**Praktikum**

**Algoritma dan Pemrograman**

****

**POLITEKNIK TELKOM**

**BANDUNG**

**2009**

**Penyusun**

Dahliar Ananda

Ahmad Suryan

Paramita Mayadewi

Lutce Rasiana

Hendra Kusmayadi

**Editor**

Ade Hendraputra

Dilarang menerbitkan kembali, menyebarluaskan atau menyimpan baik sebagian maupun seluruh isi buku dalam bentuk dan dengan cara apapun tanpa izin tertulis dari Politeknik Telkom.

**Hak cipta dilindungi undang-undang @ Politeknik Telkom 2009**

*No part of this document may be copied, reproduced, printed, distributed, modified, removed and amended in any form by any means without prior written authorization of Telkom Polytechnic.*

# Kata Pengantar

Assalamu’alaikum Wr. Wb

Segala puji bagi Allah SWT karena dengan karunia-Nya c*ourseware* ini dapat diselesaikan.

Atas nama Politeknik Telkom, kami sangat menghargai dan ingin menyampaikan terima kasih kepada penulis, penerjemah dan penyunting yang telah memberikan tenaga, pikiran, dan waktu sehingga c*ourseware* ini dapat tersusun.

Tak ada gading yang tak retak, di dunia ini tidak ada yang sempurna, oleh karena itu kami harapkan para pengguna buku ini dapat memberikan masukan perbaikan demi pengembangan selanjutnya.

Semoga *courseware* ini dapat memberikan manfaat dan membantu seluruh Sivitas Akademika Politeknik Telkom dalam memahami dan mengikuti materi perkuliahan di Politeknik Telkom.

Amin.

Wassalamu’alaikum Wr. Wb.

Bandung, Agustus 2009

Christanto Triwibisono

Wakil Direktur I

Bidang Akademik & Pengembangan

Daftar Isi

[Kata Pengantar iii](#_Toc237609520)

[Daftar Isi iv](#_Toc237609521)

[1. Tipe Data, *Operator* dan Runtunan 1](#_Toc237609522)

[1.1 *Tipe Data Dasar* 2](#_Toc237609523)

[1.2 *Variabel* 3](#_Toc237609524)

[1.3 *Konstanta* 6](#_Toc237609525)

[1.4 *Operator* 7](#_Toc237609526)

[2. Pemilihan 14](#_Toc237609527)

[3. Pengulangan 30](#_Toc237609528)

[*3.1.* *Konsep Pengulangan* 31](#_Toc237609529)

[*3.2.* *Sintaks WHILE* 31](#_Toc237609530)

[3.3 Sintaks DO…WHILE 37](#_Toc237609535)

[3.4 Sintaks FOR 41](#_Toc237609536)

[3.5 *Sintaks Pengulangan Bersarang* 46](#_Toc237609537)

[3.6 *Sintaks BREAK dan CONTINUE* 49](#_Toc237609538)

[4 Array dan Record 56](#_Toc237609539)

[4.1 Array 57](#_Toc237609540)

[4.1.1 *Array* Satu Dimensi 57](#_Toc237609541)

[4.1.2 *Array* Dua Dimensi 60](#_Toc237609542)

[4.2 *Record* 62](#_Toc237609543)

[Output yang dihasilkan 62](#_Toc237609544)

[4.2.1 Penggunaan *Record* dalam *Array* 63](#_Toc237609545)

[*4.2.2* Penggunaan *Array* dalam *Record* 64](#_Toc237609546)

[4.2.3 *Array* *Record* yang mengandung *Array* 66](#_Toc237609547)

[5. *Pemrograman Modular* 70](#_Toc237609548)

[*Variabel Lokal dan Variabel Global* 71](#_Toc237609549)

[Variabel Lokal 71](#_Toc237609550)

[Variabel Global 71](#_Toc237609551)

[*Fungsi* 71](#_Toc237609552)

[*Prosedur* 73](#_Toc237609553)

[*Fungsi Rekursif* 75](#_Toc237609554)

[*Unit* 77](#_Toc237609555)

[6. Mesin Karakter 79](#_Toc237609556)

[*Penyiapan Mesin* 80](#_Toc237609557)

[*Penggunaan Mesin* 81](#_Toc237609558)

[1. Menghitung Jumlah Karakter 81](#_Toc237609559)

[2. Menghitung Jumlah Karakter Tertentu 82](#_Toc237609560)

[3. Menghitung Jumlah Kata 83](#_Toc237609561)

[4. Studi Kasus Palindrom 84](#_Toc237609562)

[”KAPAS” dan ”PASAK” 🡪 anagram 89](#_Toc237609563)

[7. Pencarian 90](#_Toc237609564)

[7.1 Konsep Pencarian 91](#_Toc237609568)

[7.2 *Pencarian Sekuensial* 91](#_Toc237609569)

[7.3 *Pencarian Biner* 98](#_Toc237609570)

[7.4 *Pencarian Lain* 113](#_Toc237609571)

[*8.* *Pengurutan (Sorting)* 116](#_Toc237609572)

[8.1 Pengertian Sort 117](#_Toc237609574)

[8.2 Bubble Sort 117](#_Toc237609575)

[8.3 Selection Sort 119](#_Toc237609576)

[Algoritma *Selection sort* memilih elemen maksimum/minimum *array*, lalu menempatkan elemen maksimum/minimum itu pada awal atau akhir *array* (tergantung pada urutannya *ascending*/*descending*). 119](#_Toc237609577)

[8.3.1 *Maximum* *Selection Sort Ascending* 119](#_Toc237609578)

[8.3.2 *Maximum* *Selection Sort Descending* 121](#_Toc237609579)

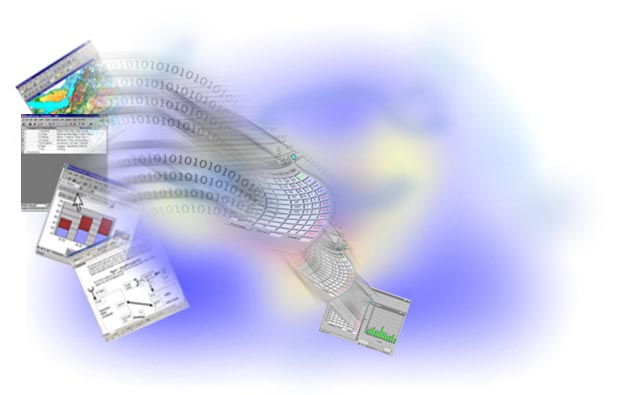
[8.3.3 *Minimum* *Selection Sort Ascending* 122](#_Toc237609580)

[8.3.4 *Minimum* *Selection Sort Descending* 123](#_Toc237609581)

[8.4 *Insertion Sort* 124](#_Toc237609582)

[Daftar Pustaka vi](#_Toc237609583)

# Tipe Data, *Operator* dan Runtunan



|  |  |
| --- | --- |
|  | **Overview** |
| Tipe data, operator, dan runtunan merupakan suatu kesatuan konsep yang paling mendasar didalam pemprograman komputer, karena tipe-tipe data dasar dan operator dapat membentuk berbagai macam ekspresi yang akan digunakan dalam program.  Sedangkan runtunan merupakan konsep dasar yang dapat memberikan gambaran tentang cara kerja sebuah program dalam komputer. | |

|  |  |
| --- | --- |
|  | **Tujuan** |
| 1. Mengenal dan membedakan tipe-tipe data dasar 2. Memahami dan dapat menggunakan tipe-tipe data dasar dalam program 3. Memahami dan dapat menggunakan operator dan penggunaannya dalam program 4. Memahami dan dapat menggunakan konsep runtunan dalam program | |

## *Tipe Data Dasar*

Tipe data dasar adalah tipe data yang dapat langsung digunakan. Secara umum terdapat 2 tipe data dasar, yaitu **numerik** dan **kategorik**. Tipe data numerik terdiri atas angka/ kumpulan angka serta dapat mengalami operasi perhitungan, sedangkan tipe data kategorik dapat berupa angka maupun huruf namun tidak dapat mengalami operasi perhitungan.

Berikut merupakan contoh beberapa tipe data dasar :

* **Integer/ bilangan bulat**

Integer adalah tipe data dasar berupa bilangan yang tidak mengandung pecahan desimal.

Contoh integer: 2 5 -10 -135 2008

Dalam pemrograman yang menggunakan bahasa pemprograman C++, secara umum dikenal beberap macam tipe data integer, yaitu:

*Tabel 1. Tipe data integer*

|  |  |  |
| --- | --- | --- |
| **Tipe** | **Ukuran** | **Nilai** |
| **Short** | 8 bit | -128 .. 127 |
| **Int** | 16 bit | -32768 .. 32767 |
| **Long** | 32 bit | -2147483648 .. 2147483647 |

* **Float/ bilangan pecahan**

Float adalah tipe data dasar berupa bilangan yang memiliki pecahan desimal. Dalam pemrograman, nilai dengan tipe data ini harus ditulis dengan sebuah titik sebagai pemisah bilangan utuh dan bilangan pecahannya.

Contoh real: .5 0.17 -3.465 92.0 4.3000+E9

Dalam pemrograman, secara umum dikenal beberapa macam tipe data real, yaitu:

*Tabel 2. Tipe data real*

|  |  |  |
| --- | --- | --- |
| **Tipe** | **Ukuran** | **Nilai** |
| **float** | 32 bit | 2.9x10-39 .. 1.7x1038 |
| **Double** | 48 bit | 5.0x10-324 .. 1.7x10308 |

Nilai pada tabel diatas berbeda dengan nilai yang ada pada tabel tipe data integer, pada tabel diatas nilai untuk tipe data merupakan tingkat ketelitian untuk masing-masing tipe data, bukan berdasarkan rentang nilai.

* **Char/ Karakter**

Char adalah tipe data dasar yang terdiri atas satu buah angka, huruf, tanda baca atau karakter khusus. Untuk menyimpan sebuah karakter, diperlukan 1 byte atau 8 bit tempat didalam memori. Dalam sebuah program, penulisan tipe data char diawali dan diakhiri dengan tanda kutip ganda.

Contoh char: “5” “A” “?” “+” “$”

Perhatikan bahwa 5 adalah integer sedangkan “5” adalah char.

* **String**

String adalah tipe data dasar yang berupa kumpulan karakter dengan panjang tertentu. Panjang string minimal adalah 0 artinya string tersebut bernilai *null*. Sama halnya dengan penulisan karakter, penulisan sebuah string juga harus diawali dan diakhiri dengan tanda petik ganda.

Contoh string:

* “Politeknik Telkom Bandung”
* “30202001”
* “z”
* “”
* **Boolean/ bilangan logika**

Sebuah data boolean memiliki tepat dua buah kemungkinan nilai, yang biasanya direpresentasikan sebagai Benar dan Salah, atau True dan False, atau dapat juga dilambangkan dengan 1 dan 0.

## *Variabel*

Variabel atau peubah adalah obyek yang nilainya dapat berubah-ubah dalam sebuah program. Variabel harus dideklarasikan sebelum digunakan didalam program dengan tujuan untuk ‘memesan’ tempat dengan ukuran tertentu (sesuai tipe datanya) pada memori. Kita dapat memberikan nama yang mencirikan variabel tersebut untuk mempermudah pemanggilan di dalam program. Pada saat mendeklarasikan sebuah variabel, pemrogram harus menyebutkan nama variabel dan tipe data dari variabel tersebut.

Sebelum kita menuliskan beberapa program dalam bahasa C++, ada baiknya kita mengenal terlebih dahulu struktur dan format penulisan program dalam bahasa C++.

Buatlah contoh progrm dibawah ini :

1. // Contoh Program C++
2. #include <stdio.h>
3. #include <conio.h>
4. /\* Program
5. Utama \*/
6. main () {
7. printf("Selamat Datang");
8. getch();
9. return 0;
10. }

Jika dijalankan, maka hasil dari program tersebut akan menampilkan kata “Selamat Datang”.

