpul sebelumnya. Hal ini memungkinkan pengguna untuk melihat isi linked list dari kedua arah. |
| 5 | 64 – 74 | int main() {    struct Node\* head = NULL;  push(&head, 6);  push(&head, 5);  push(&head, 2);  insertAfter(head->next, 5);  printf("Created DLL is: ");  printList(head);  return 0;  } | Dalam fungsi `main`, program dimulai dengan linked list berganda kosong, yang direpresentasikan oleh pointer `head` yang diatur menjadi `NULL`. Tiga nilai (6, 5, dan 2) kemudian ditambahkan ke linked list menggunakan fungsi `push`, dengan nilai-nilai ini dimasukkan ke depan linked list. Selanjutnya, fungsi `insertAfter` digunakan untuk menyisipkan nilai 5 setelah simpul kedua dalam linked list. Setelah itu, isi dari linked list dicetak menggunakan fungsi `printList`, sehingga hasilnya adalah pencetakan nilai-nilai dari simpul pertama hingga terakhir, beserta penelusuran mundur dari simpul terakhir hingga pertama. |

**3. Insertion at End**

|  |  |  |  |
| --- | --- | --- | --- |
| Nomor  Program | Baris Program | Petikan Source Code | Penjelasan |
| 1 | 5 – 9 | struct Node {  int data;  struct Node \*next;  struct Node \*prev;  }; | Kode tersebut mendefinisikan struktur untuk simpul dalam linked list berganda. Setiap simpul menyimpan data dan memiliki pointer ke simpul berikutnya dan sebelumnya, memungkinkan penelusuran maju dan mundur dalam daftar. |
| 2 | 12 – 26 | void push(struct Node\*\* head\_ref, int new\_data) {  struct Node\* new\_node = (struct Node\*)malloc(sizeof(struct Node));  new\_node->data = new\_data;  new\_node->next = (\*head\_ref);  new\_node->prev = NULL;  if ((\*head\_ref) != NULL) {  (\*head\_ref)->prev = new\_node;  }  (\*head\_ref) = new\_node;  } | Fungsi `push` digunakan untuk menambahkan simpul baru ke depan linked list berganda. Pertama, fungsi mengalokasikan memori untuk simpul baru menggunakan `malloc`, kemudian nilai `new\_data` dimasukkan ke dalam simpul tersebut. Pointer `next` dari simpul baru diatur untuk menunjuk ke simpul yang saat ini menjadi kepala (head) dari linked list. Pointer `prev` diatur menjadi `NULL`, menunjukkan simpul baru tersebut akan menjadi kepala. Selanjutnya, jika linked list tidak kosong, pointer `prev` dari kepala saat ini diatur untuk menunjuk ke simpul baru. Akhirnya, kepala dari linked list diperbarui agar menunjuk ke simpul baru yang baru saja ditambahkan. |
| 3 | 29 – 52 | void append(struct Node\*\* head\_ref, int new\_data) {  struct Node\* new\_node = (struct Node\*)malloc(sizeof(struct Node));  struct Node\* last = \*head\_ref;  new\_node->data = new\_data;  new\_node->next = NULL;  if (\*head\_ref == NULL) {  new\_node->prev = NULL;  \*head\_ref = new\_node;  return;  }  while (last->next != NULL) {  last = last->next;  }  last->next = new\_node;  new\_node->prev = last;  return;  } | Fungsi `append` digunakan untuk menambahkan simpul baru ke ujung (akhir) linked list berganda. Pertama, fungsi mengalokasikan memori untuk simpul baru menggunakan `malloc`. Kemudian, variabel `last` diinisialisasi dengan pointer ke kepala linked list. Nilai dari `new\_data` dimasukkan ke dalam simpul baru tersebut. Pointer `next` dari simpul baru diatur menjadi `NULL`, menandakan bahwa simpul baru akan menjadi simpul terakhir dalam linked list. Selanjutnya, jika linked list masih kosong (tidak ada kepala), simpul baru tersebut akan diatur sebagai kepala dan fungsi selesai. Jika linked list tidak kosong, maka dilakukan penelusuran hingga mencapai simpul terakhir, dan kemudian simpul baru disambungkan di belakang simpul terakhir tersebut. Pointer `prev` dari simpul baru diatur untuk menunjuk ke simpul terakhir, sehingga linked list terhubung ke simpul baru yang ditambahkan. |
