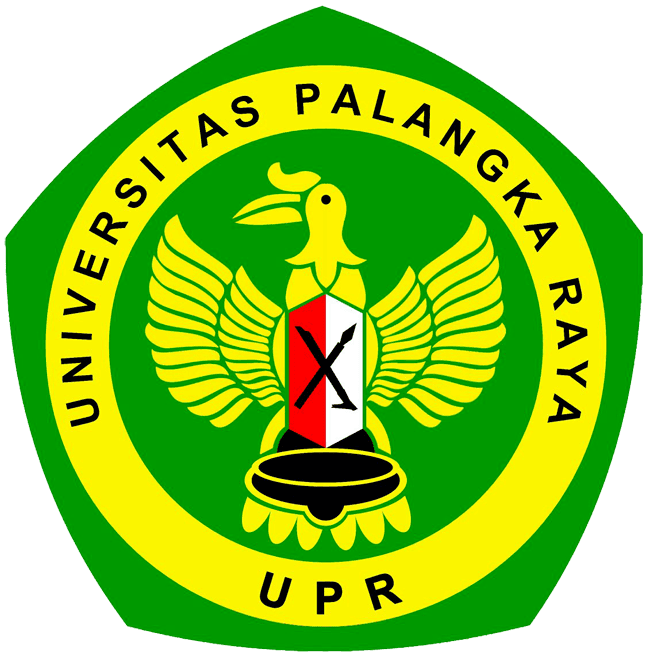
**LAPORAN HASIL PRAKTIKUM**

**STRUKTUR DATA**



|  |  |  |
| --- | --- | --- |
| **NAMA** | **:** | **INDRA FIQI RIPANI** |
| **NIM** | **:** | **213010503002** |
| **KELAS** | **:** | **F** |
| **MODUL** | **:** | **III (*LINKED LIST*)** |

**Program Studi S1 Teknik Informatika**

**Fakultas Teknik**

**Universitas Palangka Raya**

**Palangka Raya, Kalimantan Tengah**

**2022**

**BAB I**

**TUJUAN DAN LANDASAN TEORI**

1. **TUJUAN PRAKTIKUM**
2. Mahasiswa memahami struktur data *linked list*.
3. Mahasiswa mampu menggunakan struktur data *linked list* di dalam menyelesaikan masalah pemrograman.
4. **LANDASAN TEORI**

*Linked list* adalah suatu cara untuk menyimpan data dengan struktur sehingga *programmer* dapat secara otomatis menciptakan suatu tempat baru untuk menyimpan data kapan saja diperlukan. *Linked list* dikenal juga dengan sebutan senarai berantai adalah stuktur data yang terdiri dari urutan *record* data dimana setiap *record* memiliki *field* yang menyimpan alamat/referensi dari *record* selanjutnya (di dalam urutan). Elemen data yang dihubungkan dengan *link* pada *linked list* disebut Node. Biasanya dalam suatu linked list, terdapat istilah *head* dan *tail*.

Jenis *linked list* (yang akan dipelajari) adalah:

1. *Single Linked List*
2. *Double Linked List*
3. *Circular Linked List*
4. *Multiple Linked List*

Ada 5 proses dasar dalam *linked list*:

1. Proses Inisialisasi

* Proses awal 🡪 menyatakan *linked list* belum ada
* Algoritma:

|  |
| --- |
| First = Null;  Last = Null; |

1. Proses Simpul Baru

* Instruksi:

|  |
| --- |
| P = (simpul\*) malloc(sizeof(simpul)); |

* Algoritma:

|  |
| --- |
| void Buat\_Simpul (int x){  P = (simpul\*) malloc(sizeof(simpul));  if (P!=NULL){  P → Info = x;  }  else cout << “simpul gagal dibuat”;  } |

1. Membuat Simpul Awal

Syarat:

1. *Linked list* belum ada
2. Sudah ada simpul yang akan dijadikan simpul awal

Algoritma:

|  |
| --- |
| void Awal (){  First = P;  Last = P;  P → Link = NULL;  } |

1. Menambahkan Simpul Baru ke dalam *Linked List* (*Insert*)

Syarat:

1. *Linked list* sudah ada.
2. Sudah ada simpul yang akan ditambahkan *linked list*.
3. *Insert* Kanan/Akhir

Algoritma:

|  |
| --- |
| void Ins\_Akhir (){  Last → Link = P;  Last = P;  P → Link = NULL;  } |

1. *Insert* Kiri/Awal

Algoritma:

|  |
| --- |
| void Ins\_Awal (){  P → Link = First;  First = P;  } |

1. *Insert* Tengah

Algoritma:

|  |
| --- |
| void Ins\_Tengah (){  P → Link = Q → Link;  Q → Link = P;  } |

1. Menghapus Sebuah Simpul dari *Linked List* (Delete)

Syarat:

1. *Linked list* sudah ada.
2. *Delete* Kanan/Akhir

Algoritma:

|  |
| --- |
| void Del\_Akhir (){  Free (Last);  Last = Q;  Last → Link = NULL;  } |

1. *Delete* Kiri/Awal

Algoritma:

|  |
| --- |
| void Del\_Awal (){  Q = First;  First = Q → Link;  Free (Q);  } |

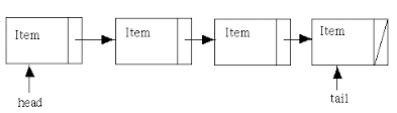
1. *Delete* Tengah

Algoritma:

|  |
| --- |
| void Del\_Tengah (){  R = Q → Link;  Q → Link = R → Link;  Free (R);  } |

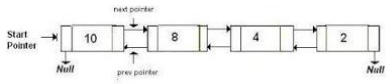
***Single Linked List***

*Single Linked List* merupakan suatu *linked list* yang hanya memiliki satu variabel *pointer* saja. Di mana pointer tersebut menunjuk ke node selanjutnya. Biasanya *field* pada *tail* menunjuk ke NULL.



***Double Linked List***

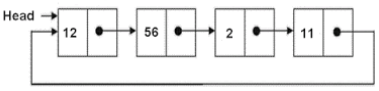
*Double Linked List* merupakan suatu *linked list* yang memiliki dua variabel *pointer* yaitu *pointer* yang menunjuk ke node selanjutnya dan *pointer* yang menunjuk ke node sebelumnya. Setiap *head* dan *tail*nya juga menunjuk ke NULL.



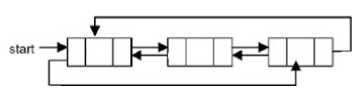
***Circular Linked List***

*Circular Linked List* merupakan suatu *linked list* dimana *tail* (node terakhir) menunjuk ke *head* (node pertama). Jadi tidak ada *pointer* yang menunjuk NULL. Ada 2 jenis *Circular Linked List*, yaitu:

* *Circular Single Linked List*

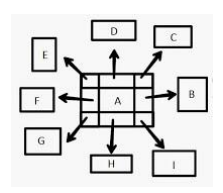


* *Circular Double Linked List*



***Multiple Linked List***

*Multiple Linked List* merupakan suatu *linked list* yang memiliki lebih dari 2 buat variabel pointer.



**BAB II**

**PROSEDUR DAN PEMBAHASAN**

1. **PROSEDUR**
2. Buatlah sebuah *linked list non circular* yang berisi nama lengkap dan nim anda!
3. Buatlah sebuah program dengan menggunakan *single linked list non circular* dengan fungsi-fungsi (buat dengan menggunakan menu):

* Menambah data (dari depan dan dari belakang)
* Menghapus data (dari depan dan dari belakang)
* Mencetak data

1. Buat sebuah program *double linked list* dengan fungsi-fungsi (buat dengan menggunkan menu):

* Menambahkan data (dari depan sesudah simpul dan dari belakang sesudah simpul)
* Menghapus data (dari tengah)
* Data yang dimasukkan langsung ke tampil dalam menu

1. **PEMBAHASAN**

Program pertama adalah pengaplikasian *single link list non circular* dalam pengelolaan data mahasiswa. Pada program harus terdapat perintah atau fungsi untuk menambahkan data dari depan dan belakang, menghapus data dari depan dan belakang, dan mencetak keseluruhan data. Berikut merupakan *source code* program.

|  |
| --- |
| #include <iostream>  #include <limits>  #include <conio.h>  using namespace std;  struct node{  string nama, nim;  node \*next;  };  node \*head, \*tail; |

Penjelasan:

Dideklarasikan “#include <iostream>”, “#include <limits>”, dan “#include <conio.h>” pada bagian *header.* Fungsinya adalah agar program dapat mengek-sekusi perintah-perintah yang terdapat di dalam *library* tersebut, seperti “cin”, “cout”, “getch”, dan lain-lain. Lalu “using namespace std;” berfungsi agar program dapat menggunakan semua fungsi yang akan digunakan di dalam program. Selanjutnya dideklarasikan suatu struct dengan nama ‘node’ yang di dalamnya terdapat variabel ‘nama’ dan ‘nim’ yang bertipe data *string* dan ‘next’ yang bertipe struct itu sendiri. Fungsi dari ‘next’ ini adalah sebagai pointer di dalam *linked list* yang akan kita buat nanti. Terakhir dideklarasikan variabel ‘\*head’ dan ‘\*tail’ bertipe ‘node’ yang merupakan pangkal dan ujung dari data di dalam suatu *linked list*.

|  |
| --- |
| void printAll(){  node \*bantu;  if(head == NULL){  cout << "\nBelum ada data.\n";  } else {  bantu = head;  while(bantu != NULL){  cout << "\nNAMA \t: " << bantu -> nama << endl;  cout << "NIM \t: " << bantu -> nim << endl;  bantu = bantu -> next;  }  }  cout << "\nPress Any Button to Continue...";  getch();  } |

Penjelasan:

Agar mempermudah *programmer* dalam membuat suatu program, maka dibuatlah prosedur-prosedur. Prosedur ‘printAll’ adalah prosedur yang apabila dipanggil di dalam program utama, dia akan mencetak semua data yang sudah diinputkan oleh *user*. Pada prosedur ini dideklarasikan ‘\*bantu’ yang bertipe ‘node’. Jika pangkal data masih kosong, maka program mencetak “Belum ada data.” pada layar. Jika pangkal data tidak kosong, maka program akan mencetak data dari pangkal data sampai ujung data. Algoritma program tersebut dapat dilakukan dengan kombinasi percabangan ‘if - else’ dan perulangan ‘while’. Lalu ada fungsi ‘getch’ agar layar menahan program pada tampilan yang ada sampai *user* menekan tombol apa pun pada *keyboard*nya.

|  |
| --- |
| void inputDepan(){  node \*baru;  baru = new node;  cin.ignore(numeric\_limits<streamsize>::max(), '\n');  cout << "\n=== PROSES MEMASUKKAN DATA ===\n\n";  cout << "Masukkan Nama\t : "; getline(cin, baru -> nama);  cout << "Masukkan NIM \t : "; getline(cin, baru -> nim);  baru -> next = NULL;  if(head == NULL){  head = tail = baru;  head -> next = NULL;  } else {  baru -> next = head; //pointer baru ke head sebelumnya  head = baru;  }  cout << "\nData telah dimasukkan\n\n" << "== DATA TERBARU  ==";  printAll();  } |

Penjelasan:

Selanjutnya dibuat prosedur ‘inputDepan’. Prosedur ini berfungsi untuk membantu *user* agar dapat memasukkan suatu data dari depan. Dideklarasikan ‘\*baru’ bertipe ‘node’ yang akan menampung objek-objek ‘node’. Program akan meminta *user* untuk menginputkan nama dan NIM mahasiswa yang akan menjadi pangkal data. Setelah data diinputkan, maka *pointer* ‘baru’ akan akan mengarahkannya ke *NULL* atau kosong. Jika data masih kosong, maka data yang diinputkan akan menjadi pangkal dan ujung data, sehingga *pointer*nya akan mengarah ke *NULL*. Jika sudah ada data yang diinputkan, maka data baru yang dimasukkan akan menjadi pangkal data yang baru dan *pointer*nya akan mengarah pada pangkal data sebelumnya. Lalu, program akan langsung mence-tak semua data dengan memanggil prosedur ‘printAll’.

|  |
| --- |
| void inputBelakang(){  node \*baru;  baru = new node;  cin.ignore(numeric\_limits<streamsize>::max(), '\n');  cout << "\n=== PROSES MEMASUKKAN DATA ===\n\n";  cout << "Masukkan Nama\t : "; getline(cin, baru -> nama);  cout << "Masukkan NIM \t : "; getline(cin, baru -> nim);  baru -> next = NULL;  if(head == NULL){  head = baru;  } else {  tail -> next = baru; //ekor ke node baru  }  tail = baru;  tail -> next = NULL;  cout << "\nData telah dimasukkan\n\n" << "== DATA TERBARU  ==";  printAll();  } |

Penjelasan:

Prosedur ‘inputBelakang’ berfungsi untuk membantu *user* memasukkan data dari belakang. Dideklarasikan ‘\*baru’ yang bertipe ‘node’ dan dibuat objek ‘baru’ dari ‘node’. Program akan meminta *user* untuk memasukkan nama dan NIM mahasiswa. Lalu *pointer* dari ‘baru’ diarahkan ke *NULL*. Jika data masih kosong, maka data yang diinputkan akan menjadi *head.* Jika data sudah ada sebelumnya, maka *pointer* dari *tail* yang ada akan diarahkan ke data yang baru dimasukkan. Sehingga data terbaru akan menjadi *tail* dan *pointer* dari *tail* yang baru akan diarahkan ke *NULL*. Lalu, program akan langsung mencetak semua data dengan memanggil prosedur ‘printAll’.

|  |
| --- |
| void hapusDepan(){  node \*hapus;  if (head == NULL){  cout << "Data masih Kosong\n";  } else {  cout<< "\nData telah dihapus\n\n" << "== DATA TERBARU  ==\n";  hapus = head;  head = head -> next;  delete hapus;  }  printAll();  } |

Penjelasan:

Prosedur ‘hapusDepan’ berfungsi untuk membantu *user* menghapus data dari depan. Dideklarasikan ‘\*hapus’ yang bertipe ‘node’. Jika data belum ada, maka program akan mencetak “Data masih kosong.”. Jika data sudah ada, maka nilai *head* akan dimasukkan ke ‘hapus’ dan *head* yang ada akan mengubah nilai-nya menjadi data setelah *head* dan data ‘hapus’ akan dihapus. Lalu program akan mencetak semua data dengan memanggil prosedur ‘printAll’.

|  |
| --- |
| void hapusBelakang(){  node \*bantu, \*hapus;  if(head == NULL){  cout << "Data masih Kosong\n";  } else if(head == tail){  cout << "\nData telah dihapus\n";  hapus = head;  head = head -> next;  delete hapus;  } else {  cout<< "\nData telah dihapus\n\n" << "== DATA TERBARU  ==\n";  bantu = head;  hapus = tail;  while(bantu -> next != tail){  bantu = bantu -> next;  }  tail = bantu;  tail -> next = NULL;  delete hapus;  }  printAll();  } |