Pada contoh program diatas :

* Pada baris pertama dituliskan diawalannya tanda *doubleslash.* Maksudnya adalah sebagai komentar, artinya baris tersebut tidak akan dieksekusi oleh program. Selain dengan menggunakan “//”, kita juga dapat menggunakan “/\* ... \*/” dimana kita dapat menuliskan komentar diantara tanda tersebut. Dengan menggunakan yang kedua, kita dapat menuliskan komentar lebih dari 1 baris, seperti terlihat pada baris ke 3 dan 4.
* Pada baris kedua, code #include <stdio.h>, yang diawali dengan tanda crash (#). Ini berfungsi untuk memberitahukan kepada preprocessor untuk meng-includekan file stdio.h, karena file tersebut akan digunakan didalam program untuk melakukan perintah.
* Pada baris ke 6 – 10 merupakan isi dari program. Dimana program utama merupakan fungsi dengan nama main(). Program utama adalah tempat dimana kode yang pertama kali dieksekusi saat program dijalankan. Fungsi main tersebut harus ada untuk setiap kali kita membuat program. Pada baris ke-7 berfungsi untuk mencetak tulisak kelayar, baris ke-8 agar instruksi berhenti sehingga hasil dilayar dapat dilihat karena program tidak langsung berakhir, baris ke-9 merupakan nilai kembali dari fungsi main, sedangkan 10 adalah penutup fungsi main.

Hal yang perlu diingat adalah : Pemprograman bahasa C++ dalah *case sensitive*, artinya penulisan huruf besar berbeda dengan penulisan huruf kecil.

Untuk menuliskan variabel, kita dapat menuliskannya pada bagian isi program, dimana format menuliskannya adalah penulisan tipe data terlebih dahulu, selanjutnya diikuti oleh spasi kemudian nama variabel. Contoh program penulisan variabelnya adalah :

1. #include <stdio.h>
2. #include <conio.h>
3. main () { // Program Utama
4. // Deklarasi Variabel
5. string sNama;
6. int iUmur;
7. char jKelamin;
9. // Membaca Inputan
10. printf(“Masukan Nama : ”);
11. scanf(“%i”,&sNama);
12. printf(“Masukan Umur : ”);
13. scanf(“%i”,&iUmur);
14. printf(“Masukan Jenis Kelamin (L/P) : ”);
15. scanf(“%i”,&jKelamin);
16. // Menampilkan Inputan
17. printf(“Nama : %s”, sNama);
18. printf(“Umur : %i”, iUmur);
19. printf(“Jenis Kelamin : %c”, jKelamin);
20. getch();
21. }

Pengisian nilai ke dalam sebuah variabel dalam sebuah program dapat dilakukan dengan 2 cara, yaitu:

* **Secara langsung**

Contoh program memberikan nilai secara langsung :

1. #include <stdio.h>
2. main () { // Program Utama
3. // Deklarasi Variabel
4. int x;
5. float TB;
6. bool status;
8. // Inisialisasi Nilai secara Langsung
9. x = 10;
10. TB = 25.654;
11. status = true;
12. // .....
13. }

* **Dengan inputan**

Contoh program memberikan nilai secara langsung :

1. #include <stdio.h>
2. main () { // Program Utama
3. // Deklarasi Variabel
4. int x;
5. string nama;
7. // Meminta Inputan Nilai
8. scanf(“%i”, &x);
9. scanf(“%s”, &nama);
10. // ...
11. }

## *Konstanta*

Berbeda dengan variabel, konstanta memiliki nilai yang tetap dan tidak dapat diubah sejak dideklarasikan hingga program berakhir.

Cara penulisan konstanta didalam program, di tulis dengan diawali dengan tanda crash (#) kemudian diikuti dengan define, selanjutnya nama konstantanya dan selanjutnya nilainya dan ditulis diluar program utama setelah pendeklarasian librari namespace. Contoh penulisannya adalah sebagai berikut :

1. ...
2. #define iMaxs 100
3. #define rPhi 3.14159
4. #define sSapa ‘Hello’
5. #define newLine ‘\n’
6. ...

Contoh program menghitung luas lingkaran berikut :

1. #include <stdio.h>
2. #include <conio.h>
3. #define Pi 3.14159
4. #define NewLine ‘\n’
5. main () {
6. int jari;
7. float luas;
8. printf(“Program Menghitung Luas Lingkaran %c”, NewLine);
9. printf(“Masukan Jari-Jari Lingkaran”);
10. scanf(“%i”, &jari);
11. luas = Pi \* jari \* jari;
12. printf(“Luas Lingkaran dengan Jari-jari %i adalah %f“, jari, luas);
13. getch();
14. }

Pada baris ke-4 kita mendefinisikan “\n” dengan nama NewLine. Didalam pemprograman dengan bahasa C++ kita mengelal istilah karakter spesial. Karakter ini dikatakan spesial karena karakter ini tidak terdapat pada keyboard. Untuk menuliskan karakter spesial diawali dengan tanda *backslash* (\), kemudian diikuti karakter yang melambangkannya. Misalnya :

* \n : untuk Baris Baru (enter)
* \t : untuk tabulasi (tab)
* \r : akhir dari string
* \0 : *null ternimated* dalam array
* \a : untuk menghasilkan bunyi *beep*
* Dll.

## *Operator*

Operator adalah pengendali operasi yang akan dilakukan pada beberapa operan sehingga membentuk sebuah ekspresi.

* **Operator aritmatik**

|  |  |  |
| --- | --- | --- |
| **Lambang** | **Deskripsi** | **Contoh** |
| **+** | Penjumlahan | x = y + z |
| **-** | Pengurangan | x = y – z |
| **\*** | Perkalian | x = y \* z |
| **/** | Pembagian | x = y / z |
| **%** | Modulo (sisa bagi) | x = y % z |

Catatan : Output dari operasi aritmatik akan memiliki tipe data yang sama dengan tipe data kedua operannya.

Contoh program dengan operasi aritmatik:

1. /\* Contoh :
2. Program Aritmatik\*/
3. #include <stdio.h>
4. #include <conio.h>
5. main () {
6. // Deklarasi Variabel
7. int iAngka1, iAngka2;
8. int iTambah,iKurang,iMod;
9. float fBagi;
10. printf(“Masukan Angka Pertama : ”);
11. scanf(“%i”, &iAngka1);
12. printf(“Masukan Angka kedua : ”);
13. scanf(“%i”, &iAngka2);
14. // Penjumlahan
15. iTambah = iAngka1 + iAngka2;
16. iKurang = iAngka1 - iAngka2;
17. iMod = iAngka1 % iAngka2;
18. ifBagi = iAngka1 / iAngka2;
19. printf(“Hasil %i + %i adalah %i”, iAngka1, iAngka2, iTambah);
20. printf(“Hasil %i - %i adalah %i”, iAngka1, iAngka2, iKurang);
21. printf(“Hasil %i mod %i adalah %i”, iAngka1, iAngka2, iMod);
22. printf(“Hasil %i / %i adalah %f”, iAngka1, iAngka2, fBagi);
23. return 0;
24. getch();
25. }

* **Operator *Assignment***

Dalam pemprograman bahasa C++, Operator ini digunakan memasukan nilai kedalam sebuah variabel, tanpa menghilangkan atau mengosongkan nilai variabel sebelumnya. Contoh penggunaan operator ini adalah sebagai berikut :

*Tabel 4. Operator relasional*

|  |  |  |
| --- | --- | --- |
| **Lambang** | **Deskripsi** | **Contoh** |
| **+=** | Menambahkan | x += 1 |
| **-=** | Mengurangkan | x -= 1 |
| **\*=** | Mengalikan | x \*= 2 |
| **/=** | Membagi | x /= 2 |
| **%=** | Mem-mod | x %= 2 |

Contoh :

1. #include <stdio.h>
2. #include <conio.h>
3. main () {
4. int i = 5;
5. printf(“Nilai i sebelumnya : %i”,i)
6. i += 3;
7. printf(“Nilai i setelahnya : %i”,i)
8. getch();
9. }

* **Increase and decrease**

Penulisan ini dilambangkan dengan ++ (Increade) dan -- (decrease). Operator ini berfungsi untuk menaikan atau menurunkan satu satuan nilai pada sebuah variabel.

Ada dua macam penulisan operator ini, yaitu simbol dapat ditulis sebelum nama variabel dan setelah variabel. Adapun perbedaab antara keduanya adalah :

|  |  |
| --- | --- |
| 1. #include <stdio.h> 2. #include <conio.h> 3. main () { 4. int b = 3; 5. int a; 6. a = ++b; 7. printf(“Nilai A = %i \n Nilai B = %i”,a,b); 8. getch(); 9. return 0; 10. } | 1. #include <stdio.h> 2. #include <conio.h> 3. main () { 4. int b = 3; 5. int a; 6. a = b++; 7. printf(“Nilai A = %i \n Nilai B = %i”,a,b); 8. getch(); 9. return 0; 10. } |

Perhatikan perbedaan kedua program tersebut dan perhatikan perbedaan hasilnya.

* **Operator relasional**

Operator ini membandingkan dua operan dan hasilnya berupa nilai boolean (BENAR atau SALAH). Operasi relasional dapat dilakukan pada dua nilai dengan tipe data yang sama: tipe data integer, riil, char, string, maupun boolean. Tipe data ini sering dugunakan dalam pengkondisian. Berikut ini adalah operator relasional:

*Tabel 4. Operator relasional*

|  |  |  |
| --- | --- | --- |
| **Lambang** | **Deskripsi** | **Contoh** |
| **==** | Sama dengan | x == y |
| **!=** | Tidak sama dengan | x != y |
| **>** | Lebih dari | x > y |
| **<** | Kurang dari | x < y |
| **>=** | Lebih dari atau sama dengan | x >= y |
| **<=** | Kurang dari atau sama dengan | x <= y |

Contoh penggunaan operator relasional dalam program :

1. #include <stdio.h>
2. #include <conio.h>
3. main () {
4. int iAngka1, iAngka2;
5. iAngka1 = 6 //pengisian variabel langsung
6. printf(“Masukan Angka Tebakan : ”);
7. scanf(“%i”, &iAngka2);
8. if (iAngka1 == iAngka2) {
9. printf(“Tebakan Anda Benar”);
10. } else {
11. printf(“Tebakan Anda Salah”);
12. }
13. getch();
14. }

* **Operator logika**

Operator logika adalah operator yang digunakan untuk mengkombinasikan hasil ekspresi yang mengandung operator relasional.

Tiga macam operator logika adalah:

*Tabel 5. Operator logika*

|  |  |  |
| --- | --- | --- |
| **Lambang** | **Deskripsi** | **Contoh** |
| **&&** | And / Dan | x > 7 && x = y |
| **||** | Or / Atau | x != y || x > 3 |
| **!** | Not / Tidak | ! (x > y) |

Pola penggunaan operator logika adalah:

*ekspresi1* **OPERATOR** *ekspresi2*

Output dari penggunaan operator AND dan OR adalah sebagai berikut:

*Tabel 6. Output operator logika*

|  |  |  |  |
| --- | --- | --- | --- |
| **ekpresi1** | **ekspresi2** | **Kombinasi dengan** | |
| **AND** | **OR** |
| True | True | True | True |
| True | False | False | True |
| False | True | False | True |
| False | False | False | False |

Pola yang mudah untuk mengingat output kedua operator logika tersebut adalah: True AND True = True, False OR False = False.

