LAPORAN TEORI
IMPLEMENTASI PROYEK AKHIR
DALAM TYPST

SOFTWARE ARCHITECTURE
AND QUALITY LABORATORY

NAMA	:	ZAIN AKBAR RIZKIA
NIM	:	202531091
KELAS	:	C
DOSEN	:	Nama Pembimbing
NO. PC	:	31
ASISTEN	:	Nama Reviewer 1
	:	Nama Reviewer 2
	:	Nama Reviewer 3
	:	Nama Reviewer 4

FAKULTAS TELEMATIKA ENERGI

TEKNIK INFORMATIKA

INSTITUT TEKNOLOGI PLN – JAKARTA

2025

DAFTAR ISI

DAFTAR GAMBAR

BAB I

PENDAHULUAN

1.1 Latar Belakang

Sistem manajemen akun bank merupakan aplikasi fundamental dalam dunia perbankan digital. Aplikasi ini dirancang untuk mengelola informasi akun pelanggan, melakukan transaksi setoran dan penarikan dana, serta menampilkan data akun yang tersimpan dalam sistem.

Pengembangan sistem manajemen akun bank menggunakan bahasa pemrograman C++ dengan paradigma pemrograman procedural memberikan fondasi yang kuat dalam memahami konsep pemrograman tingkat pemula. Melalui proyek ini, pembaca akan mempelajari penggunaan struktur data (struct), array, fungsi, sistem input-output berbasis stream, dan kontrol alur program menggunakan switch-case.

Dengan membatasi penggunaan library hanya pada `iostream` dan `cstdlib`, proyek ini menekankan pemahaman mendalam terhadap mekanisme dasar C++ tanpa bergantung pada fitur-fitur advanced atau template library. Hal ini memastikan bahwa setiap aspek program dapat dipahami secara menyeluruh dari perspektif algoritma dan logika pemrograman.

1.2 Tujuan

1. Memahami dan menerapkan struktur data `struct` dalam C++ untuk menyimpan informasi akun bank yang terdiri dari nomor akun, nama pemegang, alamat, jenis akun, dan saldo.
2. Menerapkan array sebagai kontainer untuk menyimpan hingga lima akun bank secara simultan.
3. Mengimplementasikan operasi dasar perbankan meliputi pembuatan akun baru, setoran, penarikan, pencarian, modifikasi, dan penghapusan akun.
4. Menerapkan fungsi dengan parameter referensi untuk memanipulasi data akun secara efisien.
5. Mengembangkan menu interaktif berbasis command-line interface (CLI) dengan sistem pilihan menggunakan `switch-case`.

1.3 Manfaat

1. Pemahaman konseptual tentang abstraksi data melalui struktur dan cara data diorganisir dalam memori.
2. Praktik penulisan fungsi yang reusable dan terstruktur untuk operasi spesifik.
3. Pengalaman dalam menangani input user dengan validasi dan penanganan kondisi error.
4. Kemampuan untuk merancang algoritma pencarian linear untuk menemukan akun dalam array.

5. Fondasi dasar untuk pengembangan aplikasi berbasis data yang lebih kompleks.

1.4 Batasan Masalah

1. Kapasitas maksimal sistem hanya dapat menampung lima akun bank, didefinisikan oleh konstanta `MAX_ACCOUNTS`.
2. Data akun hanya tersimpan dalam memori RAM selama program berjalan; tidak ada mekanisme penyimpanan data permanen ke file.
3. Sistem menggunakan array statis, bukan struktur data dinamis seperti linked list atau vector.
4. Nomor akun harus bernilai unik; sistem tidak menerapkan mekanisme otomatis untuk menghasilkan nomor akun.
5. Jenis akun dibatasi pada dua pilihan: ‘s’ untuk tabungan (savings) dan ‘c’ untuk giro (current).
6. Library yang digunakan terbatas pada `iostream` dan `cstdlib`; tidak menggunakan library tambahan seperti `string` dalam definisi struct.
7. Sistem tidak menerapkan enkripsi atau keamanan data; hanya validasi dasar pada nilai masukan.