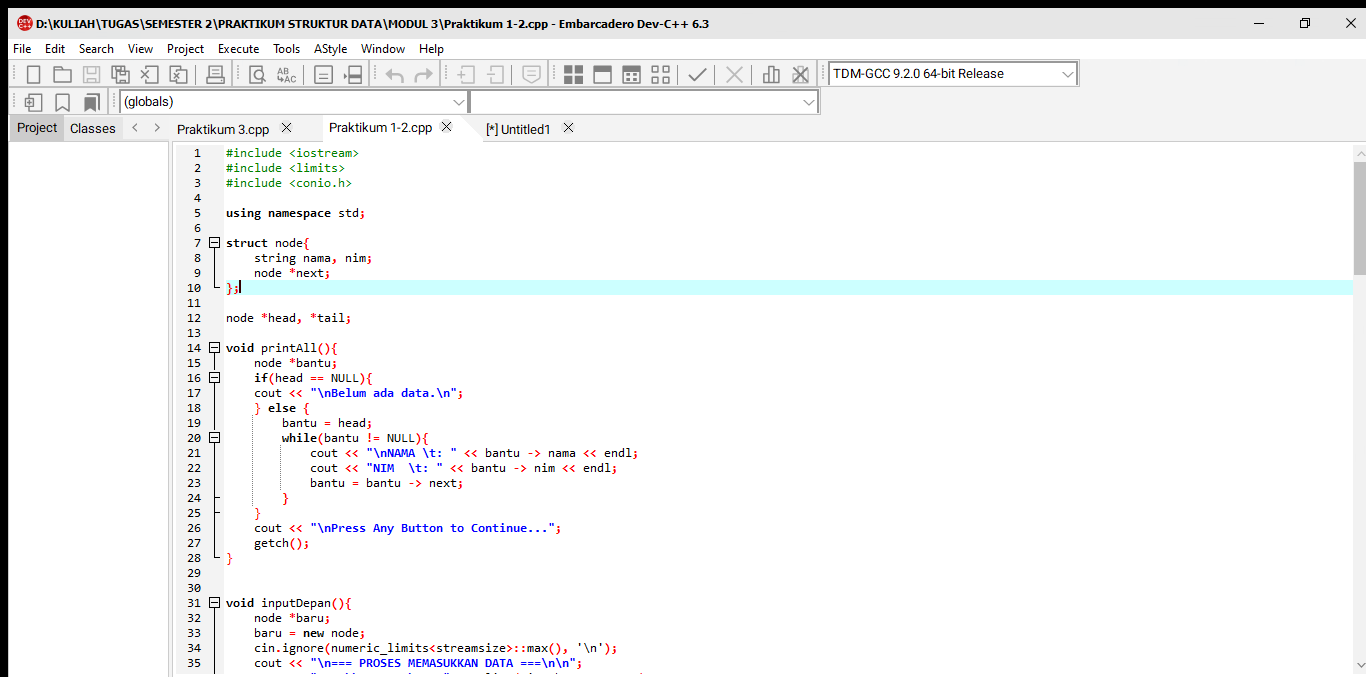
Penjelasan:

Prosedur ‘hapusBelakang’ berfungsi untuk membantu *user* agar dapat menghapus suatu data dari belakang. Dideklarasikan ‘\*bantu’ dan ‘\*hapus’ yang bertipe ‘node’. Jika data masih kosong, maka program akan mencetak “Data masih kosong”. Jika *head* sama dengan *tail,* maka *head* akan memasuk-kan nilainya ke ‘hapus’ dan *head* yang ada akan merubah nilainya menjadi data setelah *head* awal. Lalu ‘hapus’ akan dihapus. Selain dari dua kondisi sebelum-nya, maka program akan memasukkan nilai *head* ke ‘hapus’ dan nilai *tail* ke ‘hapus’. Selama nilai setelah ‘bantu’ bukan *tail,* maka program akan terus men-geser nilai ‘bantu’. Setelah data di sebelah ‘bantu’ adalah *tail* maka perulangan pun berhenti. Lalu nilai *tail* akan berubah menjadi ‘bantu’, dan *pointer* pada *tail* yang baru akan menunjuk pada *NULL*. Setelah itu, ‘hapus’ akan dihapus dan program akan mencetak semua data.

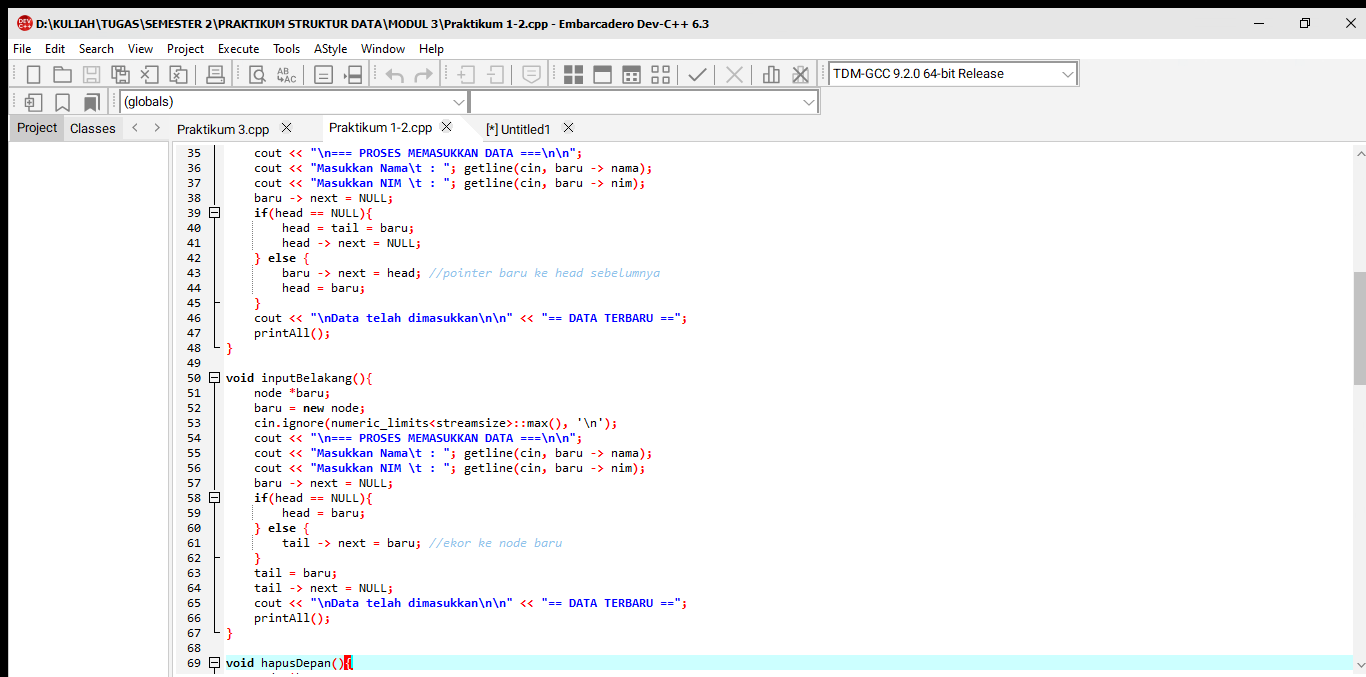
|  |
| --- |
| int main(){  int pil;  do{  system("cls");  cout << "==================================\n"  << " MENU UTAMA LINKED LIST\n"  << "================================  ==\n"  << "== 1. Input Data dari Depan ==\n"  << "== 2. Input Data dari Belakang ==\n"  << "== 3. Hapus Data dari Depan ==\n"  << "== 4. Hapus Data dari Belakang ==\n"  << "== 5. Tampilkan Semua Data ==\n"  << "== 6. Keluar ==\n\n"  << "Pilih Menu \t: ";  cin >> pil;  switch (pil){  case 1:{  inputDepan();  break;  }  case 2:{  inputBelakang();  break;  }  case 3:{  hapusDepan();  break;  }  case 4:{  hapusBelakang();  break;  }  case 5:{  printAll();  break;  }  case 6:{  cout << "\nProgram Selesai";  break;  }  default:{  cout << "\nMasukan tidak valid.\n" << endl;  cout << "\nPress Any Button to Continue...";  getch();  }  }  } while (pil != 6);  return 0;  } |

Penjelasan:

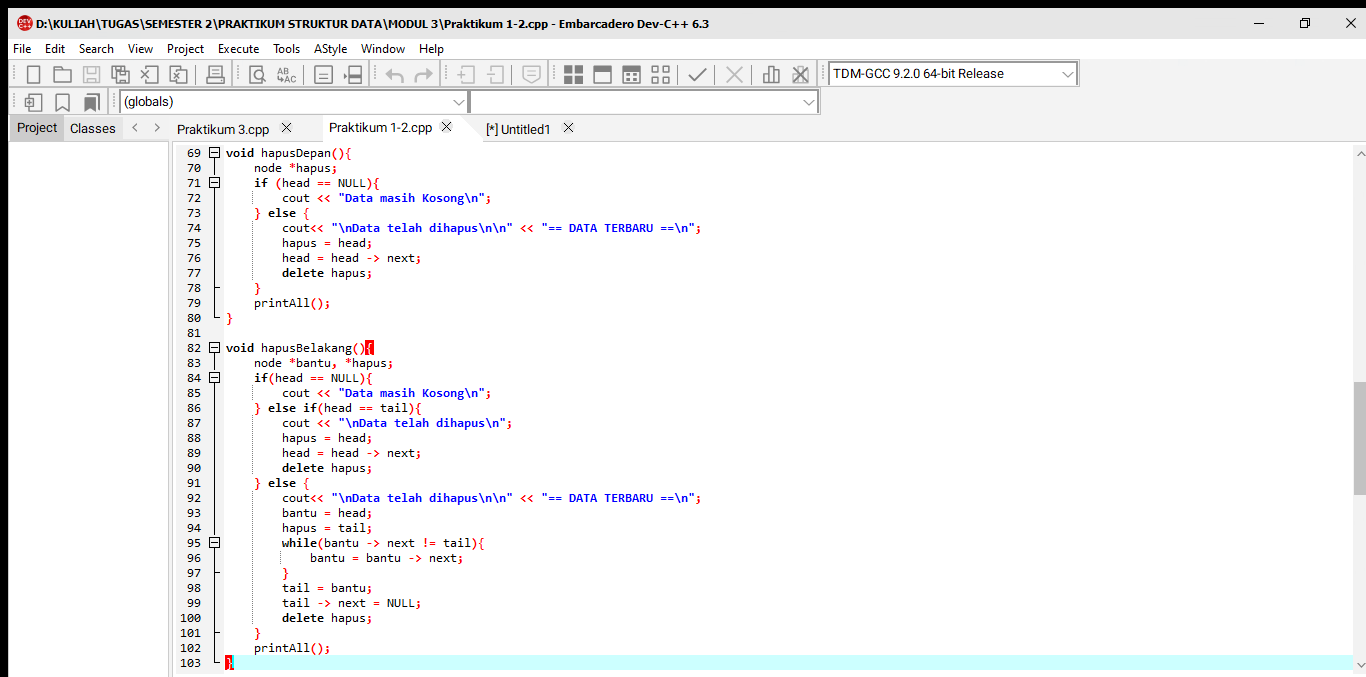
Bagian ini disebut *method main* atau program utama. Dideklarasikan variabel ‘pil’ yang bertipe data *integer* untuk menampung nilai dari inputan *user* saat memilih menu nantinya. Dilakukan perulangan menggunakan ‘do-while’ dengan kondisi perulangan akan berhenti saat *user* memasukkan ‘6’ saat memilih menu. Program akan mencetakkan ke layar seperti yang ada di dalam *source code* dan meminta masukkan pengguna. Percabangan yang digunakan adalah ‘switch(case)’. Karena telah dibuat prosedur-prosedur sebelum *method main*, maka pada program utama cukup memanggil prosedur-prosedurnya saja sesuai dengan fungsinya.



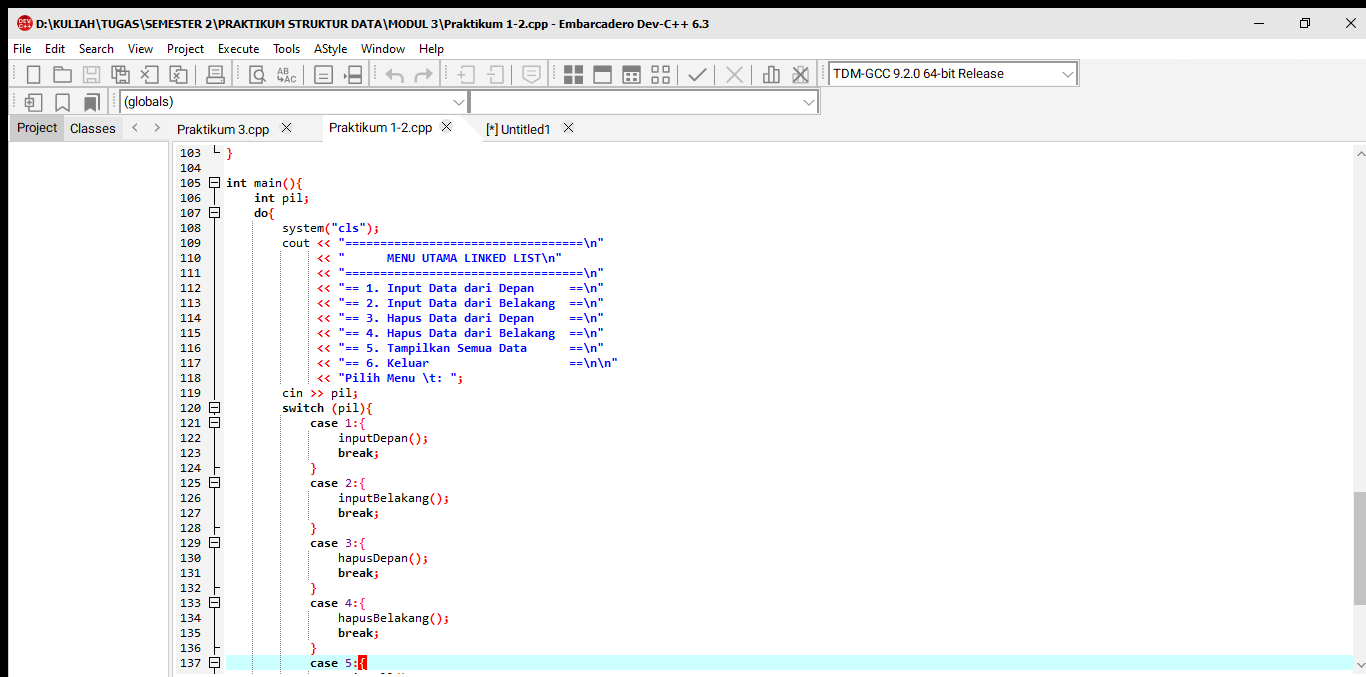
Gambar 1.1 *Source Code* Program (1)