Contoh penggunaan operator logika dalam program :

1. #include <stdio.h>
2. #include <conio.h>
3. main () {
4. int iNilai;
5. printf(“Masukan Nilai : ”);
6. scanf(“%i”, & iNilai);
7. if (iNilai <= 100 && iNilai >= 0) {
8. printf(“Nila Valid”);
9. } else {
10. printf(“Nilai Tidak Valid”);
11. }
12. getch();
13. }

|  |  |  |
| --- | --- | --- |
|  |  | |
|  | | **Latihan** | |

1. Buatlah program persamaan linier yang menerima inputan x dan menghasilkan nilai y dengan persamaan y = 3x + 4 :
2. Buatlah program menghitung keliling lingkaran dimana program menerima inputan jari-jari lingkaran.
3. Buatlah program menghitung lias segitiga dimana program menerima inputan lebar alas dan tinggi.
4. Buatlah program yang memeriksa apakah sebuah garis persamaan linier bersinggungan dengan persamaan x = y dimana inputan program adalah persamaan parameter.
5. Buatlah program yang dapat menghitung rentang waktu untuk perjalanan kendaraan, dimana program menerima inputan kecepatan perjalanan dan jarak tempuh perjalanan.
6. perbaiki program berikut ini sehinga bisa dijalankan :

#include <stdio.h>

main () {

int iPertama, iKedua;

float fHasil;

printf(”Masukan Bilangan Pertama : ”);

scanf(iPertama);

printf(”Masukan Bilangan Pertama : ”);

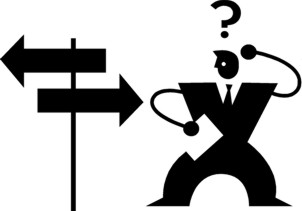
scanf(iKedua);

fHasil = iPertama / iKedua;

printf(”Hasilnya : ”,fHasil);

}

# Pemilihan



|  |  |
| --- | --- |
|  | **Overview** |
| Program dapat merepresentasikan situasi pemilihan yang sering dihadapi dalam dunia nyata. Berdasarkan satu atau beberapa kondisi, dapat ditentukan satu atau sejumlah aksi yang akan dilakukan. Dengan adanya struktur pemilihan, program dapat berjalan dengan jalur yang berbeda, berdasarkan hasil pengecekan kondisi yang dipenuhi. | |

|  |  |
| --- | --- |
|  | **Tujuan** |
| 1. Memahami struktur pemilihan dalam program 2. Mengenal struktur IF dan CASE yang dapat digunakan dalam pemilihan 3. Memahami konsep kondisi dan aksi dalam struktur pemilihan 4. Menerapkan pemilihan dalam menyelesaikan berbagai kasus | |

* 1. **Konversi Struktur IF dan CASE ke Bahasa C**

Berikut ini diberikan pedoman konversi dari algoritma ke dalam bahasa C untuk struktur IF dan CASE:

|  |  |
| --- | --- |
| Algoritma | Bahasa C |
| If kondisi then  Aksi  End if | if (kondisi) {  Aksi;  } |
| If kondisi then  Aksi1  Else  Aksi2  End if | If (kondisi) {  Aksi1;  }  else {  Aksi2;  } |
| If kondisi1 then  Aksi1  Else if kondisi2  Aksi2  Else  Aksi3  End if | if (kondisi1) {  Aksi1;  }  else if (kondisi2){  Aksi2;  }  else {  Aksi3;  } |
| Case ekspresi  Nilai1: Aksi1  Nilai2: Aksi2  Nilai3: Aksi3  End case | switch (ekspresi) {  case Nilai1: Aksi1;  Break;  case Nilai2: Aksi2;  Break;  case Nilai3: Aksi3;  } |
| Case ekspresi  Nilai1: Aksi1  Nilai2: Aksi2  Nilai3: Aksi3  Otherwise: Aksi4  End case | switch (ekspresi) {  case Nilai1: Aksi1;  Break;  case Nilai2: Aksi2;  Break;  case Nilai3: Aksi3;  Break;  default: Aksi4;  } |
| Case ekspresi  Nilai-1,Nilai-2,Nilai-3: Aksi1  Nilai-4,Nilai-5: Aksi2  Nilai-6..Nilai-8: Aksi3  Otherwise: Aksi4  End Case | switch (ekspresi) {  case Nilai1:  case Nilai2:  case Nilai3: Aksi1;  Break;  case Nilai4:  case Nilai5: Aksi2;  Break;  case Nilai6:  case Nilai7:  case Nilai8: Aksi3;  Break;  default: Aksi4;  } |

Catatan:

- penulisan kondisi pada IF dan ekspresi pada CASE dalam bahasa C harus

digunakan tanda kurung ( ).

- aksi berupa satu perintah atau lebih, masing-masing diakhiri titik koma.

- apabila aksi hanya berupa satu perintah, penggunaan { } dapat dihilangkan.

- kata “if”, “else”, “switch”, “case” dan “default” dalam bahasa C, harus

ditulis dengan huruf kecil semua.

- dalam bahasa C tidak ada kata “then”, “end if” dan “end case” tetapi

digantikan pasangan kurung kurawal { dan }

- hati-hati dengan penggunaan kesamaan, yaitu dengan “==” bukan “=”.

- string digunakan kutip dua ( seperti “test” ) bukan kutip satu (‘test’).

* 1. **Contoh dan Latihan Struktur IF**

Contoh-1: Algoritma menentukan apakah positip atau negatip dari bilangan yang diinput.

|  |  |
| --- | --- |
| Algoritma | Bahasa C |
| Algoritma PosNeg  /\* menentukan positip/negatip\*/  Kamus Data  bil : integer  ket : string  Begin  |Input(bil)  |If (bil>=0) then  | Output(‘positip’)  |Else  | Output(‘negatip’)  |End if  End | **include <stdio.h>**  **include <conio.h>**  /\* menentukan positip/negatip \*/  **int main() {**  //Kamus Data  int bil;  //Begin  printf(“Masukkan bilangan: ”);  scanf(“%d”,&bil);  if (bil>=0) {  printf(“bil adalah positip”);  }  else {  printf(“bil adalah negatip”);  }  getche();  } |
| Algoritma PosNeg  /\* menentukan positip/negatip\*/  Kamus Data  bil : integer  ket : string  Begin  |Input(bil)  |Ket 🡨 ‘positip’  |If (bil<0) then  | Ket 🡨 ‘negatip’  |End if  |Output(Ket)  End | **#include <stdio.h>**  **#include <conio.h>**  /\* menentukan positip/negatip \*/  **int main() {**  //Kamus Data  int bil;  char Ket[10];  //Begin  printf(“Masukkan bilangan: ”);  scanf(“%d”,&bil);  strcpy(Ket,”positip”);  if (bil<0) {  strcpy(Ket,”negatip”);  }  printf(“bil adalah %s”,Ket);  getche();  } |

Latihan-1: Tulis secara lengkap program bahasa C untuk potongan algoritma berikut.

|  |  |
| --- | --- |
| Algoritma-1a  |Input(bil)  |If (bil>=0) then  | Output(‘positip’)  |End if  |If (bil<0) then  | Output(‘negatip’)  |End if | Algoritma-1b  |Input(bil)  |positip 🡨 bil>=0  |If (positip=true) then  | Output(‘positip’)  |else  | Output(‘negatip’)  |End if |

Contoh-2: Algoritma menentukan bilangan terbesar dari 3 bilangan yang diinput.

|  |  |
| --- | --- |
| Algoritma Terbesar\_1  /\* menentukan terbesar dari 3 bilangan yang diinput\*/  Kamus Data  A,B,C : integer  Begin  |Input(A,B,C)  |If (A>B) then  | |If (A>C) then  | | Output(‘terbesar =’,A)  | |Else  | | Output(‘terbesar =’,C)  | |End if  |else  | |if (B>C) then  | | Output(‘terbesar =’,B)  | |Else  | | Output(‘terbesar =’,C)  | |End if  |End if  End | **#include <stdio.h>**  **#include <conio.h>**  /\* menentukan terbesar 3 bilangan \*/  **int main() {**  //Kamus Data  int A,B,C;  //Begin  printf(“Masukkan 3 bilangan: ”);  scanf("%d %d %d",&A,&B,&C);  if (A>B) {  if (A>C) {  printf(“terbesar adalah %d”,A);  }  else {  printf(“terbesar adalah %d”,C);  }  }  else {  if (B>C) {  printf(“terbesar adalah %d”,B);  }  else {  printf(“terbesar adalah %d”,C);  }  }  getche();  } |
| Algoritma Terbesar\_2  /\* menentukan terbesar dari 3 bilangan yang diinput\*/  Kamus Data  A,B,C : integer  Begin  |Input(A,B,C)  |If (A>B and A>C) then  | |Output(‘terbesar =’,A)  |Else if (B>A and B>C) then  | |Output(‘terbesar =’,B)  |Else if (C>A and C>B) then  | |Output(‘terbesar =’,C)  |End if  End | **#include <stdio.h>**  **#include <conio.h>**  /\* menentukan terbesar 3 bilangan \*/  **int main() {**  //Kamus Data  int A,B,C;  //Begin  printf(“Masukkan 3 bilangan: ”);  scanf("%d %d %d",&A,&B,&C);  if (A>B && A>C) {  printf(“terbesar adalah %d”,A);  }  else if (B>A && B>C) {  printf(“terbesar adalah %d”,B);  }  else {  printf(“terbesar adalah %d”,C);  }  getche();  } |

Latihan-2: Tulis secara lengkap program bahasa C untuk potongan algoritma berikut.

|  |  |
| --- | --- |
| Algoritma-2a  |Input(A,B,C)  |Max 🡨A  |If (B>Max) then  | |Max 🡨B  |End if  |If (C>Max) then  | |Max 🡨C  |End if  |Output(‘terbesar = ‘,Max) | Algoritma-2b  |Input(A,B,C)  |If (A>B and A>C) then  | |Output(‘terbesar =’,A)  |Else  | |if (B>A and B>C) then  | | Output(‘terbesar =’,B)  | |Else  | | if (C>A and C>B) then  | | Output(‘terbesar =’,C)  | | |End if  | |End if  |End if |

Contoh-3 : Pembayaran air minum PDAM

PDAM menerapkan pembayaran air minum perumahan dengan cara perhitungan sebagai berikut :

- Tarif per m3 untuk 10 m3 pertama (1-10) adalah Rp 2.000

- Tarif per m3 untuk 10 m3 kedua (11-20) adalah Rp 3.000

- Tarif per m3 untuk 10 m3 ketiga (21-30) adalah Rp 4.000

- Tarif per m3 untuk 10 m3 selanjutnya (31 ke atas) adalah Rp 5.000

- Pemakaian air dihitung minimal 10 m3 (kurang dari 10 m3 dianggap 10 m3)

- Biaya administrasi bulanan sebesar Rp10.000

Bagaimana membuat algoritma untuk menghitung biaya tersebut?

Contoh kasus

Penggunaan air 5 m3 dengan biaya 10 x 2.000 + 10.000 = 30.000

Penggunaan air 15 m3 dengan biaya 10 x 2.000 + 5 x 3.000 + 10.000 = 45.000

Penggunaan air 75 m3 dengan biaya 10 x 2.000 + 10 x 3.000 +

10 x 4.000 + 45 x 5.000 +10.000 = 325.000

Solusi :

Pemakaian air dibagi menjadi 4 area pemakaian (misal area a,b,c,d), baru dihitung total biaya

|  |  |
| --- | --- |
| Algoritma | Bahasa C |
| Algoritma PDAM  /\* menghitung biaya pemakaian air\*/  Kamus Data  pakai,a,b,c,d : integer  biaya : integer  Begin  |Input(pakai)  |a🡨10  |b🡨 0  |c🡨 0  |d🡨 0  |If (pakai>30) then  | |b🡨 10  | |c🡨 10  | |d🡨 pakai - 30  |Else If (pakai>20) then  | |b🡨 10  | |c🡨 pakai - 20  |Else If (pakai>10) then  | |b🡨 pakai - 10  |End if  |biaya🡨 a \* 2000 + b \* 3000 +  | c \* 4000 + d \* 5000 +  | 10000  |Output(‘biaya =’,biaya)  End | **#include <stdio.h>**  **#include <conio.h>**  /\* menghitung biaya pemakaian air\*/  **int main() {**  //Kamus Data  int pakai,a,b,c,d;  int biaya;  //Begin  printf(“Masukkan pemakaian air: ”);  scanf(“%d”,&pakai);  a=10;  b=0;  c=0;  d=0;  if (pakai>30) {  b=10;  c=10;  d=pakai – 30;  }  else if (pakai>20) {  b=10;  c=pakai – 20;  }  else if (pakai>10) {  b=pakai – 10;  }  biaya = a \* 2000 + b \* 3000 +  c \* 4000 + d \* 5000 +  10000;  printf(“biaya = %d”,biaya);  **getche();**  **return 0;**  //End  **}** |

Latihan-3: Tulis secara lengkap program bahasa C untuk potongan algoritma berikut.