BAB II

LANDASAN TEORI

2.1 Struktur Data

Struktur data dalam C++ didefinisikan menggunakan keyword `struct` untuk mengelompokkan beberapa variabel dari tipe data yang berbeda menjadi satu kesatuan logis. Menurut dokumentasi cppreference, struct dalam C++ merupakan user-defined type yang mengkombinasikan multiple members dengan tipe berbeda dalam suatu composite type. Ketika struct dideklarasikan dengan body, pernyataan tersebut sekaligus merupakan definisi yang mengalokasikan template tipe dalam sistem type C++.

Dalam proyek ini, struktur `BankAccount` digunakan untuk merepresentasikan data tunggal akun:

```
26
25  struct BankAccount {
24      int accno;      // Nomor akun, tipe integer
23      string name;    // Nama pemegang akun
22      string address; // Alamat pemegang akun
21      char actype;    // Jenis akun: 's' atau 'c'
20      float amount;   // Saldo akun dalam rupiah
19  };
18
```

Setiap anggota struct memiliki offset tetap dari awal struktur di memori. Dalam bahasa C++, akses terhadap member struktur dilakukan dengan operator dot (.) ketika menggunakan variabel struct langsung, dan operator arrow (->) ketika menggunakan pointer ke struct.

2.2 Array dan Indeksasi

Array merupakan koleksi elemen tipe data yang sama, tersimpan dalam lokasi memori yang berdampingan (contiguous). Dalam C++, deklarasi array menggunakan notasi `type name[size]`, di mana `size` merupakan konstanta yang menentukan jumlah maksimal elemen.

Menurut cppreference, subscript operator `[]` digunakan untuk mengakses elemen array pada indeks tertentu. Operator ini didefinisikan secara identik dengan ekspresi pointer arithmetic: `E1[E2]` secara fundamental equivalent dengan `*((E1)+(E2))`. Indeksasi array dalam C++ dimulai dari 0, sehingga array dengan size `n` memiliki indeks valid dari 0 hingga `n-1`.

Dalam proyek, array `BankAccount accounts[MAX_ACCOUNTS]` menyimpan hingga lima akun:

```
5   const int MAX_ACCOUNTS = 5;
6
7   struct BankAccount {
8     int accno;
9     string name;
10    string address;
11    char actype;
12    float amount;
13  };
14
15  BankAccount accounts[MAX_ACCOUNTS];
16
17  int main() {
18    // Akses elemen pertama
19    accounts[0].accno = 1001;
20    accounts[0].name = "Jane Doe";
21
22    // Iterasi seluruh array
23    for (int i = 0; i < MAX_ACCOUNTS; ++i) {
24      cout << "Account " << i
25           << " No: " << accounts[i].accno << endl;
26    }
27
28    return 0;
29  }
```

Subscript operator pada array mengembalikan lvalue apabila array adalah lvalue, memungkinkan baik pembacaan maupun penulisan nilai pada posisi tersebut.

2.3 Fungsi dan Parameter Referensi

Fungsi dalam C++ merupakan blok kode yang dapat dipanggil berulang dengan identitas dan tipe parameter tertentu. Dokumentasi cppreference menyatakan bahwa function declaration memperkenalkan nama fungsi dan tipe fungsi, sedangkan function definition mengasosiasikan nama/ tipe dengan body fungsi.