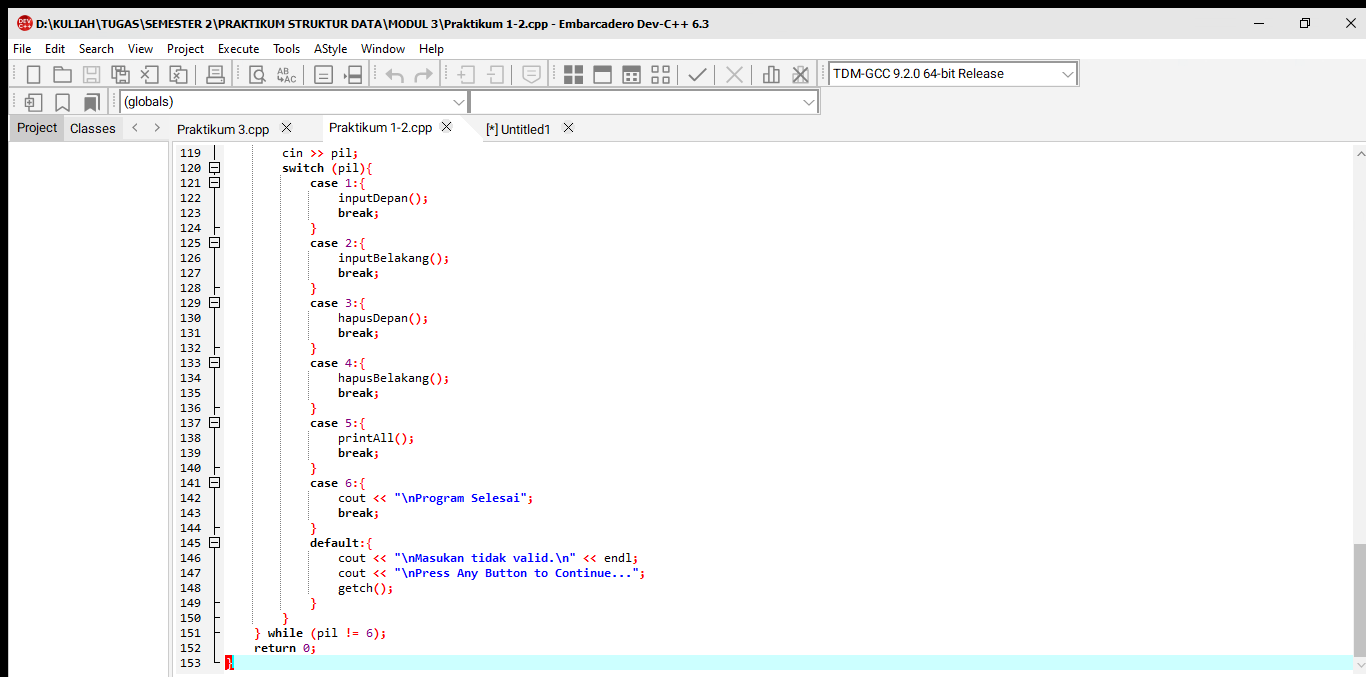
Gambar 1.2 *Source Code* Program (2)



Gambar 1.3 *Source Code* Program (3)



Gambar 1.4 *Source Code* Program (4)



Gambar 1.5 *Source Code* Program (5)



Gambar 2.1 *Output* Menu 1 (1)



Gambar 2.2 *Output* Menu 1 (2)



Gambar 2.3 *Output* Menu 2 (1)



Gambar 2.4 *Output* Menu 2 (2)



Gambar 2.5 *Output* Menu 3



Gambar 2.6 *Output* Menu 4



Gambar 2.7 *Output* Menu 5



Gambar 2.8 *Output* Menu 6

Program kedua adalah pengaplikasian *double linked list* dalam pengelolaan data. *Double linked list* merupakan suatu *linked list* yang memiliki dua variabel *pointer* yaitu *pointer* yang menunjuk ke node selanjutnya dan juga *pointer* yang menunjuk ke node sebelumnya. Program yang dibuat memiliki fungsi menamb-ahkan data dari depan sesudah simpul dan dari belakang sesudah simpul, meng-hapus data tengah, dan data langsung tampil pada menu. Berikut merupakan *source code* program.

|  |
| --- |
| #include<iostream>  #include <conio.h>  using namespace std;  struct node{  int x;  node\* next;  node\* prev;  };  node \*head, \*tail;  void init(){  head = NULL;  tail = NULL;  } |

Penjelasan:

Pada bagian *header* dideklarasikan ‘#include<iostream>’ dan ‘#include <conio.h>’ agar program dapat mengeksekusi fungsi-fungsi di dalam *library*-nya. “using namespace std;” berfungsi agar program dapat menggunakan semua fungsi yang akan digunakan di dalam program. Selanjutnya dideklarasikan suatu struct dengan nama ‘node’ yang di dalamnya terdapat variabel ‘x’ yang bertipe data *integer* dan ‘\*prev’ dan ‘\*next’ yang bertipe ‘node’ itu sendiri seba-gai *pointer* penunjuk pada *linked list* yang akan dibuat. Lalu dideklarasikan ‘\*head’ dan ‘\*tail’ yang bertipe ‘node’ sebagai pangkal dan ujung data. Setelah itu dibuat prosedur ‘init’ untuk memberi nilai *NULL* pada ‘head’ dan ‘tail’.

|  |
| --- |
| void insertFirst(int data){  node \*baru;  baru = new node;  if(head == NULL){  head = baru;  baru -> prev = NULL;  baru -> x = data;  baru -> next = NULL;  tail = baru;  } else {  baru -> next = head;  baru -> x = data;  baru -> prev = NULL;  head -> prev = baru;  head = baru;  }  } |

Penjelasan:

Apabila data masih kosong, maka *pointer* ‘next’ dan ‘prev’ data akan menunjuk ke *NULL*. Sehingga data yang baru diinputkan akan menjadi *head* dan *tail* dari *linked list* tersebut. Apabila sebelumnya sudah ada data, maka *pointer* ‘next’ data terbaru akan menunjuk pada *head* dan *pointer* ‘prev’ *head* akan menunjuk data terbaru. Sehingga *head* terbaru adalah data yang baru dimasukkan. Prosedur ini berfungsi untuk menambahkan data dari depan.

|  |
| --- |
| void insertLast(int data){  node \*baru;  baru = new node;  baru -> x = data;  if(head == NULL){  head = baru;  baru -> prev = NULL;  baru -> next = NULL;  tail = baru;  } else {  baru -> prev = tail;  tail -> next = baru;  baru -> next = NULL;  tail = baru;  }  } |

Penjelasan:

Jika data masih kosong, maka *pointer* ‘next’ dan ‘prev’ data akan menunjuk ke *NULL*. Sehingga data yang baru diinputkan akan menjadi *head* dan *tail* dari *linked list* tersebut. Namun, jika sudah ada data sebelumnya, maka *pointer* ‘prev’ data terbaru akan menunjuk pada *tail* dan *pointer* ‘next’ *tail* akan menun-juk data terbaru. Lalu dideklarasikan nilai *tail* sama dengan data terbaru. Prose-dur ini berfungsi untuk menambahkan data dari belakang.

|  |
| --- |
| void printList(){  cout << endl;  node \*temp;  if(head == NULL){  cout << "Belum ada data.";  } else {  temp = head;  while(temp != NULL){  cout << temp -> x << " -> ";  temp = temp -> next;  }  }  } |

Penjelasan:

Jika data belum ada, maka program akan mencetak “Belum ada data.”. Jika sudah ada data sebelumnya, maka selama *pointer* ‘next’ data tidak menunjuk pada *NULL* data akan terus dicetak. Prosedur ini berfungsi untuk menampilkan semua data pada program.

|  |
| --- |
| void insertAfter(int old, int data){  node \*baru;  baru = new node;  node \*temp;  temp = head;  if(head == NULL){  cout << "Tidak bisa memasukkan data." << endl;  }  if(head == tail){  if(head -> x != old){  cout << "Tidak bisa memasukkan data." << endl;  }  baru -> x = data;  head -> next = baru;  baru -> next = NULL;  head -> prev = NULL;  baru -> prev = head;  tail = baru;  }  if(tail -> x == data){  baru -> next = NULL;  baru -> prev = tail;  tail -> next = baru;  tail = baru;  }  while(temp -> x != old){  temp = temp -> next;  if(temp == NULL){  cout << "Tidak bisa memasukkan data." << endl;  cout << "Tidak ada data." << endl;  }  }  baru -> next = temp -> next;  baru -> prev = temp;  baru -> x = data;  temp -> next -> prev = baru;  temp -> next = baru;  } |

Penjelasan:

Prosedur ini berfungsi untuk membantu *user* untuk dapat memasukkan data setelah atau di depan suatu data yang telah ada berdasarkan inputan *user*. Prose-dur tidak dapat dieksekusi jika data masih kosong atau hanya ada satu ada. Ini merupakan pengembangan dari pengelolaan data *linked list* di tengah (bukan *head* dan *tail*). Sehingga dia juga tidak bisa mengeksekusi penambahan data setelah *tail.* Penambahan data setelah *tail* sudah ada prosedurnya sendiri. Prose-dur dilakukan dengan mengganti *pointer* ‘next’ data yang di sebelah kirinya diarahkan pada data terbaru, *pointer* ‘prev’ data terbaru diarahkan pada data di sebelah kirinya, *pointer* ‘next’ data terbaru diarahkan ke data di sebelah kanan-nya, dan *pointer* ‘prev’ data di sebelah kanannya diarahkan pada data terbaru.

|  |
| --- |
| void insertBefore(int old, int data){  node \*baru;  baru = new node;  node \*temp;  temp = head;  if(head == NULL){  cout<< "Tidak bisa memasukkan data." << endl;  }  if(head == tail){  if(head -> x != old){  cout << "Tidak bisa memasukkan data." << endl;  }  baru -> x = data;  head -> next = baru;  baru -> next = NULL;  head -> prev = NULL;  baru -> prev = head;  tail = baru;  }  if(tail -> x == data){  baru -> next = tail;  baru -> prev = NULL;  tail -> next = baru;  tail = baru;  }  while(temp -> x != old){  temp = temp -> next;  if(temp == NULL){  cout << "Tidak bisa memasukkan data." << endl;  cout << "Tidak ada data." << endl;  }  }  baru -> prev = temp -> prev;  baru -> next = temp;  baru -> x = data;  temp -> prev -> next = baru;  temp -> prev = baru;  } |

Penjelasan:

Prosedur ini berfungsi untuk membantu *user* untuk dapat memasukkan data sebelum atau di belakang suatu data yang telah ada berdasarkan inputan *user*. Prosedur tidak dapat dieksekusi jika data masih kosong atau hanya ada satu ada. Ini merupakan pengembangan dari pengelolaan data *linked list* di tengah (bukan *head* dan *tail*). Sehingga dia juga tidak bisa mengeksekusi penambahan data sebelum *head*. Penambahan data sebelum *head* sudah ada prosedurnya sendiri. Prosedur dilakukan dengan mengganti *pointer* ‘next’ data yang di sebelah kirinya diarahkan pada data terbaru, *pointer* ‘prev’ data terbaru diarahkan pada data di sebelah kirinya, *pointer* ‘next’ data terbaru diarahkan ke data di sebelah kanannya, dan *pointer* ‘prev’ data di sebelah kanannya diarahkan kepada data terbaru.

|  |
| --- |
| void deleteFirst(){  if(head == NULL){  return;  }  if(head == tail){  node \*hapus;  hapus = head;  head = NULL;  tail = NULL;  delete hapus;  } else {  node \*hapus;  hapus = head;  head = head -> next;  head -> prev = NULL;  delete hapus;  }  } |

Penjelasan:

Prosedur ini berfungsi untuk membantu *user* menghapus suatu data dari depan. Jika belum ada data, maka tidak akan ada yang dihapus. Jika sudah ada data, maka *head* akan dihapus dengan *head* terbaru adalah data setelah *head* se-belumnya. Sehingga *pointer* ‘prev’ dari *head* terbaru akan menunjuk pada NULL. Dalam penghapusan data ini, kita memerlukan variabel bantu, yaitu ‘hapus’.

|  |
| --- |
| void deleteLast(){  if(head==NULL){  return;  }  if(head==tail){  node \*hapus;  hapus = head;  head = NULL;  tail = NULL;  delete hapus;  } else {  node \*hapus;  hapus = tail;  tail = tail -> prev;  tail -> next = NULL;  delete hapus;  }  } |

Penjelasan:

Jika data masih kosong, maka tidak akan ada yang dihapus. Jika *tail* sama dengan *head* (hanya terdapat 1 data) nilai *tail* dan *head* akan dirubah menjadi *NULL* dan akan terhapus data awalnya. Jika bukan kedua kondisi sebelumnya, maka *tail* akan dirubah menjadi data sebelum *tail* dan *pointer* ‘next’ *tail* terbaru akan diarahkan ke *NULL.* Lalu *tail* sebelumnya akan dihapus.

|  |
| --- |
| void deleteMid(int data){  node \*temp;  temp = head;  if(head == tail)  {  if(head -> x != data){  cout << "Tidak dapat menghapus data." << endl;  }  head = NULL;  tail = NULL;  delete temp;  }  if(head -> x == data){  head = head -> next;  head -> prev = NULL;  delete temp;  } else if(tail -> x == data){  temp = tail;  tail = tail -> prev;  tail -> next = NULL;  delete temp;  }  while(temp -> x != data){  temp = temp -> next;  if(temp == NULL){  cout << "data not found" << endl;  }  }  temp -> next -> prev = temp -> prev;  temp -> prev -> next = temp -> next;  delete temp;  } |