Indeks nilai sebuah matakuliah didapat dengan cara menghitung nilai akhir berdasarkan prosentase komponen-komponennya kemudian ditentukan indeks nilainya. Misal digunakan ketentuan sebagai berikut:

- Nilai Akhir dihitung dari 30% UTS, 40%UAS, 20% Tugas dan 10% kehadiran

- Indeks Nilai ditentukan berdasarkan Nilai Akhir (NA),

Bila NA >= 85 maka Indeksnya A

Bila 85 > NA >= 70 maka Indeksnya B

Bila 70 > NA >= 55 maka Indeksnya C

Bila 55 > NA >= 40 maka Indeksnya D

Bila NA < 40 maka Indeksnya E

Bagaimana membuat algoritma untuk menentukan Indeks Nilai tersebut?

|  |  |
| --- | --- |
| Algoritma Nilai  /\* menentukan indeks nilai\*/  Kamus Data  UTS,UAS,Tugas,Abs : integer  NAS : real  Indeks : char  Begin  |Input(UTS,UAS,Tugas,Abs)  |NAS 🡨 0.3\*UTS + 0.4\*UAS +  | 0.2\*Tugas + 0.1\*ABS  |If (NAS>=85) then  | |Indeks 🡨 ‘A’  |Else If (NAS>=70 and NAS<85)  | |then  | |Indeks 🡨 ‘B’  |Else If (NAS>=55 and NAS<70)  | |then  | |Indeks 🡨 ‘C’  |Else If (NAS>=40 and NAS<55)  | |then  | |Indeks 🡨 ‘D’  |Else  | |Indeks 🡨 ‘E’  |End if  |Output(‘Indeks Nilai =’,Indeks)  End | **#include <stdio.h>**  **#include <conio.h>**  /\* menentukan indeks nilai\*/  **int main() {**  //Kamus Data  int UTS,UAS,Tugas,Abs;  float NAS;  char Indeks;  //Begin  printf(“Masukan nilai UTS,UAS,”);  printf(“Tugas dan kehadiran: ”);  scanf(“%d %d”,&UTS,&UAS);  scanf(“%d %d”,&Tugas,&Abs);  NAS = 0.3\*UTS + 0.4\*UAS +  0.2\*Tugas + 0.1\*Abs;  if (NAS>=85)  Indeks = ‘A’;  else if (NAS>=70)  Indeks = ‘B’;  else if (NAS>=55)  Indeks = ‘C’;  else if (NAS>=40)  Indeks = ‘D’;  else {  Indeks = ‘E’;  }  printf(“Indeks Nilai = %c”,Indeks);  getche();  } |

* 1. **Contoh dan Latihan Struktur IF**

Contoh-4 : Menentukan nama hari dari nomor hari yang diinput

Dinput nomor hari, ditampilkan nama harinya, bagaimana algoritmanya?

Solusi dengan IF dan CASE

|  |  |
| --- | --- |
| Algoritma NamaHari  /\* menentukan nama hari\*/  Kamus Data  NoHari : integer  NmHari : string  Begin  |Input(NoHari)  |Case NoHari  | |1: NmHari 🡨 ‘Senin’  | |2: NmHari 🡨 ‘Selasa’  | |3: NmHari 🡨 ‘Rabu’  | |4: NmHari 🡨 ‘Kamis’  | |5: NmHari 🡨 ‘Jumat’  | |6: NmHari 🡨 ‘Sabtu’  | |7: NmHari 🡨 ‘Minggu’  |End Case  |Output(NmHari)  End | **#include <stdio.h>**  **#include <conio.h>**  /\* menentukan nama hari\*/  **int main() {**  //Kamus Data  int NoHari;  char NmHari[10];  //Begin  printf(“Masukkan nomor hari: ”);  scanf(“%d”,&NoHari);  switch (NoHari) {  case 1: strcpy(NmHari,”Senin”); break;  case 2: strcpy(NmHari,”Selasa”);break;  case 3: strcpy(NmHari,”Rabu”); break;  case 4: strcpy(NmHari,”Kamis”);break;  case 5: strcpy(NmHari,”Jumat”); break;  case 6: strcpy(NmHari,”Sabtu”); break;  case 7: strcpy(NmHari,”Mingu”);  }  printf(“Nama hari = %s ”,NmHari);  getche();  } |

Latihan-4 : Merubah angka menjadi kalimat

Dinput bilangan/angka (angka dibatasi 1-99), ditampilkan kata-kata/kalimat dari bilangan tersebut, ubah potongan algoritma berikut ke bahasa C yang lengkap.

|  |
| --- |
| |Input(bil)  |pul 🡨 bil div 10  |sat 🡨 bil mod 10  |Kalimat 🡨 ‘’  |Case sat  | |1: Kalimat 🡨 ‘Satu’  | |2: Kalimat 🡨 ‘Dua’  | |3: Kalimat 🡨 ‘Tiga’  | |4: Kalimat 🡨 ‘Empat’  | |5: Kalimat 🡨 ‘Lima’  | |6: Kalimat 🡨 ‘Enam’  | |7: Kalimat 🡨 ‘Tujuh’  | |8: Kalimat 🡨 ‘Delapan’  | |9: Kalimat 🡨 ‘Sembilan’  |End Case  |Case pul  | |1: |Case sat  | | | 0: Kalimat 🡨‘Sepuluh’  | | | 1: Kalimat 🡨‘Sebelas’  | | | Otherwise: Kalimat 🡨 Kalimat + ‘ belas’  | | |End Case  | |2: |Kalimat 🡨 ‘Dua Puluh’ + Kalimat  | |3: Kalimat 🡨 ‘Tiga Puluh’ + Kalimat  | |4: Kalimat 🡨 ‘Empat Puluh’ + Kalimat  | |5: Kalimat 🡨 ‘Lima Puluh’ + Kalimat  | |6: Kalimat 🡨 ‘Enam Puluh’ + Kalimat  | |7: Kalimat 🡨 ‘Tujuh Puluh’ + Kalimat  | |8: Kalimat 🡨 ‘Delapan Puluh’ + Kalimat  | |9: Kalimat 🡨 ‘Sembilan Puluh’ + Kalimat  |End Case  |Output(Kalimat) |

**4.4. Latihan Tambahan**

1. Genap Ganjil

Tulis program untuk menentukan apakah bilangan yang diinput oleh user genap atau ganjil.

Contoh masukan :  
bilangan = 10

Contoh keluaran :  
bilangan 10 adalah genap

2. Terbesar dari 2 bilangan

Tulis program untuk menentukan bilangan terbesar dari dua bilangan yang diinput oleh user.

Contoh masukan :  
bilangan ke-1 = 30

bilangan ke-2 = 25

Contoh keluaran :  
bilangan terbesar adalah 30

3. Terkecil dari 2 bilangan

Tulis program untuk menentukan bilangan terkecil dari dua bilangan yang diinput oleh user.

Contoh masukan :  
bilangan ke-1 = 30

bilangan ke-2 = 25

Contoh keluaran :  
bilangan terkecil adalah 25

4. Lama Bekerja (selisih jam masuk dan jam pulang)

Tulis program untuk menentukan lama bekerja seorang pegawai, jika jam masuk dan jam pulang diinput. Catatan: jam berupa angka 1-12, dan seorang pegawai bekerja kurang dari 12 jam.

Contoh Masukan dan keluaran:

|  |  |  |
| --- | --- | --- |
| Jam masuk | Jam keluar | Keluaran/tampilan |
| 10 | 11 | Lama bekerja 1 jam |
| 10 | 2 | Lama bekerja 3 jam |
| 10 | 7 | Lama bekerja 9 jam |

5. Biaya Parkir

Tulis program untuk menentukan biaya parkir yang dihitung berdasarkan lama parkir. Lama parkir dihitung dari selisih jam masuk dan jam keluar diinput (seperti soal 4). Biaya parkir 2 jam pertama 2000, perjam berikutnya 500.

Contoh Masukan dan keluaran:

|  |  |  |  |
| --- | --- | --- | --- |
| Jam masuk | Jam keluar | lama | keluaran/tampilan |
| 10 | 11 | 1 | Biaya = 2000 |
| 10 | 2 | 3 | Biaya = 2500 |
| 10 | 7 | 9 | Biaya = 4500 |

6. Penggabungan resistor

Tulis program untuk menghitung resistor pengganti dari 3 buah resistor yang disusun secara seri atau paralel. Rumus untuk resistor pengganti sbb:

Seri: Rgab = R1 + R2 + R3

Paralel: 1/Rgab = 1/R1 + 1/R2 + 1/R3

Besar R1, R2 dan R3 serta jenis penyusunan (seri/paralel) diinput oleh user.

7. Buatlah program untuk menyelesaikan masalah berikut :  
Program akan menerima masukan berupa kode, jenis dan harga, dengan  
jenis adalah "A", "B", dan "C". Untuk setiap jenis, masing-masing akan diberikan diskon sebesar 10% untuk A,15% untuk B, dan 20% untuk C.   
Program akan menghitung berapa harga setelah didiskon.  
Contoh masukan :  
kode = 10  
Jenis = B  
harga = 10000  
Contoh keluaran :  
Jenis barang B mendapat diskon = 15%, Harga setelah didiskon = 8500

8. Kabisat

Buatlah program untuk menentukan apakah masukan tersebut  
merupakan tahun kabisat atau bukan (Tahun kabisat adalah tahun yang  
angka tahunnya habis dibagi 4).  
Contoh masukan : 1995  
Contoh keluaran  : 1995 bukan tahun kabisat

8. Gaji Karyawan Mingguan

Sebuah perusahaan swasta menggaji karyawannya secara mingguan dengan hitungan sebagai berikut :

golongan 1 dengan upah per jam 3.000 rupiah

golongan 2 dengan upah per jam 3.500 rupiah

golongan 3 dengan upah per jam 4.000 rupiah

golongan 4 dengan upah per jam 5.000 rupiah

Bila seorang karyawan bekerja kurang atau sama dengan 40 jam per minggu, akan dihitung dengan upah per jam seperti di atas, tetapi apabila bekerja lebih dari 40 jam, maka lebihnya akan dihitung sebagai lembur dengan upah per jam 1½ kali upah biasa.

Tulis algoritma untuk menghitung gaji mingguan karyawan, bila **golongan** dan **jam kerja** diinput dari keyboard.

9. Menghitung lama dalam jam dan menit

Buat program menghitung selisih waktu (dalam menit) berdasarkan jam mulai (jam,menit) dengan jam selesai (jam,menit). Data yang diinput adalah jam dan menit mulai, jam dan menit selesai.

Contoh Masukan dan Keluaran:

|  |  |  |  |
| --- | --- | --- | --- |
| Mulai (jam,menit) | Selesai (jam,menit) | Lama (menit) | Lama (jam,menit) |
| 10 30 | 12 10 | 100 | 1 jam 40 menit |
| 10 30 | 1 40 | 130 | 3 jam 10 menit |

Petunjuk :

* Hitung selisih jam seperti soal sebelumnya, kemudian kalikan dg 60 untuk mendapatkan menitnya.
* Hitung selisih menit mulai dengan selesai, bila selisihnya positip tambahkan ke hasil sebelumnya, bila selisihnya negatip kurangkan ke hasil sebelumnya.

10. Kuadran dari koordinat titik

Buat program dengan masukan koordinat sebuah titik (x,y) dalam sebuah system koordinat kartesian, kemudian ditentukan di kuadran mana titik tersebut.

Contoh Masukan :

Koordinat titik (x,y) : 10 -20

Contoh keluaran:

Titik 10,-20 berada pada kuadran 4

11. Menaikkan 1 menit

Buat program dengan masukan berupa jam dan menit, kemudian ditentukan setelah ditambah satu menit menjadi jam berapa.

Contoh Masukan dan Keluaran:

|  |  |
| --- | --- |
| Waktu asal  (jam,menit) | Setelah ditambah 1 menit (jam,menit) |
| 10 30 | 10 31 |
| 10 59 | 11 0 |

12. Tunjangan Anak

Pemerintah hanya memberikan tunjangan anak dari PNS maksimal sampai 3 anak saja. Setiap anak diberikan tunjangan sebesar 10% dari gaji pokok.