| 4 | 55 – 68 | void printList(struct Node\* node) {  struct Node\* last;  printf("\nTraversal in forward direction \n");  while (node != NULL) {  printf(" %d ", node->data);  last = node;  node = node->next;  }  printf("\nTraversal in reverse direction \n");  while (last != NULL) {  printf(" %d ", last->data);  last = last->prev;  }  } | Fungsi `printList` digunakan untuk mencetak isi dari linked list berganda baik secara maju maupun mundur. Pertama, fungsi mencetak nilai dari setiap simpul dalam penelusuran maju, dimulai dari simpul pertama hingga terakhir. Saat melakukan iterasi, setiap simpul yang dilewati akan ditandai sebagai simpul terakhir. Setelah penelusuran maju selesai, fungsi mencetak nilai dari setiap simpul dalam penelusuran mundur, dimulai dari simpul terakhir hingga pertama. Dalam hal ini, variabel `last` digunakan sebagai penanda simpul terakhir yang akan digunakan untuk penelusuran mundur. Hal ini memungkinkan pengguna untuk melihat isi linked list dari kedua arah, yaitu maju dan mundur. |
| 5 | 70 – 84 | int main() {  /\* Start with the empty list \*/  struct Node\* head = NULL;  // Append nodes to the list  append(&head, 6);  push(&head, 7);  push(&head, 1);  append(&head, 4);  printf("Created DLL is: ");  printList(head);  return 0;  } | Dalam fungsi `main`, program dimulai dengan linked list berganda kosong, yang direpresentasikan oleh pointer `head` yang diatur menjadi `NULL`. Beberapa simpul kemudian ditambahkan ke linked list menggunakan fungsi `append` dan `push`, dengan nilai-nilai yang ditambahkan ke berbagai posisi dalam linked list. Setelah semua simpul ditambahkan, isi dari linked list dicetak menggunakan fungsi `printList`, sehingga hasilnya adalah pencetakan nilai-nilai dari simpul pertama hingga terakhir, bersamaan dengan penelusuran mundur dari simpul terakhir hingga pertama. |

**4. Insertion before given node**

|  |  |  |  |
| --- | --- | --- | --- |
| Nomor  Program | Baris Program | Petikan Source Code | Penjelasan |
| 1 | 4 – 8 | struct Node {  int data;  struct Node\* next;  struct Node\* prev;  }; | Kode tersebut mendefinisikan struktur untuk simpul dalam linked list berganda. Setiap simpul menyimpan data dan memiliki pointer ke simpul berikutnya dan sebelumnya, memungkinkan penelusuran maju dan mundur dalam daftar. |
| 2 | 10 – 22 | void push(struct Node\*\* head\_ref, int new\_data) {  struct Node\* new\_node = (struct Node\*)malloc(sizeof(struct Node));  if (new\_node == NULL) {  printf("Memory allocation failed\n");  return;  }  new\_node->data = new\_data;  new\_node->next = (\*head\_ref);  new\_node->prev = NULL;  if ((\*head\_ref) != NULL)  (\*head\_ref)->prev = new\_node;  (\*head\_ref) = new\_node;  } | Fungsi `push` digunakan untuk menambahkan sebuah simpul baru ke depan linked list berganda. Pertama, fungsi mengalokasikan memori untuk simpul baru menggunakan `malloc`, dan jika alokasi memori gagal, fungsi akan mencetak pesan kesalahan dan mengembalikan eksekusi. Kemudian, nilai `new\_data` dimasukkan ke dalam simpul baru tersebut. Pointer `next` dari simpul baru diarahkan ke simpul yang saat ini menjadi kepala (head) dari linked list, dan pointer `prev` diatur menjadi `NULL`, menandakan simpul baru tersebut akan menjadi kepala. Selanjutnya, jika linked list tidak kosong, pointer `prev` dari kepala saat ini diubah untuk menunjuk ke simpul baru. Akhirnya, kepala linked list diperbarui agar menunjuk ke simpul baru yang baru saja ditambahkan. |
| 3 | 24 – 46 | void insertBefore(struct Node\*\* head\_ref, struct Node\* next\_node, int new\_data) {  if (next\_node == NULL) {  printf("The given next node cannot be NULL\n");  return;  }  struct Node\* new\_node = (struct Node\*)malloc(sizeof(struct Node));  if (new\_node == NULL) {  printf("Memory allocation failed\n");  return;  }  new\_node->data = new\_data;  new\_node->prev = next\_node->prev;  new\_node->next = next\_node;  if (new\_node->prev != NULL)  new\_node->prev->next = new\_node;  else  (\*head\_ref) = new\_node;  next\_node->prev = new\_node;  } | Fungsi `insertBefore` digunakan untuk menyisipkan sebuah simpul baru sebelum simpul tertentu dalam linked list berganda. Pertama, fungsi memeriksa apakah