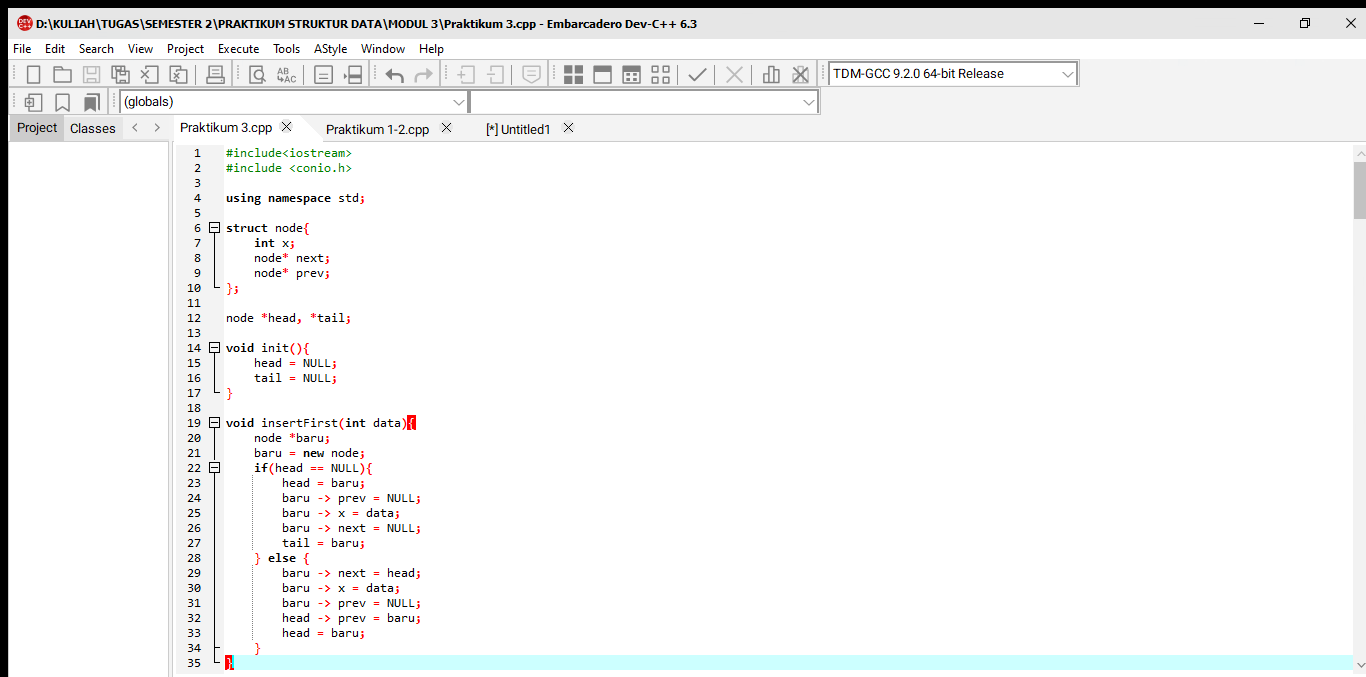
Penjelasan:

Prosedur ini berfungsi untuk menghapus suatu data di dalam *linked list* yang berada di antara *head* dan *tail*. Apabila ingin menghapus *head* atau *tail*, maka sudah ada prosedurnya sendiri. Jika belum ada data, maka tidak akan ada yang dihapus. Jika sudah ada data, maka data yang diinginkan *user* untuk dihapus akan terhapus. Caranya adalah *pointer* ‘next’ data sebelum data yang ingin diha-pus langsung menunjuk data setelah data yang ingin dihapus dan *pointer* ‘prev’ data setelah data yang ingin dihapus langsung menunjuk data sebelum data yang ingin dihapus.

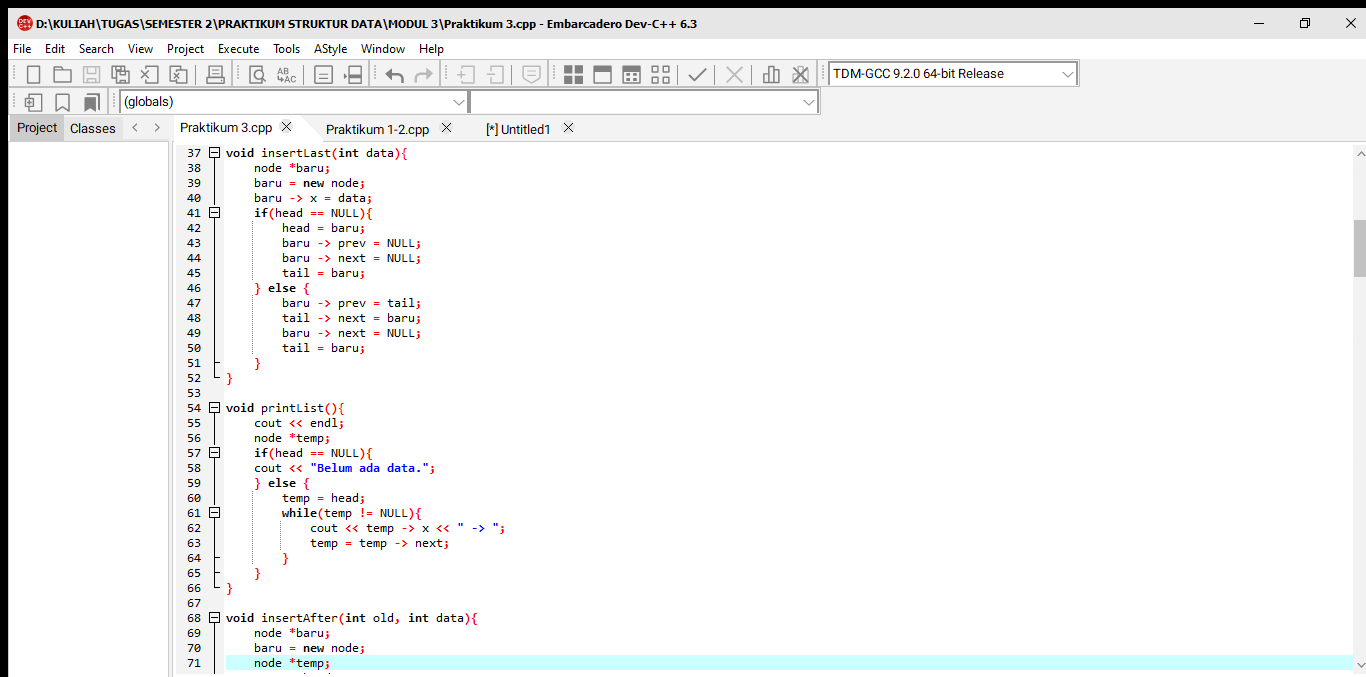
|  |
| --- |
| int main(){  init();  int baru, data, old, pil;  do{  system("cls");  cout << "==================================\n"  << "= MENU UTAMA LINKED LIST =\n"  << "==================================\n"  << "== 1. Input Data dari Depan ==\n"  << "== 2. Input Data dari Belakang ==\n"  << "== 3. Input Data Setelah Data ==\n"  << "== 4. Input Data Sebelum Data ==\n"  << "== 5. Hapus Data dari Depan ==\n"  << "== 6. Hapus Data dari Belakang ==\n"  << "== 7. Hapus Data dari Tengah ==\n"  << "== 8. Keluar ==\n"  << "==================================\n";  printList();  cout << "\n\nPilih Menu\t: ";  cin >> pil;  switch (pil){  case 1:  cout << "\nMasukkan data : ";  cin >> data;  insertFirst(data);  printList();  getch();  break;  case 2:  cout << "\nMasukkan data : ";  cin >> data;  insertLast(data);  printList();  getch();  break;  case 3:  printList();  cout << endl;  cout << "\nMasukkan setelah : ";  cin >> old;  cout << "Masukkan data baru : ";  cin >> baru;  insertAfter(old, baru);  printList();  getch();  break;  case 4:  printList();  cout << endl;  cout << "\nMasukkan sebelum : ";  cin >> old;  cout << "Masukkan data baru : ";  cin >> baru;  insertBefore(old, baru);  printList();  getch();  break;  case 5:  deleteFirst();  printList();  getch();  break;  case 6:  deleteLast();  printList();  getch();  break;  case 7:  printList();  cout << endl;  int data;  cout << "\nMasukkan data yang ingin dihapus: ";  cin >> data;  deleteMid(data);  printList();  getch();  break;  case 8:  cout << "\nProgram Selesai";  break;  default:  cout << "\nMasukan tidak valid.\n" << endl;  cout << "\nPress Any Button to Continue...";  getch();  }  } while (pil != 8);  return 0;  } |

Penjelasan:

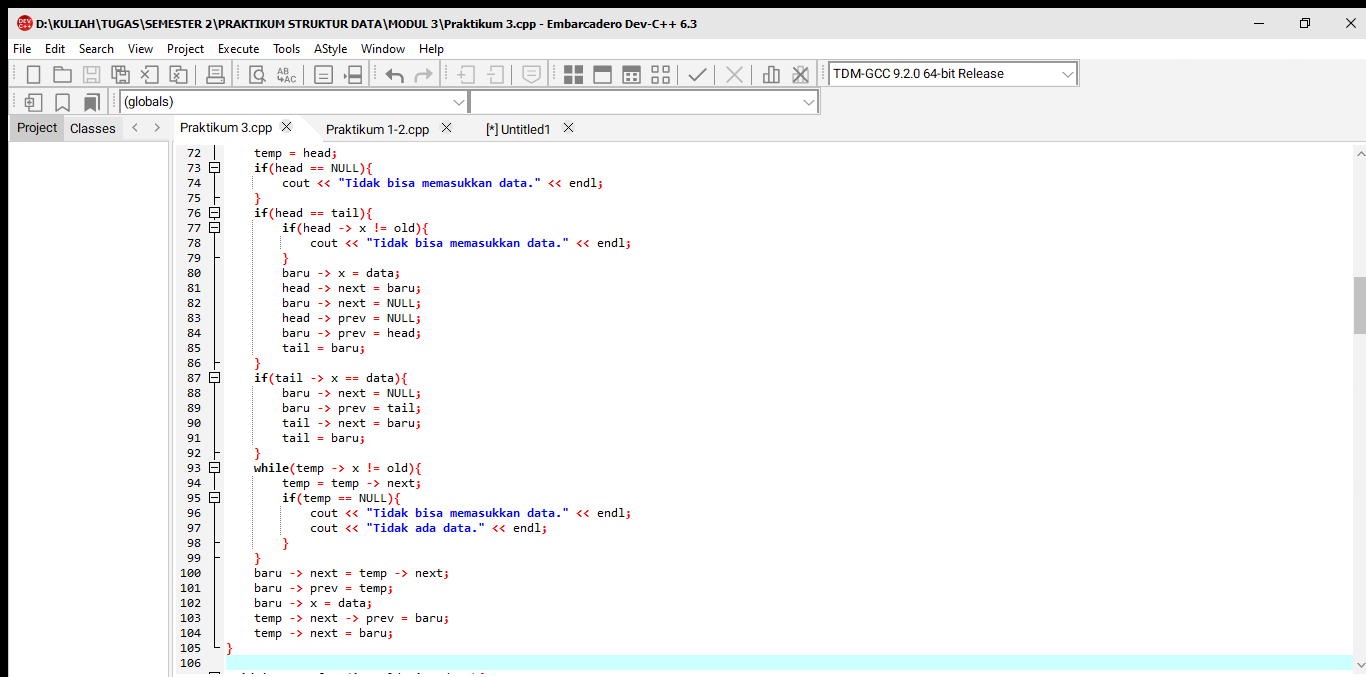
Ini merupakan program utama yang ditandai dengan “int main()”. Pada program utama ini diberi perintah untuk mencetak sederet keluaran tampilan menu utama program. *User* akan diminta untuk memasukkan input pilihan menu yang ingin diakses. Percabangan yang digunakan dalam proses ini adalah ‘switch (case)’. Pada program utama pun dideklarasikan perulangan. Selama *user* tidak memilih ‘8’ saat memilih menu, maka program akan terus diulang. Dalam setiap *case* pilihan oleh *user*, program hanya perlu memanggil prosedur-prosedur yang sudah dideklarasikan sebelumnya.



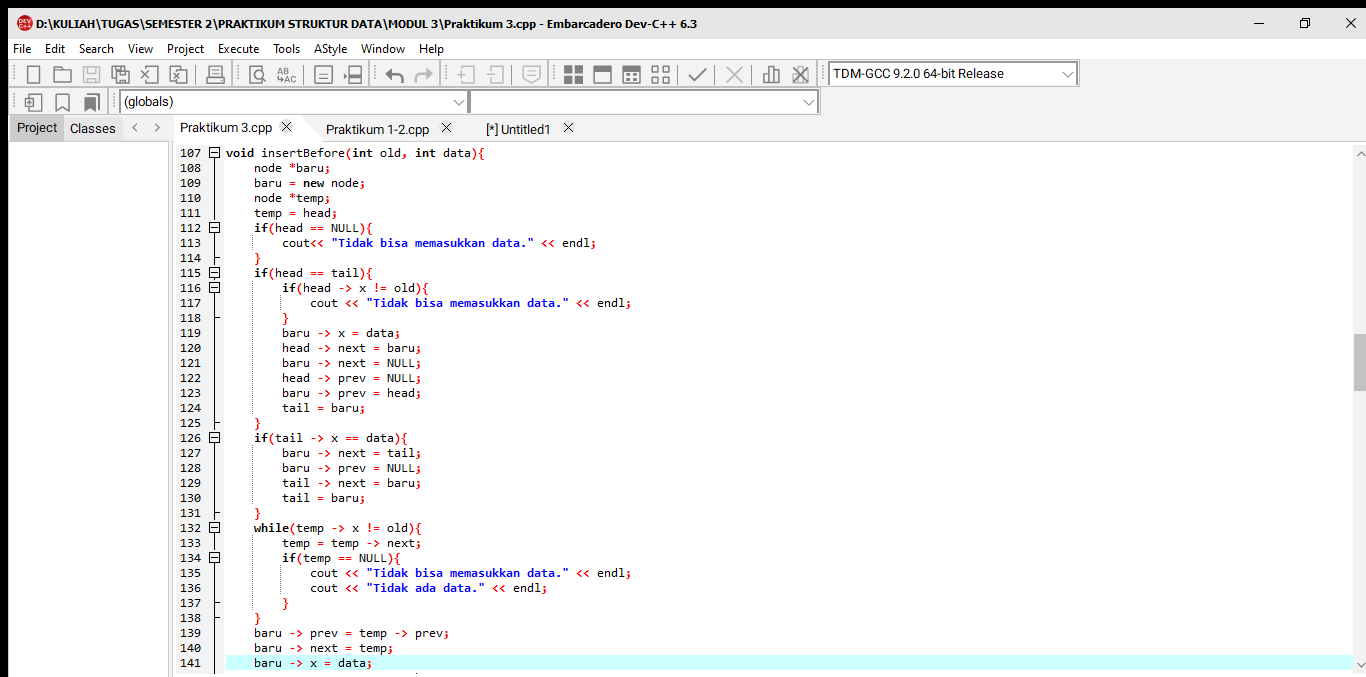
Gambar 3.1 *Source Code* Program (1)