Tulis program untuk menghitung total tunjangan anak, bila gaji pokok dan jumlah anak diinput oleh user.

Contoh Masukan dan Keluaran:

|  |  |  |
| --- | --- | --- |
| Jumlah anak | Gaji Pokok | Tunjangan Anak |
| 0 | 1.000.000 | 0 |
| 1 | 1.000.000 | 100.000 |
| 3 | 1.000.000 | 300.000 |
| 5 | 1.000.000 | 300.000 |

13. Minibus pariwisata

Sebuah acara wisata digunakan minibus kapasitas 7 penumpang per mobil. Apabila terjadi kelebihan penumpang (walaupun cuma 1), maka minibus yang digunakan ditambah 1. Tulis program untuk menghitung jumlah minibus yang diperlukan, bila jumlah pesertanya diinput.

Contoh Masukan dan Keluaran:

|  |  |
| --- | --- |
| Jumlah Peserta | Jumlah Minibus |
| 5 | 1 |
| 7 | 1 |
| 8 | 2 |

14. Menentukan nama hari untuk N hari yang lalu

Diinput jumlah berapa hari yang lalu, kemudian ditentukan nama harinya. Tulis program untuk menentukan nama hari dari sekian hari yang lalu yang diinput. (Catatan, gunakan Hari ini = ‘Senin’)

Contoh Masukan dan Keluaran:

|  |  |
| --- | --- |
| N hari yang lalu | Nama Hari |
| 0 | Senin |
| 1 | Minggu |
| 10 | Jumat |
| 20 | Selasa |

15. Menentukan nama hari untuk N hari yang lalu

Diinput jumlah berapa hari yang lalu, kemudian ditentukan nama harinya. Tulis program untuk menentukan nama hari dari sekian hari yang lalu dan nomor hari ini yang diinput.

Contoh Masukan dan Keluaran:

|  |  |  |
| --- | --- | --- |
| N hari yang lalu | Hari ini | Nama Hari |
| 0 | 1 | Senin |
| 3 | 3 | Minggu |
| 5 | 4 | Sabtu |

16. Menentukan nama hari untuk N hari yang akan datang

Diinput jumlah berapa hari yang akan datang, kemudian ditentukan nama harinya. Tulis program untuk menentukan nama hari dari sekian hari yang akan datang yang diinput. (Catatan, gunakan Hari ini = ‘Senin’)

Contoh Masukan dan Keluaran:

|  |  |
| --- | --- |
| N hari yang akan datang | Nama Hari |
| 0 | Senin |
| 1 | Selasa |
| 10 | Kamis |
| 20 | Minggu |

17. Menentukan terbesar dari 5 bilangan yang diinput

Diinput jumlah 5 buah bilangan, kemudian ditentukan bilangan mana yang terbesar.

Contoh Masukan dan Keluaran:

|  |  |
| --- | --- |
| Bilangan diinput | Terbesar |
| 10 30 25 50 13 | 50 |
| 80 30 25 50 35 | 80 |

18. Menentukan nama bulan dari nomor bulan yang diinput

Diinput nomor bulan, kemudian ditentukan nama bulannya.

Contoh Masukan dan Keluaran:

|  |  |
| --- | --- |
| Nomor bulan | Nama bulan |
| 1 | Januari |
| 5 | Mei |

19. Menentukan nomor urut tanggal dalam 1 tahun (Julian Date)

Diinput nomor tanggal, bulan dan tahun, kemudian ditentukan nomor urutnya dalam 1 tahun.

Contoh Masukan dan Keluaran:

|  |  |  |  |
| --- | --- | --- | --- |
| Tanggal | Bulan | tahun | Nomor urut |
| 1 | 1 | 2000 | 1 |
| 5 | 5 | 2000 | 126 |

20. Menentukan kalimat dari bilangan 1-999

Diinput bilangan (dibatasi antara 1 s/d 999), kemudian ditentukan kalimatnya.

Contoh Masukan dan Keluaran:

|  |  |
| --- | --- |
| bilangan | kalimat |
| 123 | seratus dua puluh tiga |
| 511 | lima ratus sebelas |

}

# Pengulangan



|  |  |
| --- | --- |
|  | **Overview** |
| Pengulangan (*Loop*) merupakan sebuah konsep yang penting dalam pemrograman. Dengan struktur pengulangan, program dapat berjalan beberapa kali sesuai inisialisasi, jumlah iterasi dan kondisi berhenti yang ditentukan. Hal ini dapat menyederhanakan program yang kompleks menjadi lebih sederhana. Dalam C disediakan berbagai perintah *Loop*, dimana setiap perintah *loop* memiliki keunikan tersendiri. Di dalam bab ini akan dipraktrekkan tentang struktur pengulangan dalam bahasa C serta contoh soal dan latihan menggunakan pengulangan. | |

|  |  |
| --- | --- |
|  | **Tujuan** |
| 1. Mengimplementasikan struktur pengulangan dalam bahasa C. 2. Menerapkan sintaks-sintaks pengulangan dalam menyelesaikan persoalan 3. Mahasiswa mampu menyelesaikan persoalan tentang pengulangan | |

## *Konsep Pengulangan*

Pengulangan merupakan sebuah konsep pemrograman yang penting karena konsep ini memungkinkan pengguna menggunakan sekumpulan baris program berulang kali dengan tiga komponen yang mengendalikannya, yaitu:

* **Inisialisasi**; menentukan kondisi awal dilakukannya pengulangan.
* **Jumlah iterasi**; menunjukkan berapa kali pengulangan akan dilakukan.
* **Kondisi berhenti**; menentukan kondisi yang dapat mengakhiri pengulangan.

Ketika mengimplementasikan dalam program, ketiga komponen ini tidak selalu dapat didefinisikan dalam struktur pengulangan. Mungkin saja salah satu komponen tersebut tidak didefinisikan. Pengulangan tetap dapat berjalan, asal komponen yang tidak didefinisikan tersebut dapat diketahui secara tersirat berdasarkan komponen lain yang didefinisikan. Hal lain yang perlu diperhatikan adalah bahwa **pengulangan harus berhenti**. Jika pengulangan tidak pernah berhenti, maka logika program salah. Pengulangan akan berhenti jika jumlah iterasi yang diminta sudah tercapai atau kondisi berhenti bernilai benar. Maka, dalam setiap pengulangan, pemrogram perlu menentukan jumlah iterasi atau kondisi berhenti dan langkah pencapaian menuju kondisi berhenti tersebut.

Pada bab ini akan dijelaskan 3 struktur perulangan di dalam C, yaitu struktur perulangan While , For, dan Do While

## *Sintaks WHILE*

Pengulangan dengan menggunakan WHILE merupakan sebuah pengulangan yang dikendalikan oleh suatu kondisi tertentu, dimana kondisi tersebut yang akan menentukan apakah perulangan itu akan terus dilaksanakan atau dihentikan. Kondisi tersebut akan dicek disetiap awal iterasi, apakah sebuah kondisi terpenuhi atau tidak. Jika kondisi terpenuhi (bernilai benar), maka iterasi akan dilanjutkan. Jika kondisi tidak terpenuhi, maka iterasi dihentikan.

Perulangan dengan WHILE dapat digunakan pada struktur perulangan yang diketahui jumlah iterasinya dan juga pada struktur perulangan yang tidak diketahui jumlah iterasinya, tetapi harus selalu terdapat kondisi berhenti.

Bentuk umum pernyataan ***while*** pada C :

|  |
| --- |
| while (kondisi)  {  //pernyataan;  } |

Contoh implementasi pengulangan dengan while :

* + - 1. Pengulangan untuk menampilkan kalimat “Halo apa khabar ? “ sebanyak 5 kali.

|  |
| --- |
| 1. #include <stdio.h> 2. main() 3. { 4. int i; //deklarasi variabel pencacah 5. i=0; //inisialisasi variabel pencacah 6. while(i<5) //pengecekan kondisi 7. { 8. printf("\nHalo apa khabar ?"); 9. i++; //penambahan bilangan pencacah 10. } 11. } |

Output sintaks diatas :

|  |
| --- |
| Halo apa khabar ?  Halo apa khabar ?  Halo apa khabar ?  Halo apa khabar ?  Halo apa khabar ? |

* + - 1. Pengulangan untuk menampilkan bilangan 1 hingga 5

|  |
| --- |
| 1. #include <stdio.h> 2. main() 3. { 4. int i=1; 5. while(i<=5) 6. { 7. printf("\nBilangan ke-%d",i); 8. i++; 9. } 10. } |

Output sintaks diatas

|  |
| --- |
| Bilangan ke-1  Bilangan ke-2  Bilangan ke-3  Bilangan ke-4  Bilangan ke-5 |

* + - 1. Pengulangan untuk meminta user memasukkan nilai

|  |
| --- |
| 1. #include <stdio.h> 2. main() 3. { 4. int i=1; 5. int input; 6. while(i<=5) 7. { 8. printf("\nMasukkan bilangan ke-%d : ",i); 9. scanf("%d",&input); 10. printf("Bilangan yang Anda masukkan : %d",input); 11. i++; 12. } 13. } |

Output dari sintaks diatas bila diinputkan : 4, 5, 6, 2, dan 9.

|  |
| --- |
| Masukkan bilangan ke-1 : 4  Bilangan yang Anda masukkan : 4  Masukkan bilangan ke-2 : 5  Bilangan yang Anda masukkan : 5  Masukkan bilangan ke-3 : 6  Bilangan yang Anda masukkan : 6  Masukkan bilangan ke-4 : 2  Bilangan yang Anda masukkan : 2  Masukkan bilangan ke-5 : 9  Bilangan yang Anda masukkan : 9 |

Contoh no 1-3 merupakan contoh bentuk pengulangan yang telah diketahui jumlah iterasi yaitu 5 kali. Jumlah iterasi yang telah diketahui akan menjadi kondisi yg digunakan untuk mengecek apakah perulangan akan dilanjutkan atau tidak.

Berikut adalah contoh dimana jumlah perulangan akan ditentukan oleh user yaitu dengan cara meminta inputan dari user.

* + - 1. Modifikasi contoh no 3 dimana user diminta untuk menentukan jumlah perulangan yang akan dilakukan

|  |
| --- |
| 1. #include <stdio.h> 2. main() 3. { 4. int jml,i; 5. int input; 6. printf("Jumlah Bilangan : "); 7. scanf("%d",&jml); 8. i=1; 9. while(i<=jml) 10. { 11. printf("\nMasukkan bilangan ke-%d : ",i); 12. scanf("%d",&input); 13. printf("Bilangan yang Anda masukkan : %d",input); 14. i++; 15. } 16. } |

Telah dikatakan di awal bahwa perulangan dengan WHILE dapat digunakan untuk struktur pengulangan yang belum diketahui jumlah iterasinya, tetapi tetap mempunyai kondisi berhenti.

Berikut adalah contoh pengulangan yang belum diketahui jumlah iterasinya, tetapi tetap mempunyai kondisi berhenti.

* + - 1. Menampilkan menu sesuai pilihan user

|  |
| --- |
| 1. #include <stdio.h> 2. main() 3. { 4. int pil=1; //inisialisasi kondisi 5. while(pil!=5) 6. { 7. printf("\nMENU : "); 8. printf("\n1. Red"); 9. printf("\n2. Blue"); 10. printf("\n3. Yellow"); 11. printf("\n4. Green"); 12. printf("\n5. Exit"); 13. printf("\nPilihan : "); 14. scanf("%d",&pil); 15. switch(pil) 16. { 17. case 1 : printf("Anda memilih Merah");break; 18. case 2 : printf("Anda memilih Biru");break; 19. case 3 : printf("Anda memilih Kuning");break; 20. case 4 : printf("Anda memilih Hijau");break; 21. case 5 : printf("Anda memilih Keluar ... bye");break; 22. default : printf("Pilihan anda salah"); 23. } 24. } 25. } |

Output dari sintaks diatas bila menu yg dipilih 2, 4, 6, dan 5

|  |
| --- |
| MENU :  1. Red  2. Blue  3. Yellow  4. Green  5. Exit  Pilihan : 2  Anda memilih Biru  MENU :  1. Red  2. Blue  3. Yellow  4. Green  5. Exit  Pilihan : 4  Anda memilih Hijau  MENU :  1. Red  2. Blue  3. Yellow  4. Green  5. Exit  Pilihan : 6  Pilihan anda salah  MENU :  1. Red  2. Blue  3. Yellow  4. Green  5. Exit  Pilihan : 5  Anda memilih Keluar ... bye |

Contoh diatas merupakan pengulangan yang belum diketahui jumlah iterasinya tetapi mempunyai kondisi yang akan menyebabkan pengulangan dihentikan, yaitu ketika user memasukkan pilihan 5. Selama user tidak memilih angka 5, maka pengulangan akan tetap dilakukan.