Parameter referensi dalam C++ (reference parameter) memungkinkan fungsi memodifikasi argumen asli tanpa membuat salinan. Notasi parameter referensi menggunakan symbol ampersand (&):

```
12
13  void deposit(BankAccount &acc) {
14      float d;
15      cout << "\nMasukkan jumlah untuk disetor = ";
16      cin >> d;
17      if (d > 0) {
18          acc.amount += d; // Modifikasi saldo langsung
19          cout << "Setoran berhasil. Saldo Baru = "
20               << acc.amount << endl;
21      } else {
22          cout << "Jumlah setoran harus positif."
23               << endl;
24      }
25  }
26
```

Parameter referensi berbeda dengan parameter pointer, karena referensi tidak memerlukan dereference dan tidak dapat bernilai null. Fungsi yang menerima referensi dapat secara langsung mengakses dan memodifikasi object asli melalui reference tersebut tanpa overhead pembuatan salinan (copy).

2.4 Stream Input-Output

Stream dalam C++ merupakan abstraksi untuk komunikasi data antara program dan external device. Menurut dokumentasi cppreference, iostream header mendefinisikan object global `cin` dengan tipe `istream` untuk membaca input dari keyboard, dan `cout` dengan tipe `ostream` untuk menampilkan output ke layar.

Ekstraksi data dari stream menggunakan operator >>:

```
4   int main() {
5       int accno;
6       cout << "Masukkan nomor akun: ";
7       cin >> accno; // Baca integer dari input stream
8       cout << "Nomor akun yang dimasukkan: " << accno
9            << endl;
10      return 0;
11  }
```

Insersi data ke stream menggunakan operator <<:

```
3
4   int main() {
5       float amount = 500.0;
6       cout << "Saldo: " << amount << endl;
7       return 0;
8   }
```

Ketika menggunakan `cin >>` diikuti `getline()`, diperlukan `cin.ignore()` untuk membersihkan karakter newline yang tertinggal di input buffer:

```
4
5   int main() {
6       int age;
7       cout << "Enter your age: ";
8       cin >> age;
9       cin.ignore(10000, '\n'); // Bersihkan buffer
10      string name;
11      cout << "Enter your name: ";
12      getline(cin, name); // Baca string dengan spasi
13      cout << "Name: " << name << ", Age: " << age
14           << endl;
15      return 0;
16  }
```

2.5 Kontrol Alur Program

Kontrol alur program menggunakan `switch-case` memungkinkan eksekusi kode berbeda berdasarkan nilai satu expression. Struktur switch-case dalam proyek ini merespons pilihan menu user:

```
int main() {
    int choice;
    cout << "Enter a choice (1 or 2): ";
    cin >> choice;

    switch (choice) {
    case 1:
        cout << "You chose 1." << endl;
        break;
    case 2:
        cout << "You chose 2." << endl;
        break;
    default:
        cout << "Pilihan tidak valid." << endl;
    }
    return 0;
}
```

Statement `break` penting untuk menghentikan eksekusi dan keluar dari switch-case. Tanpa `break`, eksekusi akan berlanjut ke case berikutnya (fall-through behavior).

2.6 Algoritma Pencarian Linear

Pencarian linear merupakan algoritma dasar yang memeriksa setiap elemen dalam array secara berurutan sampai menemukan elemen target atau mencapai akhir array. Dalam proyek, fungsi `find_account_idx()` mengimplementasikan algoritma ini untuk mencari akun berdasarkan nomor akun:

```
int find_account_idx(int acc_num,
                     const BankAccount accounts[],
                     int size) {
    for (int i = 0; i < size; ++i) {
        if (accounts[i].accno == acc_num) {
            return i; // Ditemukan pada indeks i
        }
    }
    return -1; // Tidak ditemukan
}
```

Kompleksitas waktu pencarian linear adalah O(n), di mana n adalah jumlah elemen yang diperiksa. Algoritma ini cukup efisien untuk dataset berukuran kecil seperti lima akun, namun untuk dataset lebih besar diperlukan struktur data dan algoritma yang lebih sophisticated.