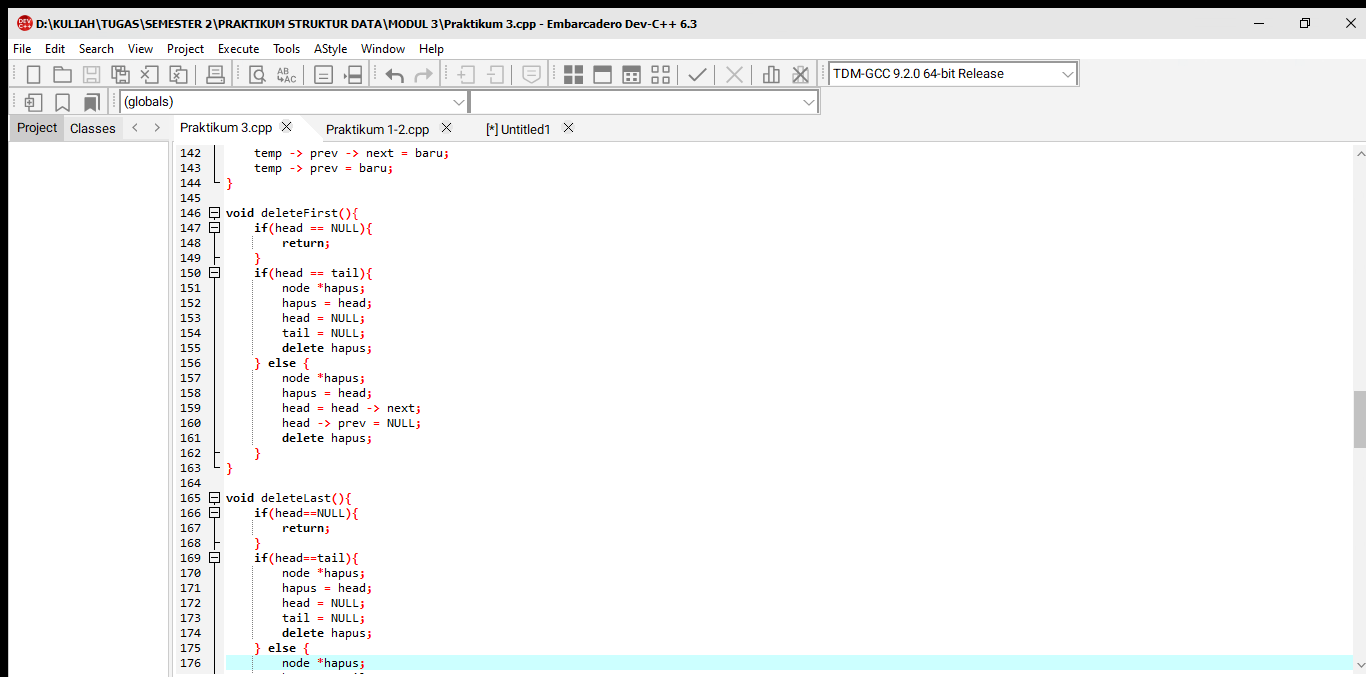
Gambar 3.2 *Source Code* Program (2)



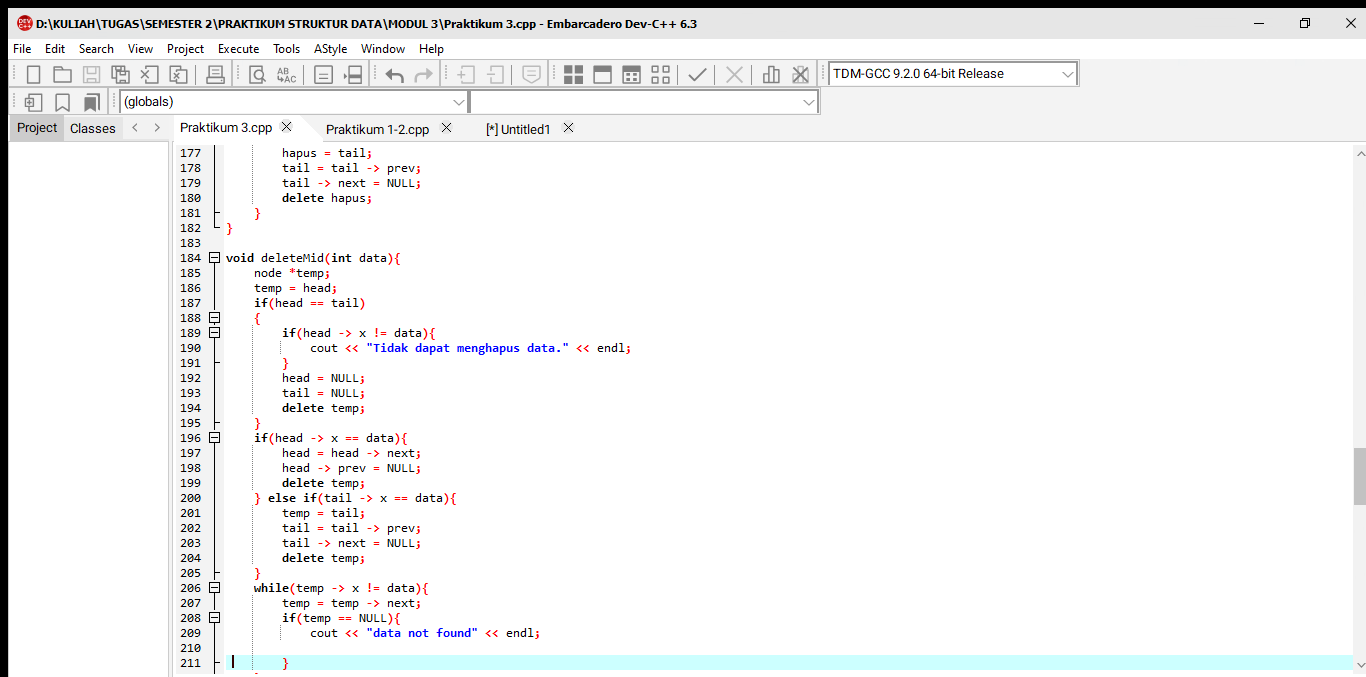
Gambar 3.3 *Source Code* Program (3)



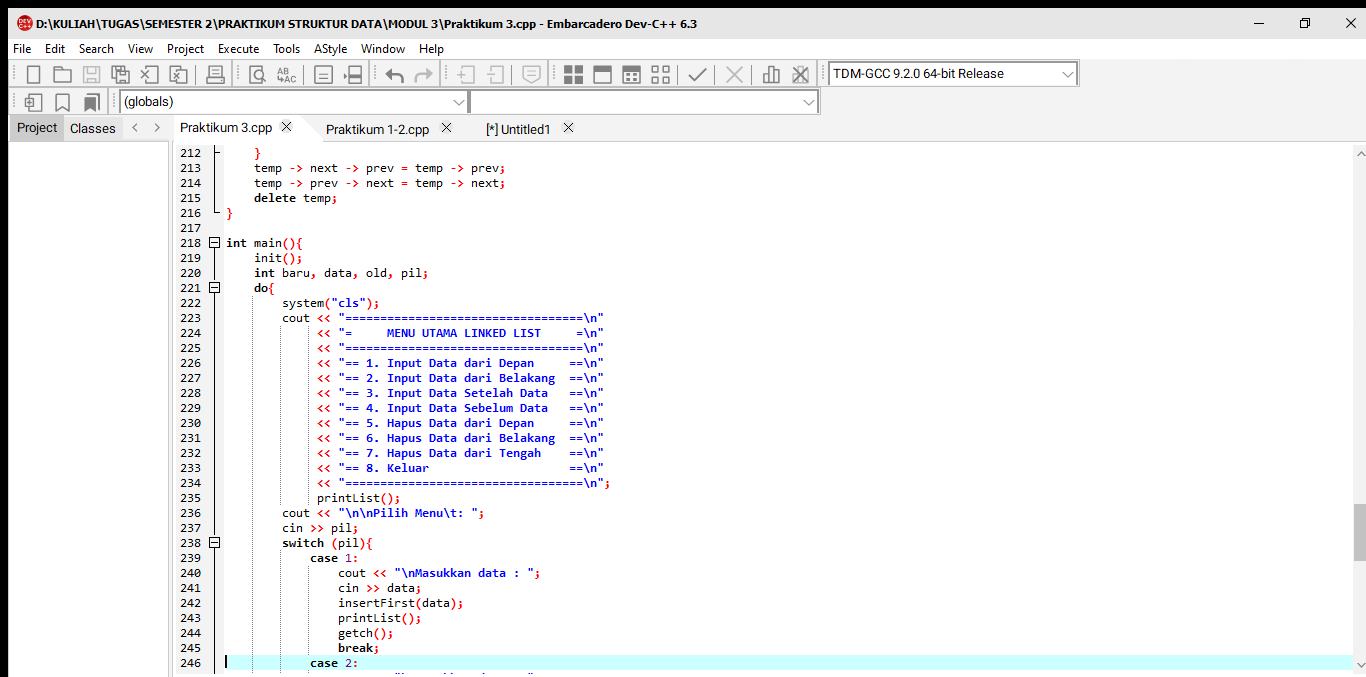
Gambar 3.4 *Source Code* Program (4)



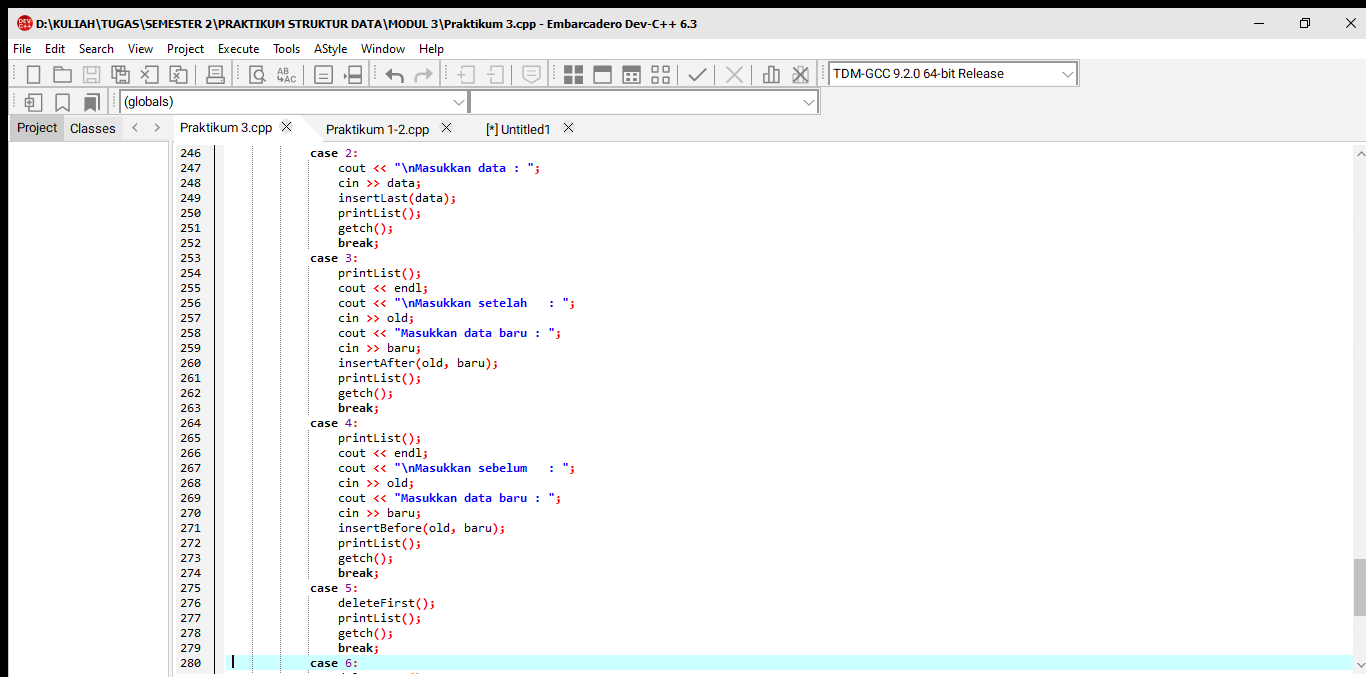
Gambar 3.5 *Source Code* Program (5)



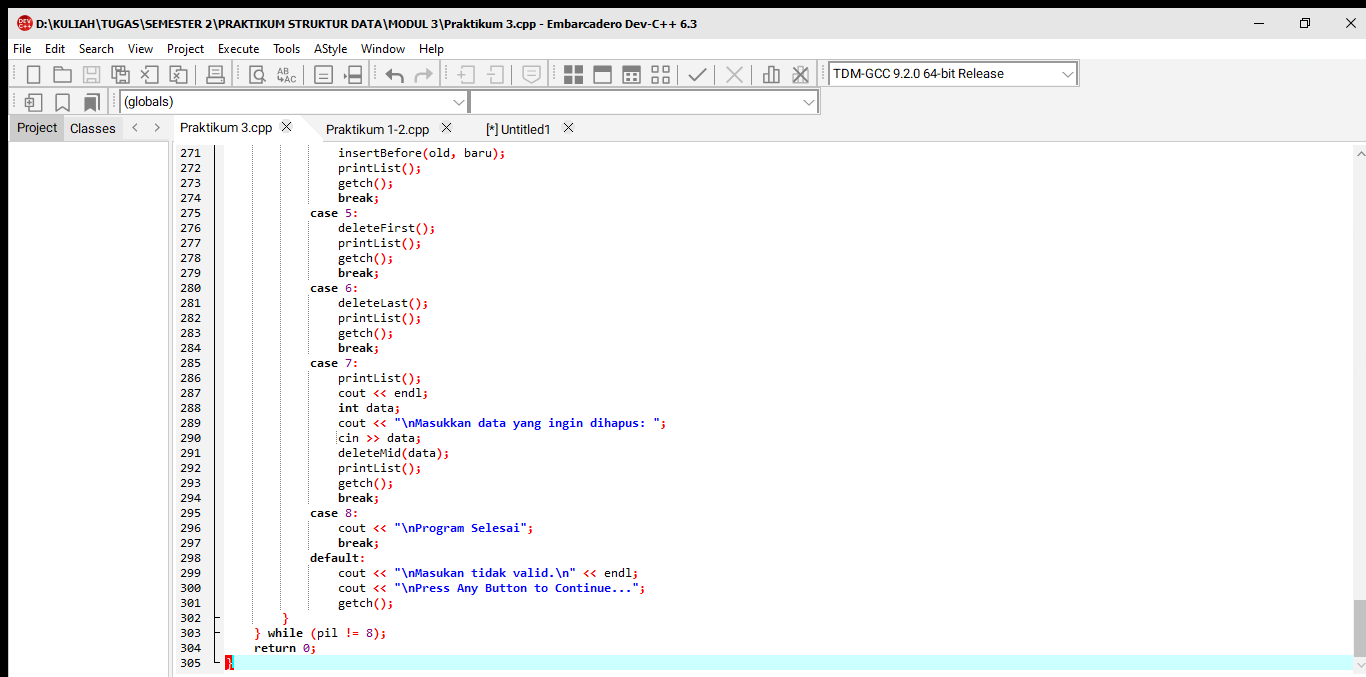
Gambar 3.6 *Source Code* Program (6)



Gambar 3.7 *Source Code* Program (7)



Gambar 3.8 *Source Code* Program (8)



Gambar 3.9 *Source Code* Program (9)



Gambar 4.1 *Output* Menu 1



Gambar 4.2 *Output* Menu 2



Gambar 4.3 *Output* Menu 3



Gambar 4.4 *Output* Menu 4



Gambar 4.5 *Output* Menu 5



Gambar 4.6 *Output* Menu 6



Gambar 4.7 *Output* Menu 1 (2)



Gambar 4.8 *Output* Menu 7



Gambar 4.9 *Output* Menu 8

**BAB III**

**KESIMPULAN**

1. **KESIMPULAN**

*Linked list* adalah suatu cara untuk menyimpan data dengan suatu struktur sehingga *programmer* dapat secara otomatis menciptakan suatu tempat baru untuk menyimpan data kapan saja diperlukan. *Linked list* dikenal juga dengan sebutan senarai berantai adalah stuktur data yang terdiri dari urutan *record* data dimana setiap *record* memiliki *field* yang menyimpan alamat/referensi dari *record* selanjutnya (di dalam urutan). Elemen data yang dihubungkan dengan *link* pada *linked list* disebut Node. Biasanya dalam suatu linked list, terdapat istilah *head* dan *tail*.