* + - 1. User input angka hingga memilih keluar, angka yang telah dimasukkan akan dijumlah dan dihitung jumlah iterasi yang telah dilakukan.

|  |
| --- |
| 1. #include <stdio.h> 2. main() 3. { 4. int i, input, jmlInput; 5. i=0; 6. jmlInput=0; 7. input=0; //inisialisasi kondisi 8. while (input != 99 ) 9. { 10. printf("Masukkan angka (input '99' untuk keluar) : "); 11. scanf("%d",&input); 12. jmlInput=jmlInput+input; 13. i++; 14. } 15. printf("\nAnda telah input angka sebanyak %d kali",i-1); 16. printf("\nJumlah inputan anda : %d",jmlInput -99); 17. } |

Output

|  |
| --- |
| Masukkan angka (input '99' untuk keluar) : 5  Masukkan angka (input '99' untuk keluar) : 6  Masukkan angka (input '99' untuk keluar) : 8  Masukkan angka (input '99' untuk keluar) : 4  Masukkan angka (input '99' untuk keluar) : 99  Anda telah input angka sebanyak 4 kali  Jumlah inputan anda : 23 |

Pada contoh ke-5, variable ‘i’ digunakan untuk menghitung jumlah iterasi yang telah dilakukan (iterasi memasukkan angka yg dijumlah). Variabel ‘input’ digunakan untuk menampung inputan dari user, dan variabel ‘jmlInput’ digunakan untuk menampung hasil penjumlahan tiap inputan user. Kondisi berhenti yang digunakan adalah jika user memasukkan angka 99, maka pengulangan akan dihentikan. Selama user tidak memasukkan angka 99, maka pengulangan akan diteruskan. Pada program contoh no 5, variabel ‘input’ diinisialisasi dengan 0, hal ini dilakukan akan pada pengecekan kondisi akan memberikan nilai true sehingga pengulangan akan dijalankan untuk menampilkan Menu.

Bentuk pengulangan yang membutuhkan agar pengulangan dilakukan tanpa adanya pengecekan kondisi di awal akan lebih cocok jika menggunakan struktur *do...while* daripada struktur *while*.



## Sintaks DO…WHILE

Sintaks DO... WHILE... melakukan pengulangan serupa dengan sintaks WHILE. Penggunaan sintaks ini juga tidak harus menyebutkan jumlah pengulangan yang harus dilakukan, karena dapat digunakan untuk perulangan dengan jumlah iterasinya yang belum diketahui, tetapi harus mempunyai kondisi berhenti.

Bedanya, jika pada sintaks WHILE kondisi dievaluasi/ diuji sebelum aksi pengulangan dilakukan, sedangkan pada sintaks DO...WHILE pengujian kondisi dilakukan setelah aksi pengulangan dilakukan.

Bentuk umum pernyataan ***do...while*** pada C adalah:

|  |
| --- |
| do  {  //pernyataan  }while(kondisi); |

Pada struktur pengulangan dengan sintaks DO... WHILE..., aksi akan terus dilakukan hingga kondisi yang dicek di akhir pengulangan, bernilai benar. Dengan sintaks ini, pengulangan pasti dilakukan minimal satu kali, yakni pada iterasi pertama sebelum pengecekan kondisi. WHILE dengan DO WHILE seringkali memberikan hasil yang sama, tetapi ada kalanya hasilnya akan berbeda, sehingga harus berhati-hati dalam penggunaan kondisi antara WHILE dengan DO WHILE.

Berikut ini adalah contoh penggunaan sintaks do...while pada C dengan studi kasus serupa dengan contoh yang terdapat pada struktur while :

* + - 1. Pengulangan untuk menampilkan kalimat “Halo apa khabar ? “ sebanyak 5 kali.

|  |
| --- |
| 1. #include <stdio.h> 2. main() 3. { 4. int i; //deklarasi variabel pencacah 5. i=0; //inisialisasi variabel pencacah 6. do 7. { 8. printf("\nHalo apa khabar ?"); 9. i++; //penambahan bilangan pencacah 10. } 11. while(i<5); //pengecekan kondisi 12. } |

* + - 1. Pengulangan untuk menampilkan bilangan 1 hingga 5

|  |
| --- |
| 1. #include <stdio.h> 2. main() 3. { 4. int i=1; 5. do 6. { 7. printf("\nBilangan ke-%d",i); 8. i++; 9. } 10. while(i<=5); 11. } |

* + - 1. Pengulangan untuk meminta user memasukkan nilai

|  |
| --- |
| 1. #include <stdio.h> 2. main() 3. { 4. int i=1; 5. int input; 6. do 7. { 8. printf("\nMasukkan bilangan ke-%d : ",i); 9. scanf("%d",&input); 10. printf("Bilangan yang Anda masukkan : %d",input); 11. i++; 12. } 13. while(i<=5); 14. } |

* + - 1. Modifikasi contoh no 9 dimana user diminta untuk menentukan jumlah perulangan yang akan dilakukan

|  |
| --- |
| 1. #include <stdio.h> 2. main() 3. { 4. int jml,i; 5. int input; 6. printf("Jumlah Bilangan : "); 7. scanf("%d",&jml); 8. i=1; 9. do( 10. { 11. printf("\nMasukkan bilangan ke-%d : ",i); 12. scanf("%d",&input); 13. printf("Bilangan yang Anda masukkan : %d",input); 14. i++; 15. } 16. while(i<=jml); 17. } |

Sintaks do...while juga dapat digunakan untuk pengulangan yang belum diketahui jumlah iterasinya dan contoh berikut juga akan memperlihatkan keuntungan menggunakan sintaks do...while, dimana tidak perlu adanya inisialisasi kondisi agar dapat memasuki pengulangan.

* + - 1. Menampilkan menu sesuai pilihan user

|  |
| --- |
| 1. #include <stdio.h> 2. main() 3. { 4. int pil; //deklarasi kondisi 5. do 6. { 7. printf("\nMENU : "); 8. printf("\n1. Red"); 9. printf("\n2. Blue"); 10. printf("\n3. Yellow"); 11. printf("\n4. Green"); 12. printf("\n5. Exit"); 13. printf("\nPilihan : "); 14. scanf("%d",&pil); 15. switch(pil) 16. { 17. case 1 : printf("Anda memilih Merah");break; 18. case 2 : printf("Anda memilih Biru");break; 19. case 3 : printf("Anda memilih Kuning");break; 20. case 4 : printf("Anda memilih Hijau");break; 21. case 5 : printf("Anda memilih Keluar ... bye");break; 22. default : printf("Pilihan anda salah"); 23. } 24. } 25. while(pil!=5); 26. } |

* + - 1. User input angka hingga memilih keluar, angka yang telah dimasukkan akan dijumlah dan dihitung jumlah iterasi yang telah dilakukan.

|  |
| --- |
| 1. #include <stdio.h> 2. main() 3. { 4. int i, input, jmlInput; 5. i=0; 6. jmlInput=0; 7. do 8. { 9. printf("Masukkan angka (input '99' untuk keluar) : "); 10. scanf("%d",&input); 11. jmlInput=jmlInput+input; 12. i++; 13. } 14. while (input != 99 ); 15. printf("\nAnda telah input angka sebanyak %d kali",i-1); 16. printf("\nJumlah inputan anda : %d",jmlInput-99); 17. } |

## Sintaks FOR

Sintaks pengulangan FOR merupakan sintaks yang relatif paling mudah digunakan. Sintaks ini serupa dengan sintaks WHILE... DO... dalam hal pengecekan kondisi dilakukan di awal. Dalam menggunakan struktur pengulangan dengan sintaks FOR, pemrogram harus mendefinisikan nilai awal dan nilai akhir pencacah yang menunjukkan jumlah iterasi. Setiap kali iterasi berlangsung, nilai pencacah akan diubah. Jika pencacah sudah mencapai nilai akhir yang ditentukan, maka pengulangan akan berhenti.

Bentuk umum pengulangan dengan sintaks ***for*** pada C adalah:

|  |
| --- |
| for (inisialisasi ; kondisi ; perubahan)  {  //pernyataan  } |

Dimana :

* Inisialisasi : untuk memberikan nilai awal untuk variabel pencacah.
* Kondisi : kondisi pengulangan akan berhenti atau tidak.
* Perubahan : pengubahan nilai variabel pencacah untuk mencapai kondisi berhenti, dapat berupa kenaikan ataupun penurunan. Pengubah variabel pencacah tidak harus selalu naik atau turun satu, tetapi dapat dilakukan pengubahan variabel pencacah lebih dari satu.
* Pernyataan : aksi yang akan diulang

Berikut ini adalah contoh penggunaan sintaks for pada C dengan studi kasus serupa dengan contoh yang terdapat pada struktur while dan do ...while :

1. Pengulangan untuk menampilkan kalimat “Halo apa khabar ? “ sebanyak 5 kali.

|  |
| --- |
| 1. #include <stdio.h> 2. main() 3. { 4. int i; //deklarasi variabel pencacah 5. //inisialisasi variabel pencacah; 6. //pengecekan kondisi; 7. //penambahan bilangan pencacah 8. for(i=0;i<5;i++) 9. { 10. printf("\nHalo apa khabar ?"); 11. } 12. } |

1. Pengulangan untuk menampilkan bilangan 1 hingga 5

|  |
| --- |
| 1. #include <stdio.h> 2. main() 3. { 4. int i; 5. for(i=1;i<=5;i++) 6. { 7. printf("\nBilangan ke-%d",i); 8. } 9. } |

1. Pengulangan untuk meminta user memasukkan nilai

|  |
| --- |
| 1. #include <stdio.h> 2. main() 3. { 4. int i; 5. int input; 6. for(i=1;i<=5;i++) 7. { 8. printf("\nMasukkan bilangan ke-%d : ",i); 9. scanf("%d",&input); 10. printf("Bilangan yang Anda masukkan : %d",input); 11. } 12. } |

1. Modifikasi contoh no 15 dimana user diminta untuk menentukan jumlah perulangan yang akan dilakukan

|  |
| --- |
| 1. #include <stdio.h> 2. main() 3. { 4. int jml,i; 5. int input; 6. printf("Jumlah Bilangan : "); 7. scanf("%d",&jml); 8. for(i=1;i<=jml;i++) 9. { 10. printf("\nMasukkan bilangan ke-%d : ",i); 11. scanf("%d",&input); 12. printf("Bilangan yang Anda masukkan : %d",input); 13. } 14. } |

Sintaks for juga dapat digunakan untuk pengulangan yang belum diketahui jumlah iterasinya.