BAB III

PERANCANGAN

3.1 Analisis Masalah

3.1.1 Data yang Dibutuhkan

Sistem manajemen akun bank memerlukan data berikut untuk setiap akun:

1. Nomor akun (integer): identifier unik untuk setiap akun, dipilih user saat pembuatan akun.
2. Nama pemegang (string): nama lengkap pemegang akun, dapat mengandung spasi dan karakter alphabetic.
3. Alamat (string): lokasi tempat tinggal pemegang akun, dapat mengandung spasi, karakter numerik, dan punctuation.
4. Jenis akun (char): klasifikasi akun sebagai ‘s’ untuk tabungan atau ‘c’ untuk giro.
5. Saldo (float): jumlah dana yang tersimpan dalam akun, dinyatakan dalam rupiah dengan dua desimal.

Data-data tersebut dienkapsulasi dalam struktur `BankAccount` dan disimpan dalam array dengan kapasitas maksimal lima akun.

3.1.2 Output yang Diinginkan

Sistem dirancang untuk menghasilkan output berikut:

1. Menu interaktif yang menampilkan delapan pilihan operasi kepada user.
2. Konfirmasi keberhasilan operasi dengan pesan yang jelas dan informatif.
3. Pesan error ketika operasi gagal, misalnya saldo tidak mencukupi atau akun tidak ditemukan.
4. Display detail akun yang menampilkan semua data akun dalam format terstruktur.
5. Informasi saldo setelah setiap transaksi setoran atau penarikan.

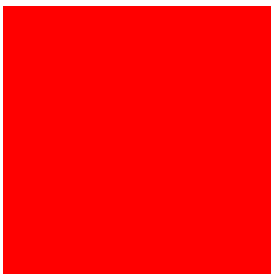
3.1.3 Proses Data

Proses dalam sistem terdiri dari:

1. Inisialisasi: Semua akun diinisialisasi dengan nilai default (nomor akun 0 menandakan slot kosong).
2. Pembuatan akun: User memasukkan data akun baru, sistem menyimpan pada slot pertama yang kosong.

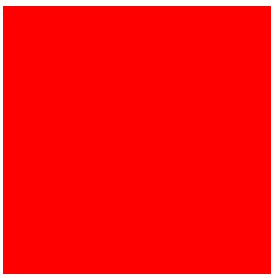
3. Pencarian akun: Sistem melakukan pencarian linear untuk menemukan akun berdasarkan nomor akun.
4. Transaksi setoran: Saldo ditambahkan dengan jumlah yang disetorkan, dengan validasi jumlah positif.
5. Transaksi penarikan: Saldo dikurangi dengan jumlah yang ditarik, dengan validasi saldo cukup dan jumlah positif.
6. Modifikasi akun: User dapat mengubah nama, alamat, dan jenis akun (tidak termasuk nomor akun dan saldo).
7. Penghapusan akun: Slot akun dikosongkan dengan reinisialisasi nilai-nilai-nya.
8. Tampilan semua akun: Sistem mengiterasi seluruh array dan menampilkan akun yang valid (nomor akun bukan 0).