Ada 5 proses dasar di dalam *linked list*, yaitu:

1. Proses inisialisasi
2. Proses simpul baru
3. Membuat simpul awal
4. Menambah simpul baru ke dalam *linked list*
5. Menghapus sebuah simpul dalam *linked list*

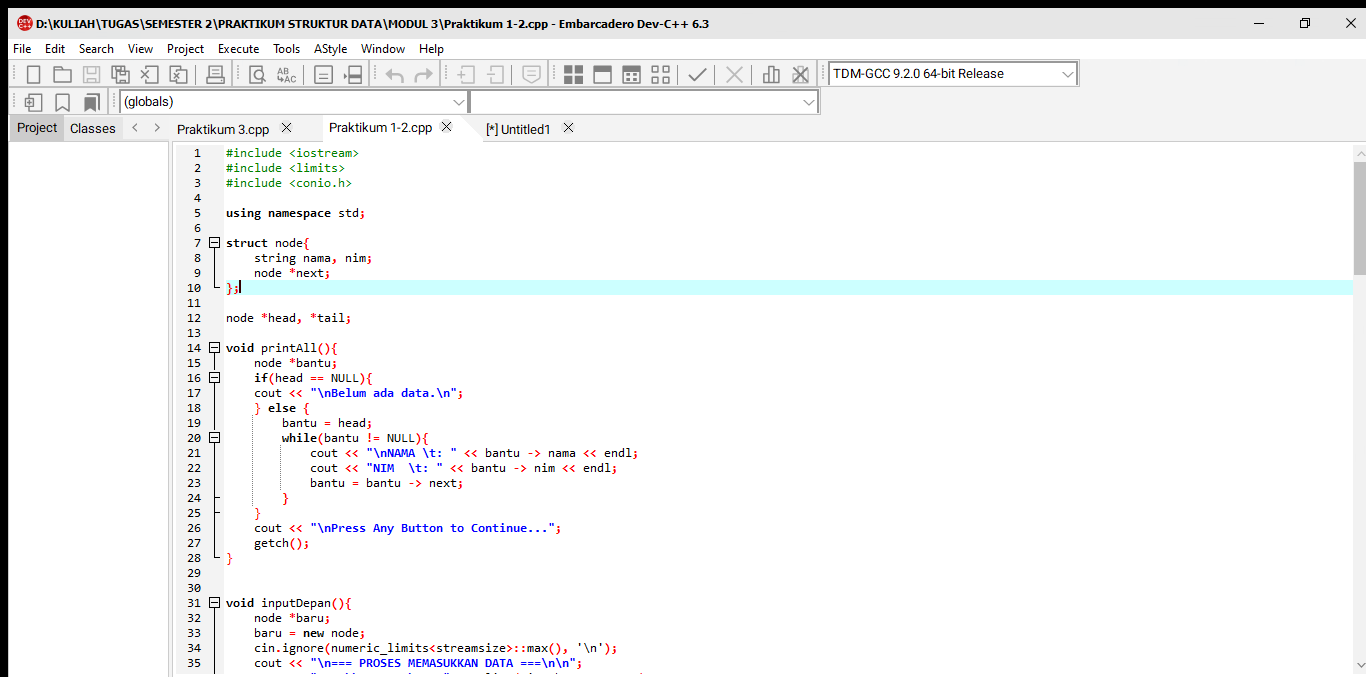
Ada beberapa jenis *linked list*, yaitu sebagai berikut.

1. *Single linked list*
2. *Double linked list*
3. *Circular linked list*
4. *Multiple linked list*

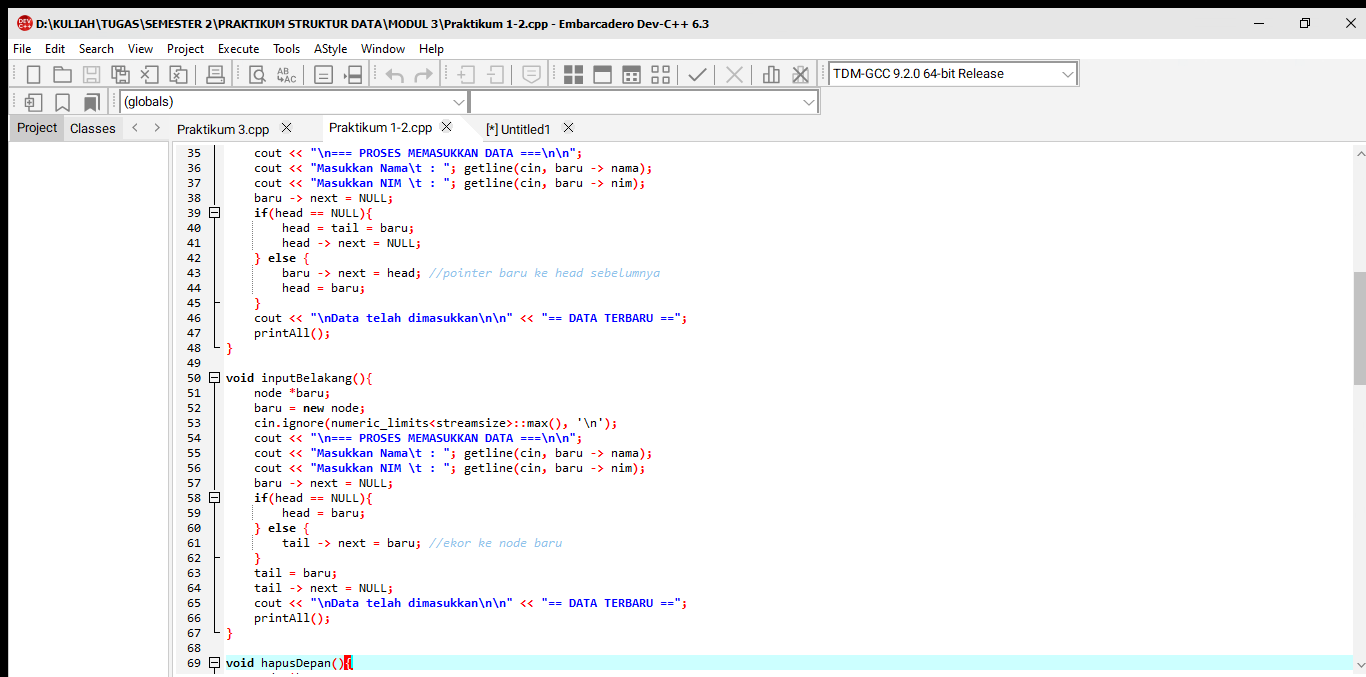
**DAFTAR PUSTAKA**

Dosen Teknik Informatika. 2020. *Modul Praktikum Struktur Data*. Palangka Raya. Jurusan Teknik Informatika Fakultas Teknik Universitas Palangka Raya.

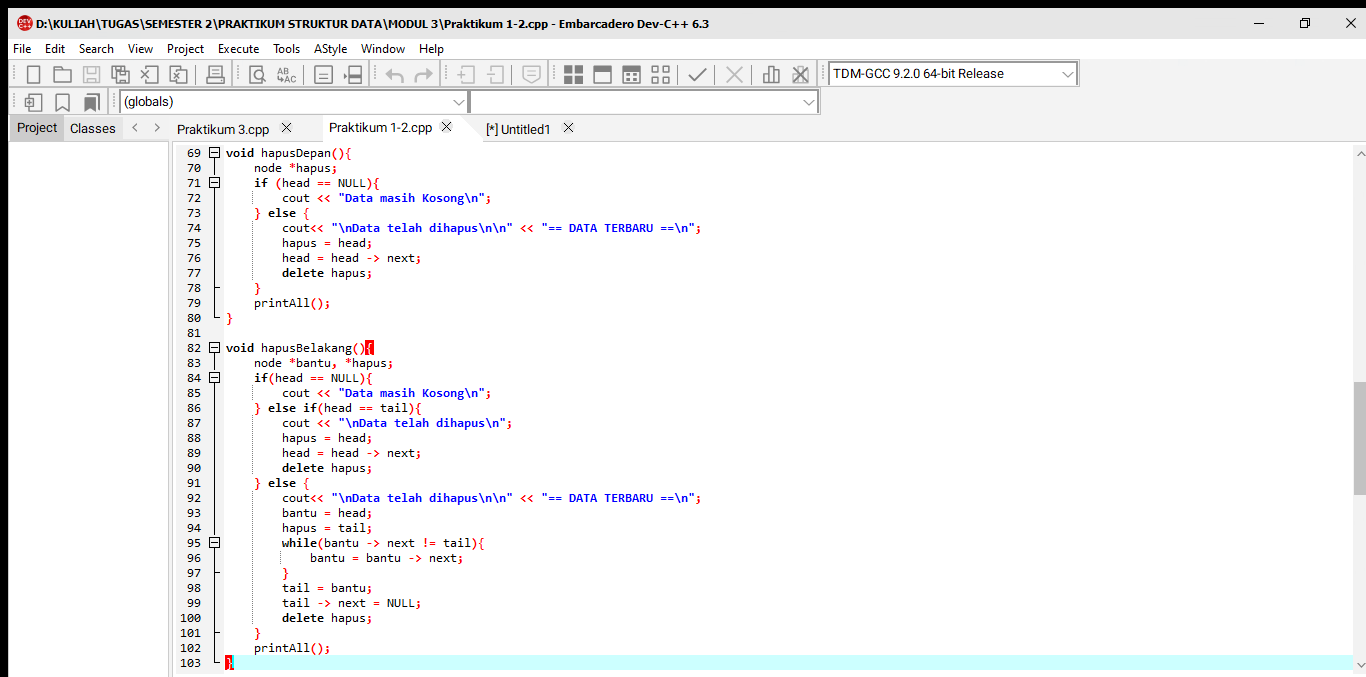
**LAMPIRAN**



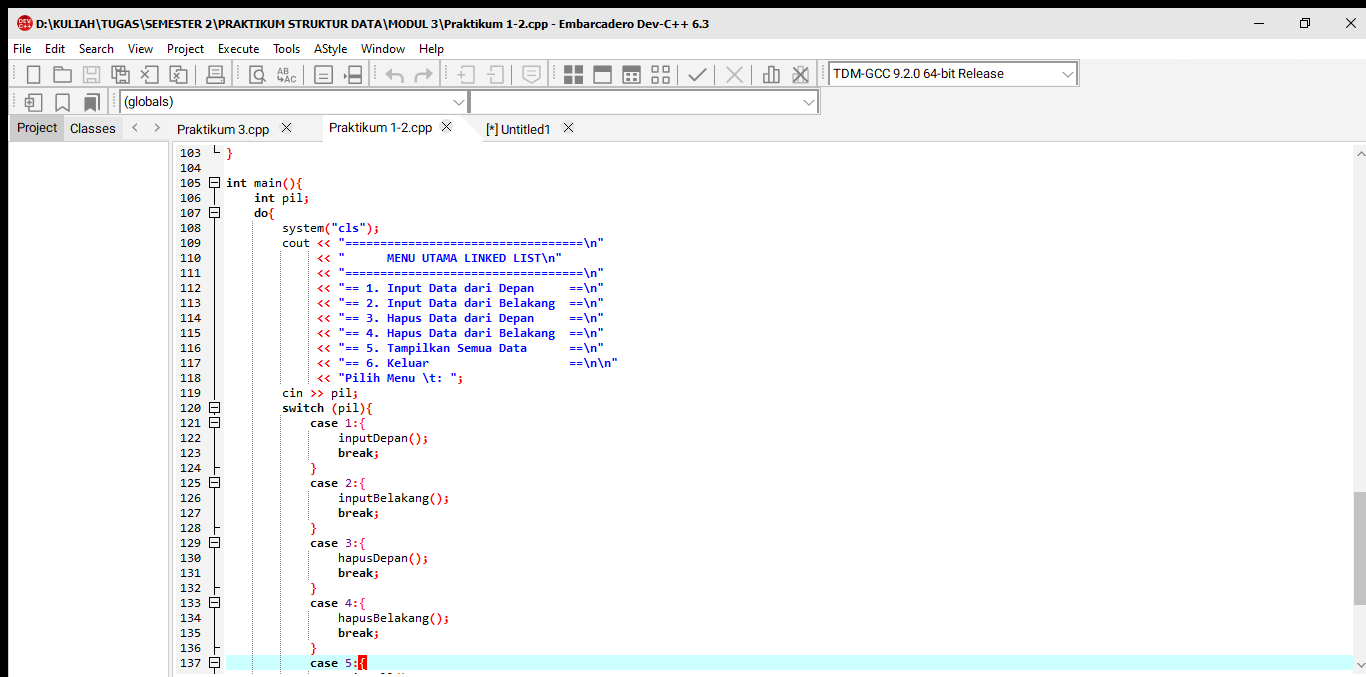
Gambar 1.1 *Source Code* Program (1)



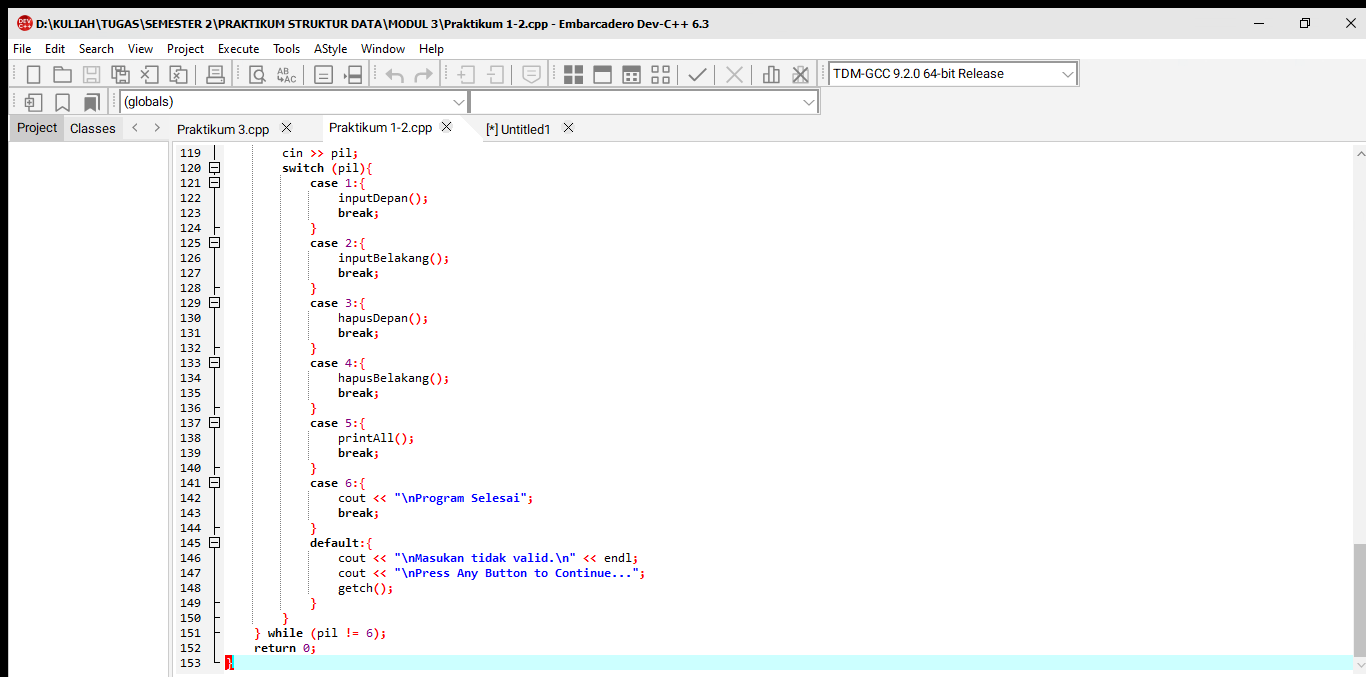
Gambar 1.2 *Source Code* Program (2)