1. Menampilkan menu sesuai pilihan user

|  |
| --- |
| 1. #include <stdio.h> 2. main() 3. { 4. int pil=0; //inisialisasi kondisi 5. for (;pil!=5;) 6. { 7. printf("\nMENU : "); 8. printf("\n1. Red"); 9. printf("\n2. Blue"); 10. printf("\n3. Yellow"); 11. printf("\n4. Green"); 12. printf("\n5. Exit"); 13. printf("\nPilihan : "); 14. scanf("%d",&pil); 15. switch(pil) 16. { 17. case 1 : printf("Anda memilih Merah");break; 18. case 2 : printf("Anda memilih Biru");break; 19. case 3 : printf("Anda memilih Kuning");break; 20. case 4 : printf("Anda memilih Hijau");break; 21. case 5 : printf("Anda memilih Keluar ... bye");break; 22. default : printf("Pilihan anda salah"); 23. } 24. } 25. } |

1. User input angka hingga memilih keluar, angka yang telah dimasukkan akan dijumlah dan dihitung jumlah iterasi yang telah dilakukan.

|  |
| --- |
| 1. #include <stdio.h> 2. main() 3. { 4. int i, input, jmlInput; 5. i=0; 6. jmlInput=0; 7. input=0; 8. for(;input!= 99;) 9. { 10. printf("Masukkan angka (input '99' untuk keluar) : "); 11. scanf("%d",&input); 12. jmlInput=jmlInput+input; 13. i++; 14. } 15. printf("\nAnda telah input angka sebanyak %d kali",i-1); 16. printf("\nJumlah inputan anda : %d",jmlInput-99); 17. } |

Semua struktur pengulangan yang telah dijelaskan diatas dapat menggunakan variabel pencacah dengan hitungan menurun. Berikut adalah contoh penggunaan pencacah mundur untk ketiga struktur diatas

|  |
| --- |
| //sintaks while  #include <stdio.h>  main()  {  int i=5;  printf("Hitung Mundur !!!");  while (i>0)  {  printf("\n%d",i);  i--;  }  printf("\nDORRR!!!!");  } |
| //sintaks do...while  #include <stdio.h>  main()  {  int i=5;  printf("Hitung Mundur !!!");  do  {  printf("\n%d",i);  i--;  }  while (i>0);  printf("\nDORRR!!!!");  } |
| //sintaks for  #include <stdio.h>  main()  {  int i;  printf("Hitung Mundur !!!");  for (i=5;i>0;i--)  {  printf("\n%d",i);  }  printf("\nDORRR!!!!");  } |

Output dari program diatas :

|  |
| --- |
| Hitung Mundur !!!  5  4  3  2  1  DORRR!!!! |

## *Sintaks Pengulangan Bersarang*

Sama halnya dengan struktur pemilihan, struktur pengulangan juga dapat disusun bersarang. Sebuah struktur pengulangan bisa berada dalam struktur pengulangan lainnya. Atau, sebuah struktur pengulangan bisa mengandung struktur pengulangan lain di dalamnya.

Berikut adalah contoh struktur pengulangan bersarang :

* + - 1. Pengulangan while di dalam while

|  |
| --- |
| 1. #include <stdio.h> 2. main() 3. { 4. int i,j; 5. i=1; 6. while(i<=5) 7. { 8. j=1; 9. while(j<=5) 10. { 11. if(i==j) 12. printf("%d",j); 13. else 14. printf(" "); 15. j++; 16. } 17. printf("\n"); 18. i++; 19. } 20. } |

* + - 1. Pengulangan do…while di dalam do…while

|  |
| --- |
| 1. #include <stdio.h> 2. main() 3. { 4. int i,j; 5. i=1; 6. do 7. { 8. j=1; 9. do 10. { 11. if(i==j) 12. printf("%d",j); 13. else 14. printf(" "); 15. j++; 16. } 17. while(j<=5); 18. printf("\n"); 19. i++; 20. } 21. while(i<=5); 22. } |

* + - 1. Pengulangan for di dalam for

|  |
| --- |
| 1. #include <stdio.h> 2. main() 3. { 4. int i,j; 5. for(i=1;i<=5;i++) 6. { 7. for(j=1;j<=5;j++) 8. { 9. if(i==j) 10. printf("%d",j); 11. else 12. printf(" "); 13. } 14. printf("\n"); 15. } 16. } |

* + - 1. Pengulangan gabungan for di dalam while

|  |
| --- |
| 1. #include <stdio.h> 2. main() 3. { 4. int i,j; 5. for(i=1;i<=5;i++) 6. { 7. j=1; 8. while(j<=5) 9. { 10. if(i==j) 11. printf("%d",j); 12. else 13. printf(" "); 14. j++; 15. } 16. printf("\n"); 17. } 18. } |

Setiap struktur pengulangan dapat digabung penggunaannya dengan struktur pengulangan yang lainnya, tetapi tetap harus memperhatikan karakter tiap struktur pengulangan.

Penggabungan penggunaan struktur pengulangan pada pengulangan bersarang dapat menggunakan lebih dari 2 pengulangan, bisa saja terjadi kebutuhan menggunakan struktur bersarang dengan kedalaman 3 atau lebih.

## *Sintaks BREAK dan CONTINUE*

Sintaks *BREAK* dan *CONTINUE* merupakan sintaks yang digunakan untuk menghentikan pengulangan dan melanjutkan ke perintah atau aksi berikutnya. Sintaks BREAK dan CONTINUE dapat digunakan baik di dalam struktur pengulangan WHILE, DO...WHILE, dan FOR.

Sintaks *BREAK* digunakan untuk menghentikan pengulangan kemudian keluar dari struktur pengulangan tanpa melanjutkan perintah di dalam struktur pengulangan. Berikut adalah contoh penggunaan sintaks break :

* + - 1. Menampilkan 6 angka kemudian break ketika mencapai angka 4

|  |
| --- |
| 1. #include <stdio.h> 2. main() 3. { 4. int i; 5. i=1; 6. while(i<=6) 7. { 8. printf("\nBilangan ke-%d",i); 9. if (i==4) 10. { 11. printf("\nOooops...break"); 12. break; 13. } 14. i++; 15. } 16. } |

* + - 1. Menampilkan menu sesuai pilihan user

|  |
| --- |
| 1. #include <stdio.h> 2. main() 3. { 4. int i, input, jmlInput; 5. i=0; 6. jmlInput=0; 7. while (1) 8. { 9. printf("Masukkan angka (input '99' untuk keluar) : "); 10. scanf("%d",&input); 11. if (input==99) 12. { 13. break; 14. } 15. jmlInput=jmlInput+input; 16. i++; 17. } 18. printf("\nAnda telah input angka sebanyak %d kali",i); 19. printf("\nJumlah inputan anda : %d",jmlInput); 20. } |

Program diatas merupakan modifikasi pemberhentian pengulangan dengan menggunakan sintaks Break, perhatikan perbedaan program diatas dengan contoh program no 5.

Pada contoh diatas, didalam while kondisi diberikan angka 1, angka 1 mengindikasikan true dimana sintaks di dalam while akan dijalankan jika kondisi bernilai true. Sehingga while (1) akan menyebabkan pengulangan akan dijalankan terus menerus, padahal setiap pengulangan membutuhkan kondisi berhenti, kondisi berhenti ini tidak dideklarasikan di dalam statement while (kondisi) tetapi berada di dalam tubuh while yaitu if(input==99).Sedangkan untuk mengeluarkan dari pengulangan maka digunakan sintaks break.

Sintaks *CONTINUE* digunakan untuk kembali ke awal pengulangan tanpa menjalankan perintah berikutnya.

Contoh penggunaan sintaks CONTINUE adalah sebagai berikut :

* + - 1. Menampilkan 6 angka, ketika mencapai angka 3 maka akan di teruskan (continue) ke angka berikutnya

|  |
| --- |
| 1. #include <stdio.h> 2. main() 3. { 4. int i; 5. for(i=1;i<=6;i++) 6. { 7. if ( i==4) 8. { 9. printf("\nContinue"); 10. continue; 11. } 12. printf("\nBilangan ke-%d",i); 13. } 14. } |

* + - 1. Contoh aplikasi dengan menggunakan break dan continue

|  |  |
| --- | --- |
| 1. #include <stdio.h> 2. main() 3. { 4. while (1) 5. { 6. char pil; 7. float nil1,nil2,nil3,rerata; 8. int i; 9. printf("\nHitung Rerata\n"); 10. printf("Nilai ke 1 : "); 11. scanf("%f",&nil1); 12. printf("Nilai ke 2 : "); 13. scanf("%f",&nil2); 14. printf("Nilai ke 3 : "); 15. scanf("%f",&nil3); 16. rerata=(nil1+nil2+nil3)/3; 17. printf("Rerata : %f",rerata); 18. printf("\nHitung Rerata lagi(y/n): "); 19. scanf("%s",&pil); 20. if (pil=='y' || pil=='Y') 21. continue; 22. else 23. break; 24. } 25. } | |
|  | | **Rangkuman** | |

1. Struktur pengulangan memungkinkan program melakukan satu atau lebih aksi beberapa kali.
2. Tiga komponen pengendali aksi adalah inisialisasi, jumlah iterasi, dan kondisi berhenti.
3. Tiga struktur pengulangan yang dapat digunakan pada bahasa pemrograman C adalah struktur pengulangan dengan sintaks WHILE, DO...WHILE, dan FOR.
4. Struktur pengulangan dapat dibuat bersarang dengan sintaks pengulangan yang sama atau berbeda, bahkan dapat digabungkan dengan struktur pemilihan.
5. Pemilihan struktur pengulangan yang digunakan disesuaikan dengan kebutuhan, karena dalam beberapa kasus masing-masing struktur pengulangan dapat memberikan hasil yang berbeda.
6. Untuk keluar dari struktur pengulangan sebelum kondisi berhenti, kita dapat menggunakan sintaks BREAK
7. Untuk melanjutkan struktur pengulangan ke awal pengulangan maka dapat digunakan sintaks CONTINUE
8. Hal yang terpenting dari struktur pengulangan adalah kondisi berhenti yang akan memberikan kondisi apakah pengulangan dilakukan atau tidak.

|  |  |
| --- | --- |
|  | **Soal Latihan** |

1. Buatlah sebuah program untuk menampilkan bilangan ganjil antara 1-20 dengan menggunakan struktur while
2. Buatlah sebuah program untuk menampilkan bilangan kelipatan 3, dengan range berupa inputan dari user. Gunakan struktur do…while

Contoh inputan :

|  |
| --- |
| Awal : 1  Akhir : 10  Kelipatan 3 : 3 6 9 |
| Awal : 15  Akhir : 30  Kelipatan 3 : 15 18 21 24 27 30 |

1. Buatlah program untuk menghitung nilai faktorial suatu bilangan yang diinputkan oleh user.

Rumus faktorial adalah sebagai berikut :

n! = n\*(n-1)\*(n-2)\*…\*(n-(n-1))

n merupakan inputan dari user.

Boleh menggunakan struktur pengulangan manapun.

Contoh inputan

|  |
| --- |
| Faktorial dari : 3  3! = 3\*2\*1=6 |
| Faktorial dari : 5  5! = 5\*4\*3\*2\*1 = 120 |

1. Modifikasi program faktorial diatas, sehingga jika user memasukkan angka <= 0 maka akan diulang kembali untuk memasukkan angka

Hint : gunakan continue

1. Buatlah sebuah program untuk menghitung gaji n karyawan, n merupakan sebuah bilangan bulat yang akan diinputkan oleh user, kemudian akan dihitung gaji perkaryawan dengan ketentuan sebagai berikut :

Gaji per jam : Rp 10.000,-

Bila jam kerja > 7 jam, maka sisa jam kerja dihitung sebagai lembur yg besarnya 1.5 \* gaji per jam

Kemudian ditampilkan total gaji karyawan yang harus dibayarkan oleh perusahaan.

Jumlah jam kerja tiap karyawan akan diinputkan oleh user

Contoh inputan

|  |
| --- |
| Jumlah karyawan : 3 [*inputan user*]  Jam kerja karyawan 1 : 8 [*inputan user*]  Total Gaji : 85000  Jam kerja karyawan 2 : 6 [*inputan user*]  Total Gaji : 60000  Jam kerja karyawan 3 : 12 [*inputan user*]  Total Gaji : 145000  Total Gaji karyawan : 290000 |

1. Buatlah program untuk menampilkan angka seperti berikut :

Contoh inputan :

|  |
| --- |
| Jumlah angka : 5  1 5  2 4  3  2 4  1 5 |
| Jumlah angka : 6  1 6  2 5  34  34  2 5  1 6 |
| Jumlah angka : 9  1 9  2 8  3 7  4 6  5  4 6  3 7  2 8  1 9 |

1. Buatlah program untuk menampilkan bintang berbentuk belah ketupat seperti berikut :

Contoh inputan :

|  |
| --- |
| Jumlah bintang : 5  \*  \* \*  \* \*  \* \*  \* |
| Jumlah bintang : 6  \*\*  \* \*  \* \*  \* \*  \* \*  \*\* |
| Jumlah bintang : 9  \*  \* \*  \* \*  \* \*  \* \*  \* \*  \* \*  \* \*  \* |

# Array dan Record

|  |  |  |
| --- | --- | --- |
|  | | **Overview** |
| Dalam bab ini akan dijelaskan bagaimana menerapkan tipe data *array* dan tipe data bentukan, dalam hal ini *record* dalam bahasa C. Tipe data *array* yang akan dijelaskan dalam bab ini mencakup tipe data *array* dimensi satu dan *array* dimensi dua. | | |
|  | | |
|  | **Tujuan** | |
| 1. Mengetahui dan memahami penggunaan *array*/larik serta record. 2. Mengetahui dan dapat mengimplementasikan cara mendeklarasikan tipe data *array* dan *record* dalam bahasa C. 3. Dapat membuat program sederhana yang melibatkan *array* satu dimensi dan *array* dua dimensi dalam bahasa C. 4. Dapat membuat program sederhana menggunakan *record* dalam bahasa C. | | |

## Array

*Array* sering disebut sebagai **larik**. *Array* merupakan koleksi data dengan setiap elemen data menggunakan nama yang sama dan masing-masing elemen data bertipe sama. Setiap komponen /elemen *array* dapat diakses dan dibedakan melalui indeks *array*.