simpul yang diberikan sebagai simpul selanjutnya tidak boleh `NULL`, dan jika iya, maka akan mencetak pesan kesalahan dan mengembalikan eksekusi. Selanjutnya, fungsi mengalokasikan memori untuk simpul baru menggunakan `malloc`, dan jika alokasi memori gagal, akan mencetak pesan kesalahan dan mengembalikan eksekusi. Nilai `new\_data` dimasukkan ke dalam simpul baru tersebut. Pointer `prev` dari simpul baru diatur untuk menunjuk ke simpul sebelum simpul selanjutnya, dan pointer `next` diatur untuk menunjuk ke simpul selanjutnya itu sendiri. Jika simpul sebelum simpul selanjutnya tidak kosong, maka pointer `next` dari simpul tersebut diubah untuk menunjuk ke simpul baru. Jika simpul tersebut adalah kepala linked list, maka kepala linked list diperbarui agar menunjuk ke simpul baru. Akhirnya, pointer `prev` dari simpul selanjutnya diatur untuk menunjuk ke simpul baru yang baru saja ditambahkan. |
| 4 | 48 – 61 | void printList(struct Node\* node) {  struct Node\* last = NULL;  printf("\nTraversal in forward direction \n");  while (node != NULL) {  printf(" %d ", node->data);  last = node;  node = node->next;  }  printf("\nTraversal in reverse direction \n");  while (last != NULL) {  printf(" %d ", last->data);  last = last->prev;  }  } | Fungsi `printList` digunakan untuk mencetak isi dari linked list berganda baik secara maju maupun mundur. Pertama, fungsi melakukan penelusuran maju dari simpul pertama hingga terakhir, dengan mencetak nilai dari setiap simpul dalam setiap iterasi. Saat melakukan penelusuran maju, variabel `last` diatur untuk menunjuk ke simpul saat ini, sehingga akan menyimpan referensi ke simpul terakhir setelah penelusuran selesai. Setelah penelusuran maju selesai, fungsi mencetak nilai dari simpul terakhir hingga pertama dalam penelusuran mundur. Hal ini memungkinkan pengguna untuk melihat isi linked list dari kedua arah, yaitu maju dan mundur. |
| 5 | 63 – 74 | void freeList(struct Node\*\* head\_ref) {  struct Node\* current = \*head\_ref;  struct Node\* next;  while (current != NULL) {  next = current->next;  free(current);  current = next;  }  \*head\_ref = NULL;  } | Fungsi `freeList` digunakan untuk membebaskan memori yang dialokasikan untuk setiap simpul dalam linked list berganda. Pertama, fungsi menginisialisasi variabel `current` dengan pointer ke kepala linked list. Kemudian, dalam sebuah loop while, fungsi memproses setiap simpul dalam linked list. Untuk setiap iterasi, fungsi menyimpan pointer ke simpul berikutnya dalam variabel `next`, kemudian membebaskan memori yang dialokasikan untuk simpul saat ini menggunakan fungsi `free`. Setelah itu, variabel `current` diarahkan ke simpul berikutnya untuk iterasi selanjutnya. Proses ini berlanjut hingga seluruh simpul dalam linked list telah dilewati. Terakhir, setelah seluruh simpul telah dibebaskan, pointer kepala linked list diatur kembali menjadi `NULL` untuk menunjukkan bahwa linked list sekarang kosong dan memori yang sebelumnya dialokasikan telah dibebaskan. |
| 6 | 76 – 90 | int main() {  struct Node\* head = NULL;  push(&head, 7);  push(&head, 1);  push(&head, 4);  insertBefore(&head, head->next, 8);  printf("Created Doubly Linked List is: ");  printList(head);  freeList(&head);  return 0;  } | Dalam fungsi `main`, program dimulai dengan linked list berganda kosong, yang direpresentasikan oleh pointer `head` yang diatur menjadi `NULL`. Beberapa simpul kemudian ditambahkan ke linked list menggunakan fungsi `push`, dengan nilai-nilai yang ditambahkan ke depan linked list. Selanjutnya, fungsi `insertBefore` digunakan untuk menyisipkan nilai 8 sebelum simpul kedua dalam linked list. Setelah seluruh simpul ditambahkan, isi dari linked list dicetak menggunakan fungsi `printList`. Setelah selesai, fungsi `freeList` digunakan untuk membebaskan memori yang dialokasikan untuk linked list tersebut, sehingga program dapat mengembalikan memori yang digunakan dan menghindari kebocoran memori. |