3.2 Flowchart



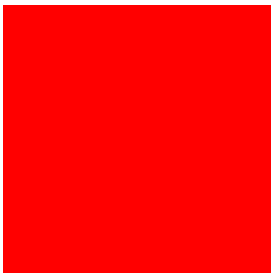
Gambar 3.9 Diagram Alur Utama

3.2.1 Flowchart Pembuatan Akun



Gambar 3.10 Diagram Alur Pembuatan Akun

3.2.2 Flowchart Setor Tunai



Gambar 3.11 Diagram Alur Setor Tunai

3.2.3 Flowchart Tarik Tunai

Gambar 3.12 Diagram Alur Tarik Tunai

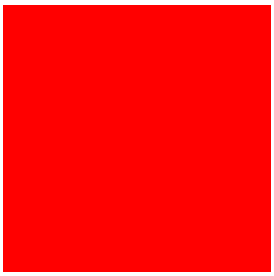
3.2.4 Flowchart Tampil Akun

Gambar 3.13 Diagram Alur Tampil Akun

3.2.5 Flowchart Ubah Akun

Gambar 3.14 Diagram Alur Ubah Akun

3.2.6 Flowchart Hapus Akun



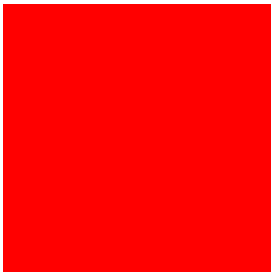
Gambar 3.15 Diagram Alur Hapus Akun

3.2.7 Flowchart Tampil Semua Akun

Gambar 3.16 Diagram Alur Tampil Semua Akun

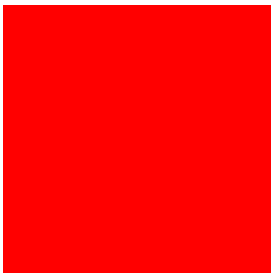
3.3 Algoritma

3.3.1 Pseudocode Utama Program

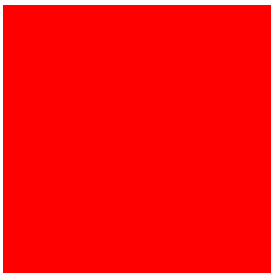


3.3.2 Pseudocode Fungsi-Fungsi Pendukung

3.3.2.1 Pseudocode initialize_account

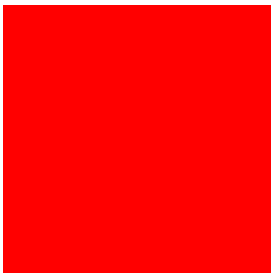


3.3.2.2 Pseudocode find_available_slot

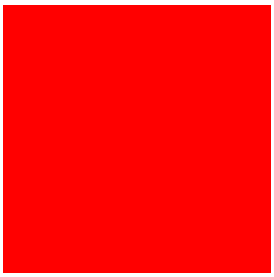


3.3.2.3 Pseudocode find_account_idx

3.3.2.4 Pseudocode deposit



3.3.2.5 Pseudocode withdraw



BAB IV

HASIL DAN PEMBAHASAN

4.1 Implementasi Struktur Data BankAccount

Struktur `BankAccount` merupakan inti dari sistem penyimpanan data. Definisi struktur ini mengelompokkan lima anggota yang mewakili atribut-atribut lengkap satu akun bank:

```
4
5   struct BankAccount {
6       int accno;   // Tipe int untuk nomor akun numerik
7       string name; // Tipe string untuk nama pemegang
8       string
9           address; // Tipe string untuk alamat pemegang
10      char actype; // Tipe char untuk jenis akun (s
11                   // atau c)
12      float amount; // Tipe float untuk saldo dalam
13                    // rupiah
14  };
15
```

Penggunaan `int` untuk nomor akun memastikan identifikasi numerik yang efisien. Tipe `float` untuk saldo memungkinkan representasi nilai dengan presisi desimal, meskipun untuk aplikasi finansial production-grade biasanya menggunakan tipe integer dengan unit cent untuk menghindari floating-point precision issues. Tipe `char` untuk jenis akun menghemat memori dibandingkan menggunakan string.

Ketika struct dideklarasikan, compiler mengalokasikan template tipe ini, namun tidak mengalokasikan memori untuk data aktual sampai variabel struct dideklarasikan. Deklarasi array `BankAccount accounts[MAX_ACCOUNTS]` mengalokasikan memori untuk lima instance struktur secara kontinu dalam memori.

4.2 Implementasi Array dan Sistem Indeksasi

Array statis digunakan untuk menyimpan akun-akun dengan kapasitas tetap:

```
4
5   const int MAX_ACCOUNTS = 5;
6
7   struct BankAccount {
8     int accno;
9     string name;
10    string address;
11    char actype;
12    float amount;
13  };
14
15  BankAccount accounts[MAX_ACCOUNTS];
16
```

Konstanta `MAX_ACCOUNTS` didefinisikan dengan nilai 5, membentuk batas atas jumlah akun yang dapat disimpan. Array ini dideklarasikan dengan scope global dalam fungsi `main()`, sehingga seluruh fungsi dapat mengaksesnya.