Gambar 1.3 *Source Code* Program (3)



Gambar 1.4 *Source Code* Program (4)



Gambar 1.5 *Source Code* Program (5)



Gambar 2.1 *Output* Menu 1 (1)



Gambar 2.2 *Output* Menu 1 (2)



Gambar 2.3 *Output* Menu 2 (1)



Gambar 2.4 *Output* Menu 2 (2)



Gambar 2.5 *Output* Menu 3



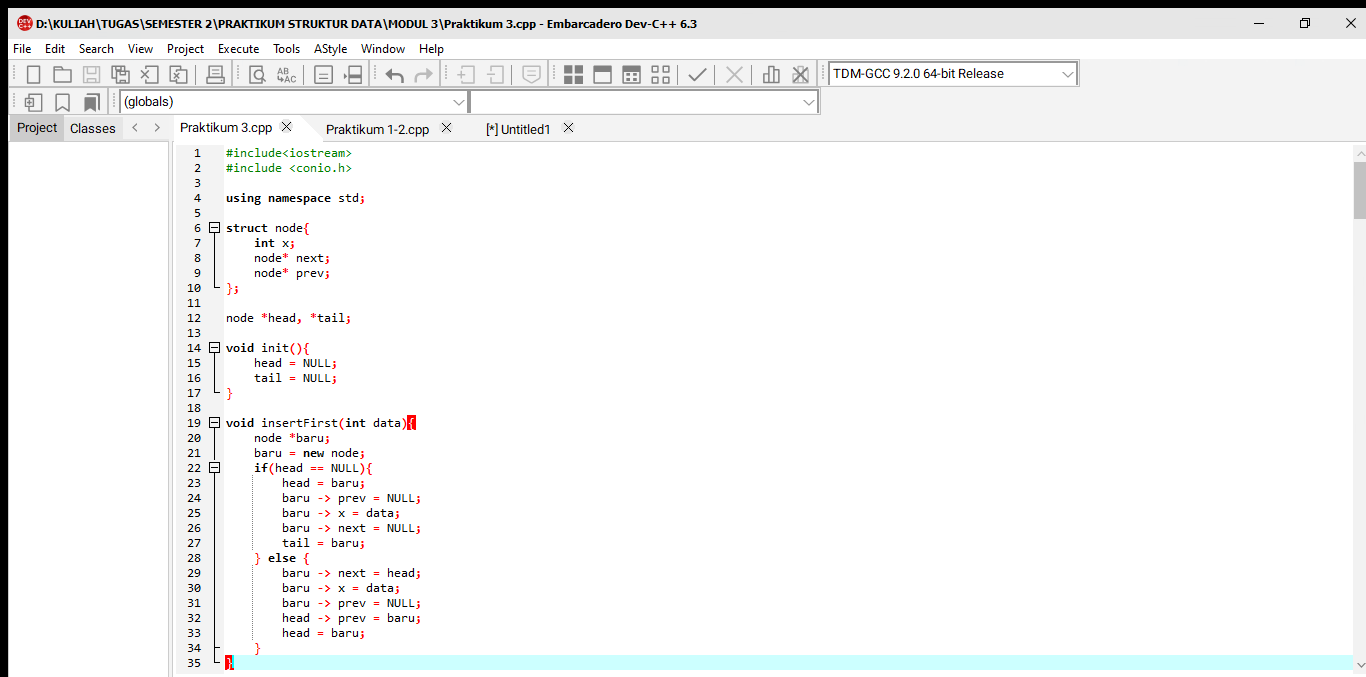
Gambar 2.6 *Output* Menu 4



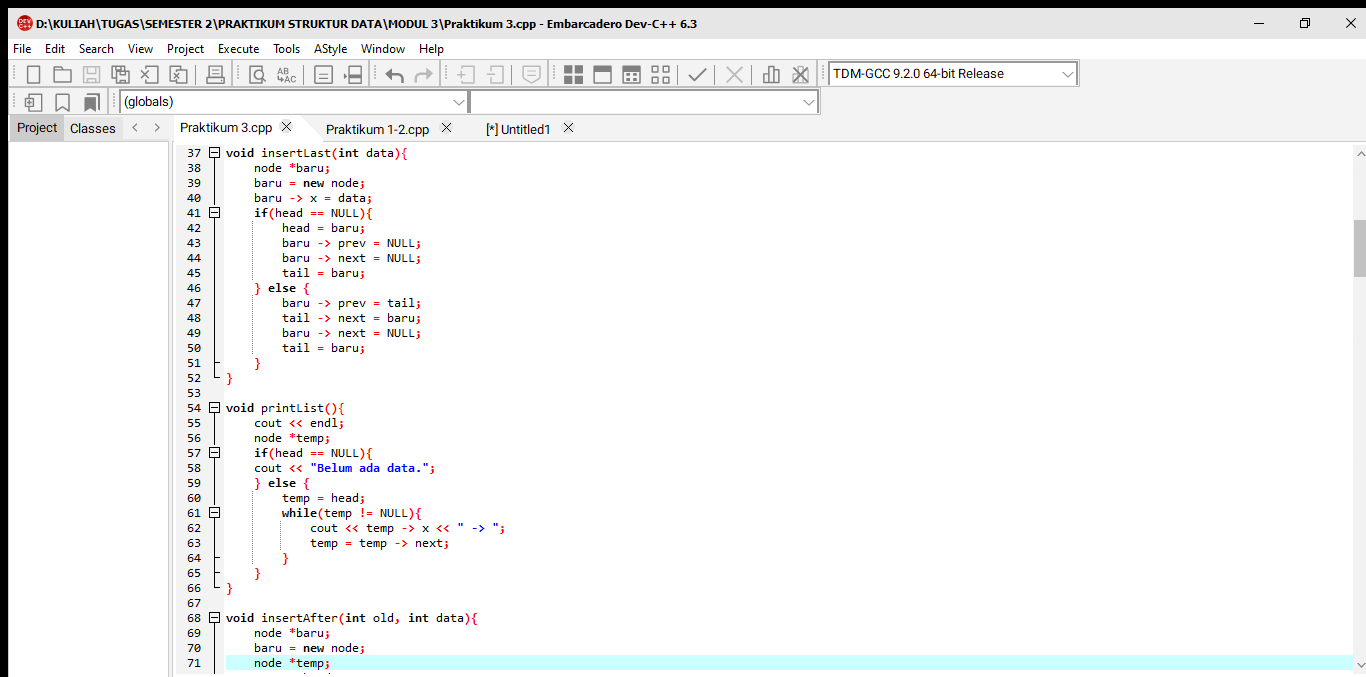
Gambar 2.7 *Output* Menu 5



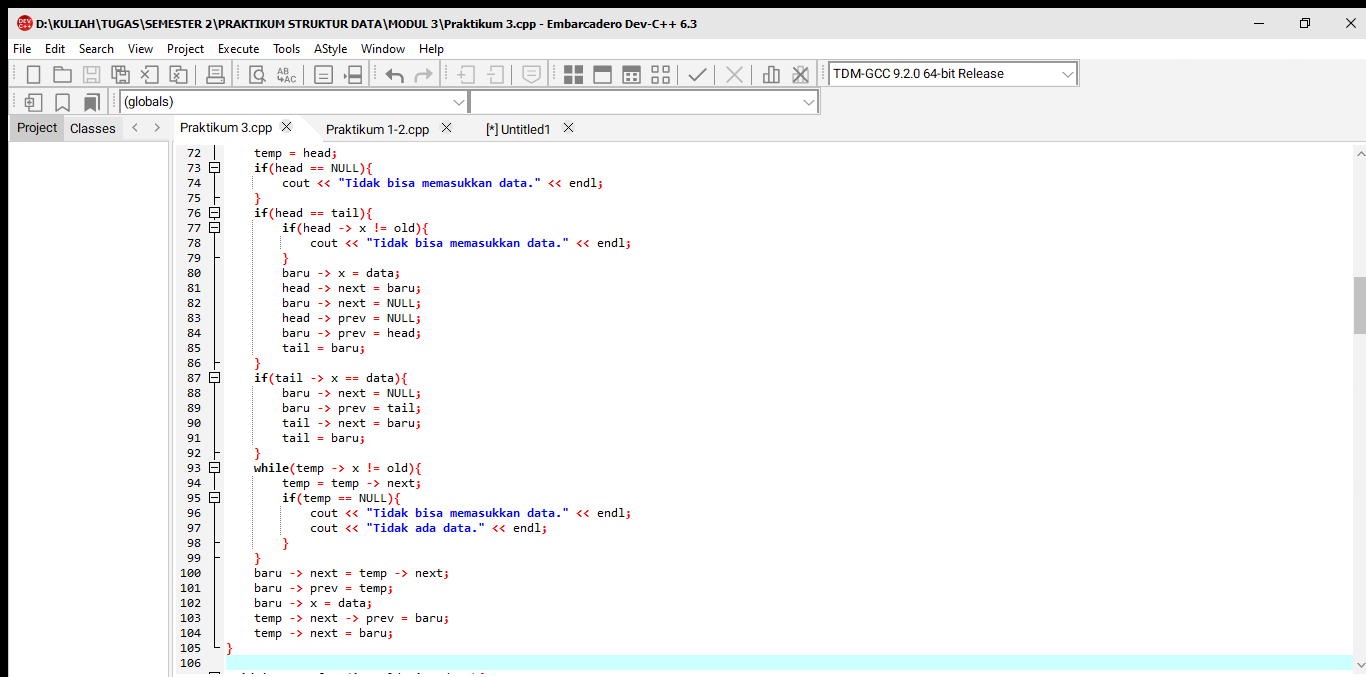
Gambar 2.8 *Output* Menu 6



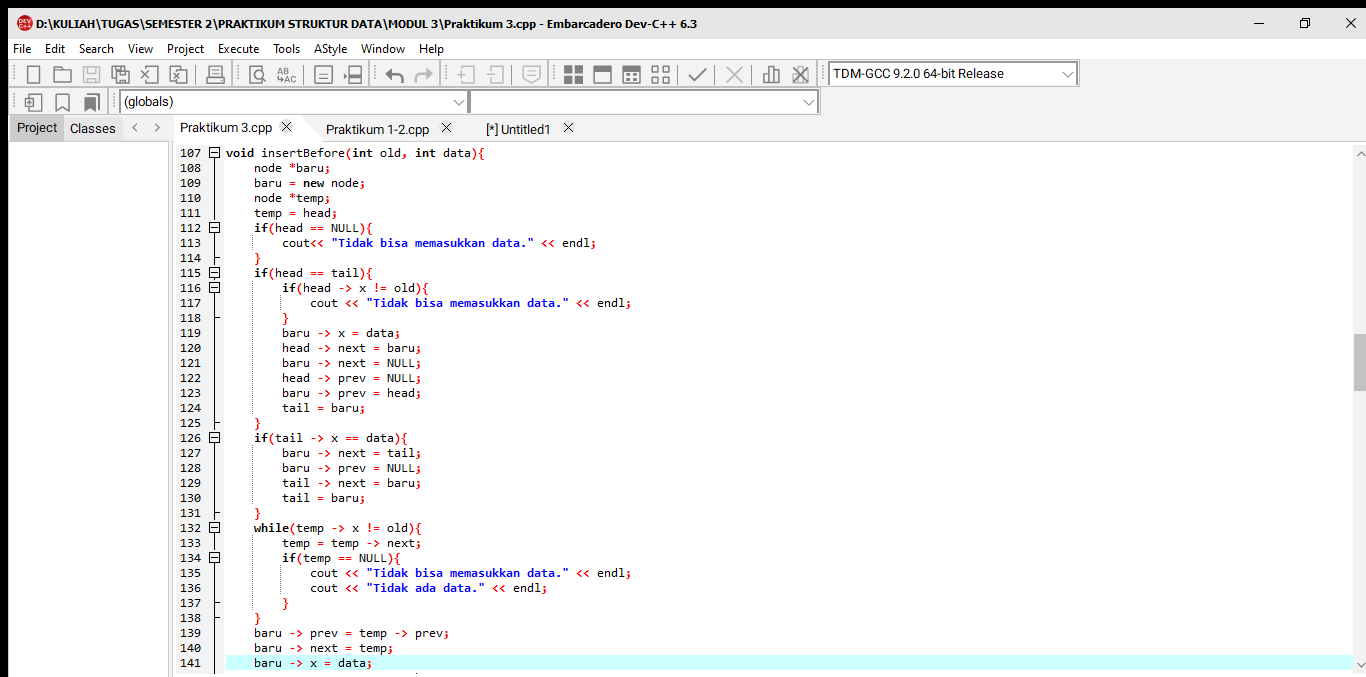
Gambar 3.1 *Source Code* Program (1)