Akses elemen array menggunakan subscript operator `[]`:

```
10    string address;
11    char actype;
12    float amount;
13  };
14
15  BankAccount accounts[MAX_ACCOUNTS];
16
17  int main() {
18    accounts[0].accno = 1001; // Akses elemen pertama
19    accounts[4].amount =
20        500000.0; // Akses elemen terakhir
21
```

Subscript operator `[]` dalam C++ secara definitif equivalent dengan pointer arithmetic. Expresi `accounts[i]` sebenarnya diterjemahkan menjadi `*(accounts + i)`, di mana `accounts + i` menghitung address dengan offset sebesar `i * sizeof(BankAccount)` dari base address array.

Inisialisasi array dilakukan melalui looping:

```
24
25  int main() {
26      for (int i = 0; i < MAX_ACCOUNTS; ++i) {
27          initialize_account(accounts[i]);
28      }
29      cout << "All accounts initialized." << endl;
30      return 0;
31  }
```

Setiap iterasi memanggil fungsi `initialize_account()` dengan parameter referensi, mengakses setiap elemen array secara berurutan. Indeks dimulai dari 0 dan berakhir pada 4, mencakup seluruh kapasitas array.

4.3 Implementasi Fungsi dengan Parameter Referensi

Fungsi-fungsi dalam sistem menggunakan parameter referensi untuk memodifikasi data akun secara langsung tanpa membuat salinan:

```
12
13  void deposit(BankAccount &acc) {
14      float d;
15      cout << "\nMasukkan jumlah untuk disetor = ";
16      cin >> d;
17      if (d > 0) {
18          acc.amount += d;
19          cout << "Setoran berhasil. Saldo Baru = "
20               << acc.amount << endl;
21      } else {
22          cout << "Jumlah setoran harus positif."
23               << endl;
24      }
25  }
26
```

Parameter `BankAccount& acc` mendeklarasikan referensi ke struktur `BankAccount`. Ketika fungsi memodifikasi `acc.amount`, perubahan tersebut mempengaruhi object asli dalam array, bukan salinan. Ini berbeda dengan parameter pass-by-value yang akan membuat salinan dan perubahan tidak akan terlihat di caller.

Referensi dalam C++ bersifat immutable setelah inisialisasi; setelah referensi `acc` terikat pada object tertentu, tidak dapat diubah untuk mereferensikan object lain. Referensi juga tidak dapat bernilai null, berbeda dengan pointer.

Fungsi `modify_account()` memanfaatkan referensi untuk mengubah multiple members:

```
12
13  void modify_account(BankAccount &acc) {
14      cout << "\nMengubah No. Akun: " << acc.accno
15           << endl;
16      cout << "Masukkan nama baru: ";
17      cin.ignore();
18      getline(cin, acc.name);
19      cout << "Masukkan alamat baru: ";
20      getline(cin, acc.address);
21      cout << "Masukkan tipe akun baru (s/c): ";
22      cin >> acc.actype;
23      cout << "\nAkun berhasil diubah." << endl;
24  }
25
```

Dalam fungsi ini, `cin.ignore()` membersihkan karakter newline dari buffer input setelah `cin >> actype` pada operasi sebelumnya, memastikan `getline()` bekerja dengan benar. Referensi memungkinkan modifikasi langsung terhadap anggota-anggota struktur di dalam array.

4.4 Implementasi Stream Input-Output

Stream input-output menggunakan object global `cin` dan `cout` dari header `<iostream>`:

```
1   #include <iostream>
1   using namespace std;
2
3   int main() {
4       cout << "iostream included and namespace std "
5               "used."
6            << endl;
7       return 0;
8   }
```

Operator ekstraksi >> membaca data dari input stream:

```
3
4   int main() {
5       int choice;
6       cout << "Masukkan nomor pilihan: ";
7       cin >> choice;
8       cout << "Pilihan Anda: " << choice << endl;
9       return 0;
10  }
```

Untuk membaca string dengan spasi, digunakan `getline()`:

```
3
4   int main() {
5       string name;
6       cout << "Masukkan nama lengkap: ";
7       cin.ignore();
8       getline(cin, name);
9       cout << "Nama Anda: " << name << endl;
10      return 0;
11  }
```

Operator insersi << menampilkan data ke output stream:

```
7
8   int main() {
9       BankAccount acc = {5000.0f};
10      cout << "Saldo Baru = " << acc.amount << endl;
11      return 0;
12  }
```

4.5 Implementasi Kontrol Alur dengan Switch-Case

Menu utama program menggunakan `switch-case` untuk mengarahkan eksekusi berdasarkan pilihan user:

```
26
27    switch (choice) {
28    case 1:
29      acc_idx = find_available_slot(accounts,
30                                    MAX_ACCOUNTS);
31      if (acc_idx != -1) {
32        create_new_account(accounts[acc_idx]);
33        cout << "Akun Berhasil Dibuat..." << endl;
34      } else {
35        cout << "Batas akun terlampaui. Tidak dapat "
36                "membuat akun baru."
37             << endl;
38      }
39      break;
40    case 2:
41      // Logika setoran
42      break;
43    case 8:
44      exit(0);
45    default:
46      cout << "Pilihan tidak valid. Silakan coba "
47              "lagi."
48           << endl;
49    }
50
```

Statement `break` penting untuk menghentikan eksekusi dan keluar dari switch block. Tanpa `break`, eksekusi akan terus ke case berikutnya (fall-through), yang umumnya tidak diinginkan.

Case `8` memanggil `exit(0)` untuk menghentikan program dengan status 0 (normal termination). `cstdlib` header menyediakan fungsi `exit()`.

Case `default` menangani pilihan yang tidak valid. Struktur switch-case ini membuat menu-driven program yang mudah dipahami dan diperluas.

4.6 Implementasi Algoritma Pencarian Linear

Fungsi `find_account_idx()` mengimplementasikan pencarian linear untuk menemukan akun berdasarkan nomor akun:

```
 8
 9  int find_account_idx(int acc_num,
10                       const BankAccount accounts[],
11                       int size) {
12      for (int i = 0; i < size; ++i) {
13          if (accounts[i].accno == acc_num) {
14              return i;
15          }
16      }
17      return -1;
18  }
19
```

Fungsi menerima tiga parameter: nomor akun yang dicari (`acc_num`), array akun (`accounts` dengan qualifier `const` karena fungsi hanya membaca), dan ukuran array (`size`).

Loop `for` iterasi dari indeks 0 sampai `size - 1`. Setiap iterasi membandingkan `accounts[i].accno` dengan `acc_num`. Ketika match ditemukan, fungsi segera mengembalikan indeks `i`. Jika seluruh array diperiksa tanpa menemukan match, fungsi mengembalikan −1 sebagai sentinel value yang menunjukkan akun tidak ditemukan.

Caller menggunakan return value ini untuk pemeriksaan:

```
29
30      acc_idx = find_account_idx(acc_num, accounts,
31                                 MAX_ACCOUNTS);
32      if (acc_idx != -1) {
33          deposit(accounts[acc_idx]);
34      } else {
35          cout << "Akun tidak ditemukan." << endl;
36      }
37      return 0;
38  }
```

Pemeriksaan `acc_idx != -1` menentukan apakah pencarian berhasil. Jika berhasil, akun pada indeks `acc_idx` dapat diakses untuk operasi lebih lanjut.