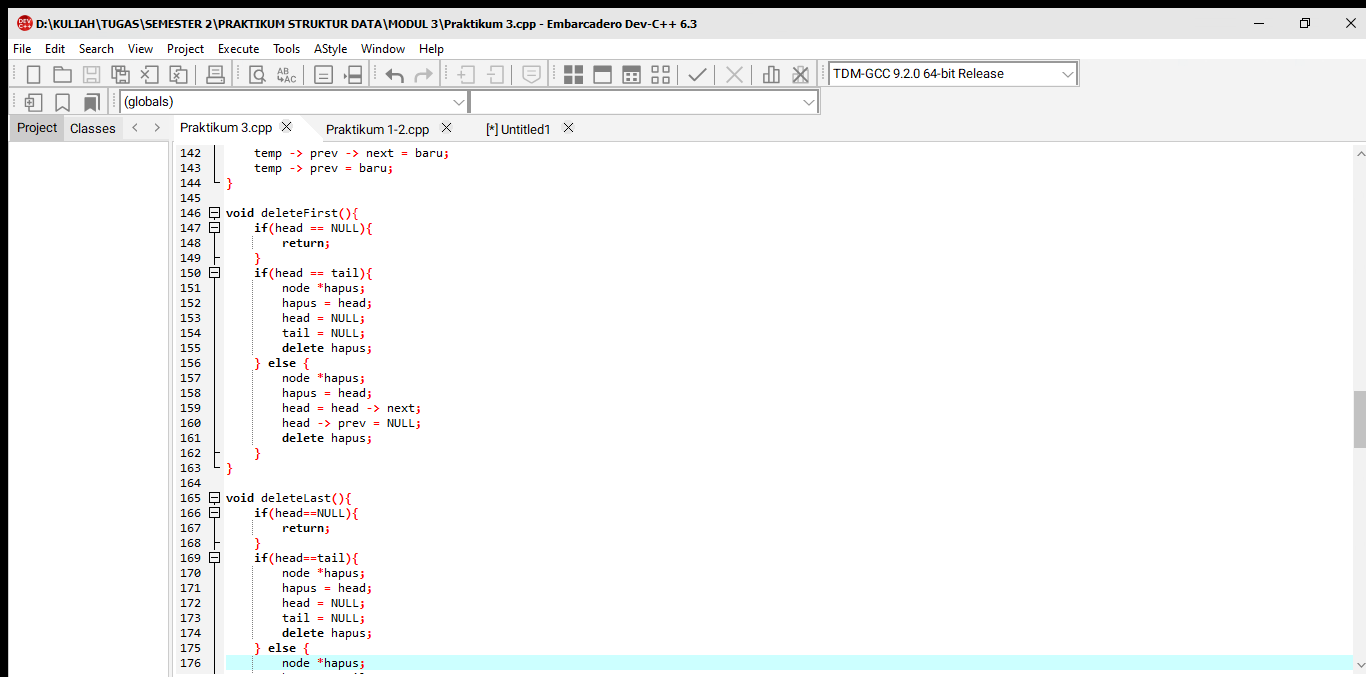
Gambar 3.2 *Source Code* Program (2)



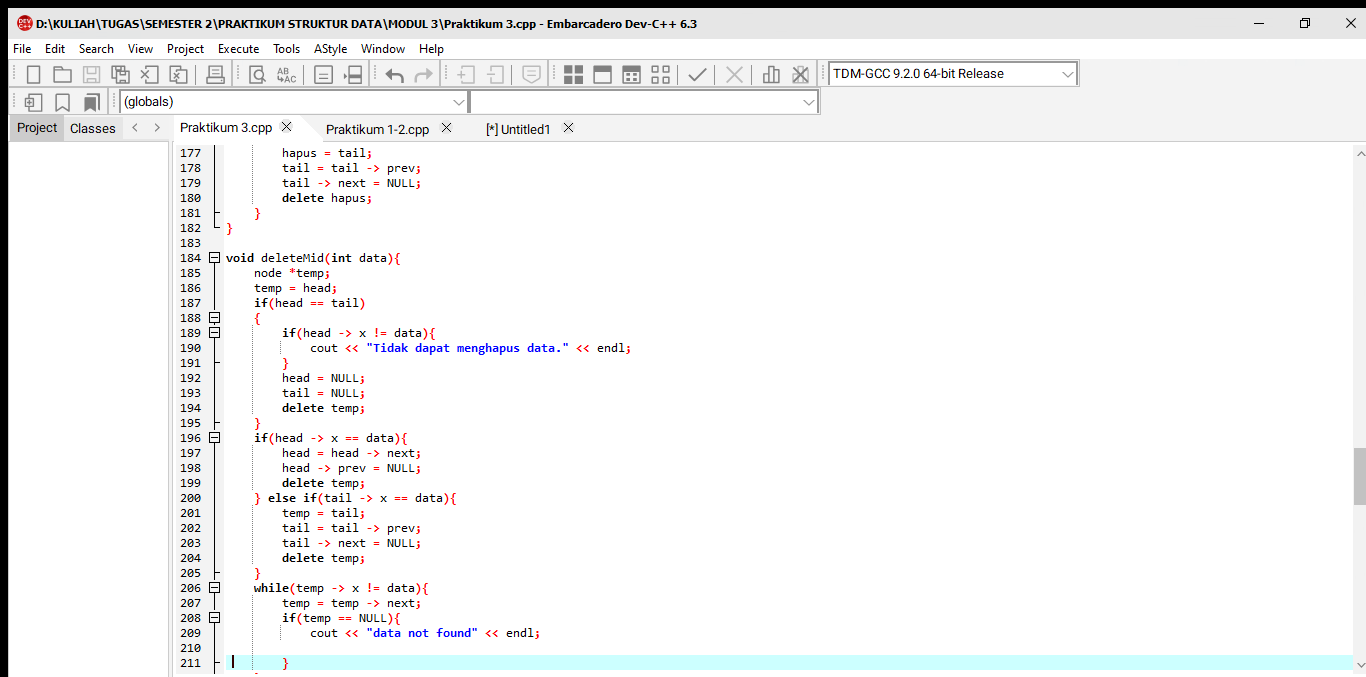
Gambar 3.3 *Source Code* Program (3)



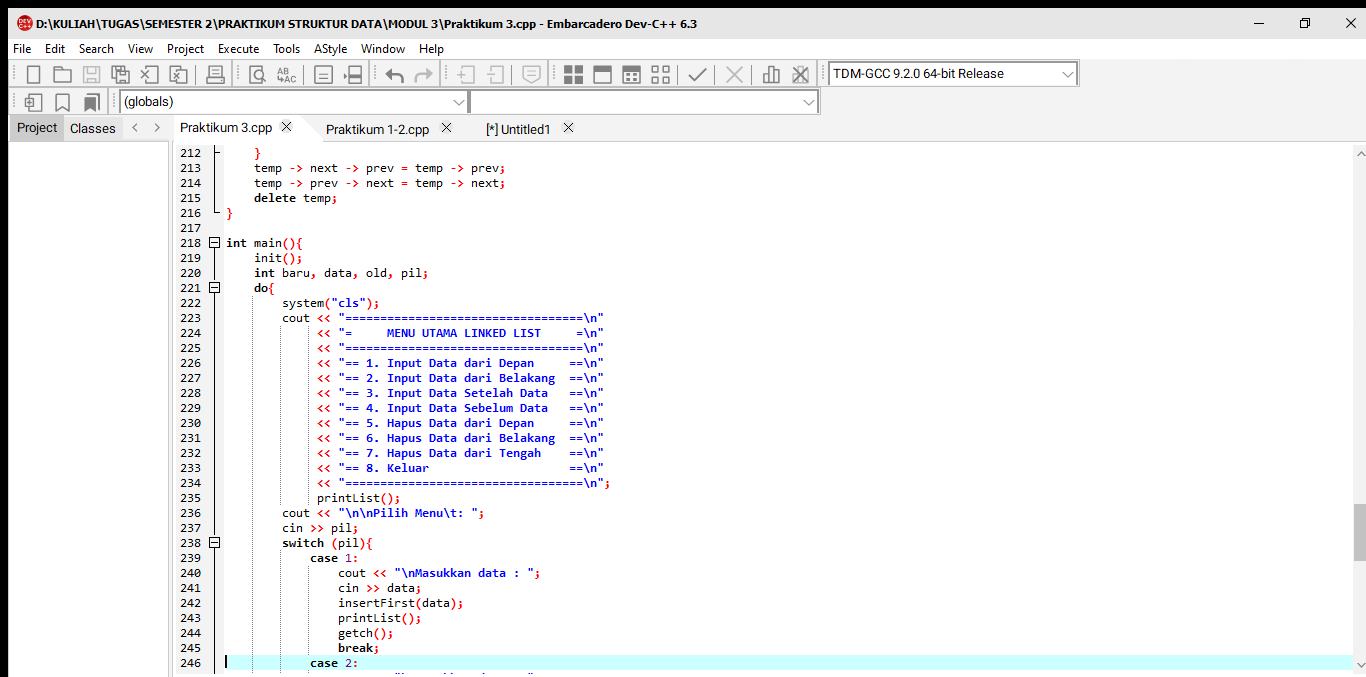
Gambar 3.4 *Source Code* Program (4)



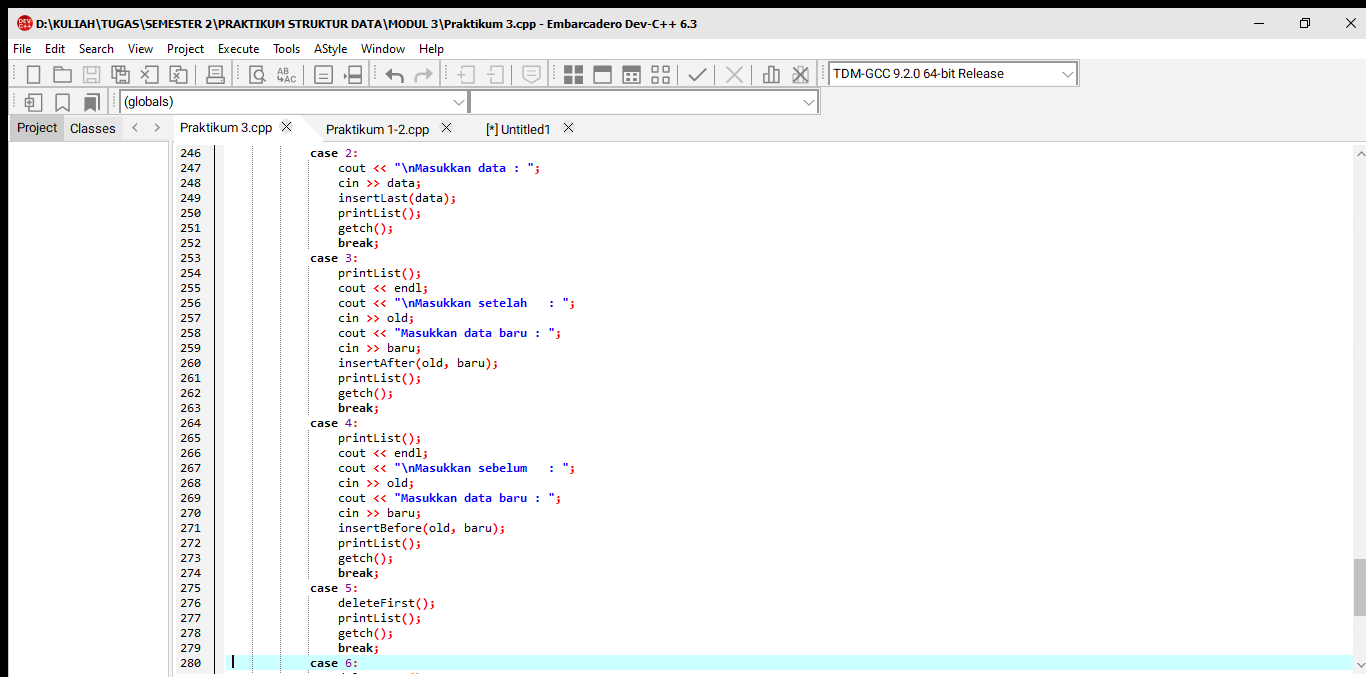
Gambar 3.5 *Source Code* Program (5)



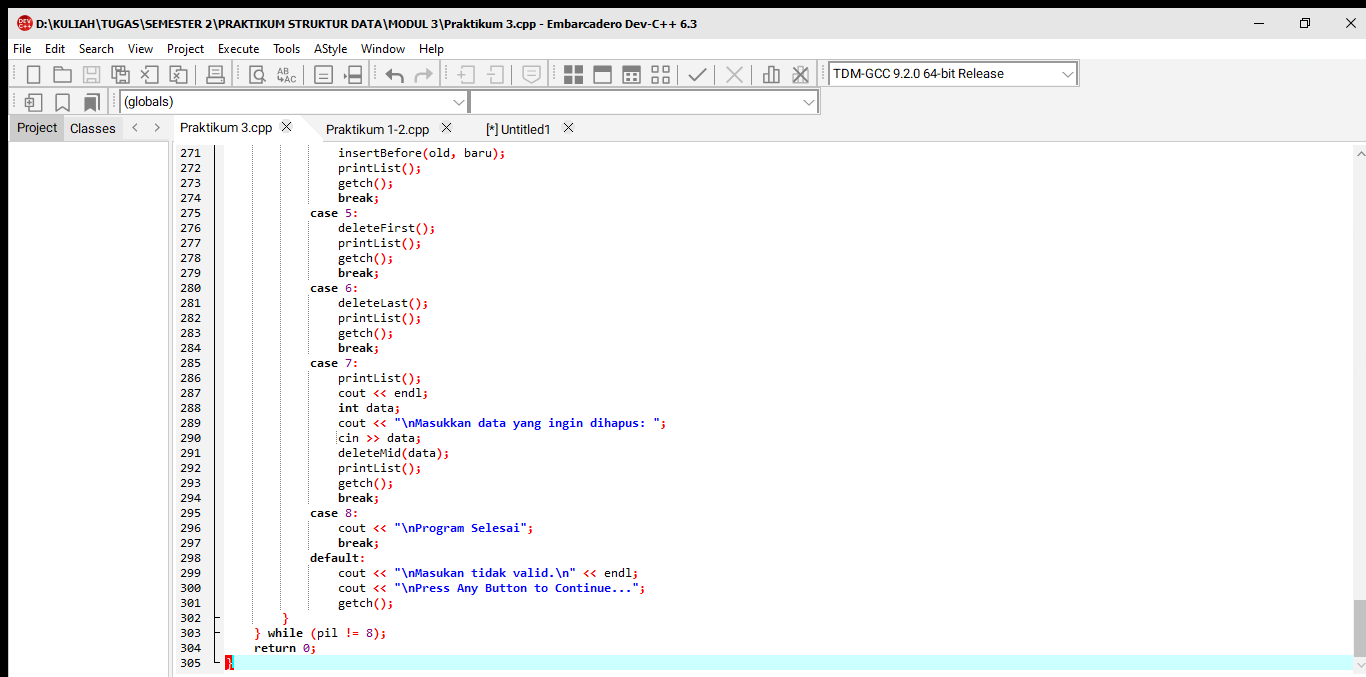
Gambar 3.6 *Source Code* Program (6)



Gambar 3.7 *Source Code* Program (7)



Gambar 3.8 *Source Code* Program (8)



Gambar 3.9 *Source Code* Program (9)



Gambar 4.1 *Output* Menu 1



Gambar 4.2 *Output* Menu 2



Gambar 4.3 *Output* Menu 3



Gambar 4.4 *Output* Menu 4



Gambar 4.5 *Output* Menu 5



Gambar 4.6 *Output* Menu 6



Gambar 4.7 *Output* Menu 1 (2)



Gambar 4.8 *Output* Menu 7



Gambar 4.9 *Output* Menu 8