4.7 Implementasi Fungsi Validasi dan Utilitas

Fungsi `is_account_empty()` digunakan untuk memeriksa apakah slot akun kosong:

```
7
8   bool is_account_empty(const BankAccount &acc) {
9     return acc.accno == 0;
10  }
11
12  int main() {
13    BankAccount acc1 = {0};
14    BankAccount acc2 = {101};
15    cout << "Account 1 is empty: " << boolalpha
16         << is_account_empty(acc1) << endl;
```

Fungsi ini menggunakan konvensi bahwa nomor akun 0 menunjukkan slot kosong (tidak valid). Return type `bool` memberikan hasil boolean yang dapat langsung digunakan dalam kondisi if.

Fungsi `find_available_slot()` mencari slot kosong pertama:

```
11
12  int find_available_slot(
13      const BankAccount accounts[], int size) {
14    for (int i = 0; i < size; ++i) {
15      if (is_account_empty(accounts[i])) {
16        return i;
17      }
18    }
19    return -1;
20  }
21
```

Fungsi ini memanfaatkan `is_account_empty()` untuk memeriksa setiap slot. Ketika slot kosong ditemukan, indeks tersebut dikembalikan. Jika semua slot penuh, −1 dikembalikan.

Fungsi `initialize_account()` mereset akun ke state kosong:

```
11
12  void initialize_account(BankAccount &acc) {
13    acc.accno = 0;
14    acc.name = "";
15    acc.address = "";
16    acc.actype = '\0';
17    acc.amount = 0.0;
18  }
19
```

Penggunaan parameter referensi memungkinkan modifikasi object asli. Semua anggota diatur ke nilai default: nomor akun 0, string kosong, karakter null, dan saldo 0.0.

BAB V

KESIMPULAN

Sistem manajemen akun bank yang dikembangkan dalam proyek ini mendemonstrasikan penerapan konsep-konsep fundamental pemrograman C++ secara komprehensif. Melalui implementasi, pembaca telah mempelajari dan memahami:

1. Penggunaan struktur data `struct` untuk abstraksi dan enkapsulasi atribut-atribut logis ke dalam composite type yang bermakna. Struct `BankAccount` mengelompokkan lima anggota dengan tipe berbeda menjadi kesatuan yang merepresentasikan entitas akun bank.
2. Penerapan array statis sebagai kontainer untuk menyimpan multiple instances dari struktur dengan kapasitas tetap. Array ini memungkinkan akses O(1) ke setiap akun melalui subscript operator, meskipun dengan trade-off kapasitas terbatas.
3. Implementasi fungsi dengan parameter referensi yang memungkinkan fungsi memodifikasi data asli tanpa overhead pembuatan salinan. Referensi memberikan semantik yang lebih intuitif dibandingkan pointer untuk kasus-kasus seperti ini.
4. Penggunaan stream input-output berbasis object `cin` dan `cout` untuk komunikasi interaktif dengan user. Operator ekstraksi dan insersi memudahkan pembacaan dan penulisan data dengan tipe otomatis conversion.
5. Desain menu interaktif berbasis `switch-case` yang memungkinkan user memilih dari operasi yang berbeda-beda. Struktur ini scalable dan mudah untuk menambahkan operasi baru.
6. Implementasi algoritma pencarian linear untuk menemukan akun dalam array. Meskipun kompleksitas O(n), algoritma ini cukup efisien untuk dataset berukuran kecil dan mudah dipahami.
7. Pemahaman tentang convention dan best practices dalam desain program, seperti menggunakan sentinel value (-1) untuk menunjukkan kondisi khusus, dan `const` qualifier untuk parameter yang tidak dimodifikasi.

Proyek ini memberikan fondasi yang solid untuk memahami konsep pemrograman tingkat pemula dan untuk melanjutkan ke topik-topik lebih advanced seperti linked list, dynamic memory allocation, file I/O, dan object-oriented programming. Sistem yang dikembangkan dapat diperluas dengan fitur-fitur tambahan seperti penyimpanan data ke file, enkripsi password, atau analisis transaksi historis.

DAFTAR PUSTAKA

LAMPIRAN A

lampiran001.png (hasil run) lampiram002.png etc..