*Array* dapat digolongkan menjadi:

1. *Array* berdimensi satu
2. *Array* berdimensi dua
3. *Array* berdimensi banyak

### *Array* Satu Dimensi

Bentuk umum deklarasi *array* dimensi satu dalam bahasa C:



dimana:

* tipe\_data untuk menyatakan jenis tipe data elemen *array* (misal char, int, long, dsb…)
* nama\_var menyatakan nama variabel *array*
* ukuran untuk menyatakan jumlah maksimal elemen *array*

Contoh deklarasi *array* dalam bahasa C.

float nilai[5];

deklarasi di atas menyatakan bahwa *array* nilai mempunyai 5 elemen bertipe float.

Dalam C, data *array* disimpan dalam memori pada lokasi alamat yang berurutan. Elemen pertama memiliki indeks 0 (nol). Jika variabel *array* nilai diatas memiliki 5 elemen, maka elemen pertama memiliki indeks 0, elemen kedua memiliki indeks 1, elemen ketiga memiliki indeks 2, dan seterusnya. Jadi elemen terakhir dalam *array* nilai memiliki indeks 4.

Bentuk umum pengaksesan elemen *array*:



Untuk *array* nilai,

nilai[0] 🡪 elemen ke-1 dari *array* nilai

nilai[4] 🡪 elemen ke-5 dari *array* nilai

Contoh memasukkan data ke *array* dalam C:

nilai[0] = 70; //menyimpan nilai 70 dalam elemen ke-1 dari array nilai (indeks 0)

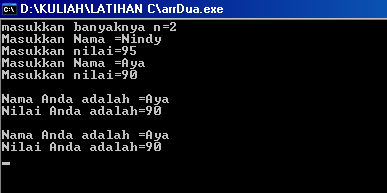
scanf(“%f”,&nilai[2]); /\*meminta masukan untuk elemen ke-3 array nilai

(indeks 2)\*/

Contoh program sederhana dalam C tanpa menggunakan *array*

|  |
| --- |
| 1. #include<stdio.h>  2. #include<conio.h>  3. main()  4. {  5. int i,n,nilai;  6. char nama[20];  7. printf("masukkan banyaknya n=");scanf("%d",&n);  8. for (i=1; i<=n; i++)  9. {  10. fflush(stdin); //menghapus isi buffer  11. printf("Masukkan Nama =");gets(nama);  12. printf("Masukkan nilai=");scanf("%d",&nilai);  13. }  14. for (i=1; i<=n; i++)  15. {  16. printf("\nNama Anda adalah =%s\n",nama);  17. printf("Nilai Anda adalah=%d\n",nilai);  18. }  19. getche(); //menahan tampilan pada layar  20.} |

Output yang dihasilkan:



Berikut ini adalah contoh program bahasa C dengan menggunakan *array*. Perhatikan **perbedaan output** yang terjadi dengan program sebelumnya.

|  |
| --- |
| 1. #include<stdio.h>  2. #include<conio.h>  3. main()  4. {  5. int i,n,nilai[20];  6. /\*mendefinisikan array nama dg panjang max. 20 char,dan memiliki jumlah  7. elemen array 20\*/  8. char nama[20][20];  9. printf("masukkan banyaknya n=");scanf("%d",&n);  10. for (i=1; i<=n; i++)  11. {  12. fflush(stdin); //menghapus isi buffer  13. printf("Masukkan Nama =");gets(nama[i]);  14. printf("Masukkan nilai=");scanf("%d",&nilai[i]);  15. }  16. for (i=1; i<=n; i++)  17. {  18. printf("\nNama Anda adalah =%s\n",nama[i]);  19. printf("Nilai Anda adalah=%d\n",nilai[i]);  20. }  21. getche(); //menahan tampilan pada layar  22.} |

Output yang dihasilkan:



### *Array* Dua Dimensi

*Array* dua dimensi merupakan *array* yang terdiri dari m buah baris dan n buah kolom. Bentuknya dapat berupa matriks atau tabel.

Bentuk umum *array* dua dimensi dalam bahasa C:



Contoh :

int X[3][2]; //mendefinisikan array 2 dimensi dengan 3 baris, 2 kolom

|  |  |
| --- | --- |
| X[0][0] | X[0][1] |
| X[1][0] | X[1][1] |
| X[2][0] | X[2][1] |

* **Cara mengakses *array*** :

Untuk mengakses *array*, misalnya kita ingin mengisi elemen *array* baris 1 kolom 2 dengan 10 maka perintahnya adalah sbb :

X[0][1] = 10;

* Untuk mengisi dan menampilkan isi elemen *array* ada dua cara yaitu :

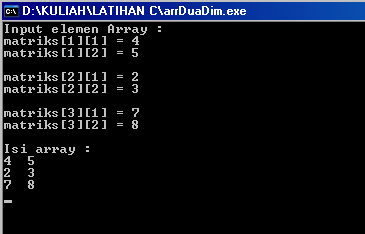
♦ *Row Major Order* (secara baris per baris)

♦ *Column Major Order* (secara kolom per kolom)

Contoh program C untuk mengisi dan menampilkan isi elemen *array* secara *row major order*.

|  |
| --- |
| 1./\* Program menginput nilai(bilangan) ke dalam array dimensi dua dan menampilkannya secara  2. Row major order \*/  3. #include <stdio.h>  4. #include <conio.h>  5. void main()  6. { int baris, kolom, matriks[3][2];  7. // Input elemen array secara Row Major Order  8. printf("Input elemen Array : \n");  9. for(baris=0; baris<3; baris++)  10. { for(kolom=0; kolom<2; kolom++)  11. { printf("matriks[%i][%i] = ", baris+1, kolom+1);  12. scanf("%i", &matriks[baris][kolom]);  13. } //end loop kolom  14. printf("\n");  15. } //end loop baris  15. // Tampilkan elemen Array secara Row Major Order  17. printf("Isi array : \n");  18. for(baris=0; baris<3; baris++)  19. { for(kolom=0; kolom<2; kolom++)  20. { printf("%i ", matriks[baris][kolom]);}//end loop klm  21. printf("\n");  22. } //end loop baris  23. getche();  24. } //end program |

Output yang dihasilkan:



Bagaimana dengan *Column Major Order* ? Coba Anda buat program C-nya!

## *Record*

*Record* merupakan kumpulan data yang tidak sejenis atau variasi. File merupakan berkas kumpulan dari beberapa *record* yang tersimpan dan mempunyai nama. Dalam bahasa C untuk penulisan *record* dikenal dengan nama **struct**.

Bentuk umum penulisan *record* dalam bahasa C.

|  |
| --- |
| **struct** nama\_type\_record { tipe\_data nm\_var;  :  } var\_record; |

Contoh :

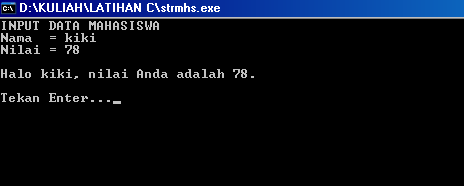
**struct** mahasiswa { char nama[20];

int nilai;

} nilaiMhs;

|  |
| --- |
| 1. #include <stdio.h>  2. #include <conio.h>  3. main()  4. { struct mahasiswa {char nama[20];  5. int nilai; } nilaiMhs;  6. printf("INPUT DATA MAHASISWA\n");  7. printf("Nama = "); scanf("%s",&nilaiMhs.nama);  9. printf("Nilai = "); scanf("%i",&nilaiMhs.nilai);  10. //menampilkan kembali data mahasiswa  11. printf("\nHalo %s, nilai Anda adalah  12. %i.", nilaiMhs.nama,nilaiMhs.nilai);  13. printf("\n\nTekan Enter...");  14. getche();  15. } |

### Output yang dihasilkan



### Penggunaan *Record* dalam *Array*

Dalam contoh program *record* sebelumnya, kita hanya dapat menyimpan satu data mahasiswa saja. Jika ingin menyimpan beberapa buah data mahasiswa maka dapat menggunakan array.

Contoh deklarasi penggunaan Record dalam Array di bahasa C.

**struct** mahasiswa { char nama[20];

int nilai;

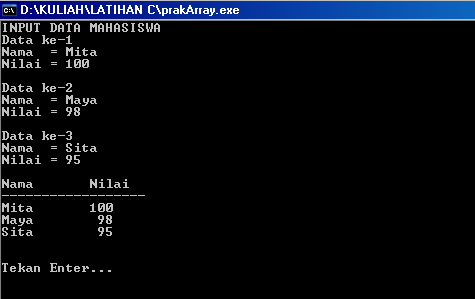
} nilaiMhs[3];

Deklarasi di atas menjelaskan bahwa terdapat *array record* nilaiMhs yang memiliki 3 elemen.

Berikut contoh program penggunaan *record* dalam *array*.

|  |
| --- |
| 1. #include <stdio.h>  2. #include <conio.h>  3. main()  4. { struct mahasiswa {char nama[20];  5. int nilai;  6. } nilaiMhs[3];  7. int i,a;  8. a=1;  9. printf("INPUT DATA MAHASISWA");  10. for(i=0;i<3;i++)  11. { printf("\nData ke-%i\n", a);  12. printf("Nama = "); scanf("%s",&nilaiMhs[i].nama);  13. printf("Nilai = "); scanf("%i",&nilaiMhs[i].nilai);  14. a++;  15. } //end loop i  16.  17. //menampilkan kembali data mahasiswa  18. printf("\nNama Nilai\n");  19. printf("------------------\n");  20. for(i=0;i<3;i++)  21. {  22. printf("%s%10i\n",nilaiMhs[i].nama,nilaiMhs[i].nilai);  23. } //end loop i  24. printf("\n\nTekan Enter...");  25. getche();  26. } |

Output yang dihasilkan:



### Penggunaan *Array* dalam *Record*

Seorang mahasiswa dapat saja memiliki lebih dari satu nilai. Dalam contoh program *record* sebelumnya, asumsi yang digunakan adalah mahasiswa hanya memiliki sebuah nilai saja. Apabila memang dibutuhkan untuk menyimpan beberapa buah nilai bagi seorang mahasiswa, kita dapat menerapkan tipe data *array* untuk field nilai. Berikut contoh deklarasinya dalam bahasa C:

**struct** mahasiswa { char nama[20];

int nilai[3];

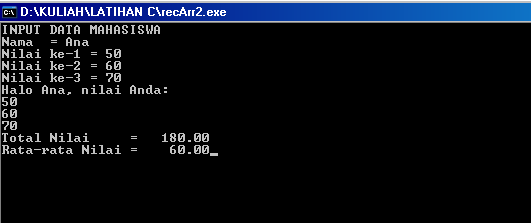
} nilaiMhs;

Deklarasi di atas menjelaskan bahwa terdapat *record* nilaiMhs, dimana *record* tersebut memiliki variabel *array* nilai yang terdiri dari 3 elemen.

Contoh program penggunaan *array* dalam *record*.

|  |
| --- |
| 1. #include <conio.h>  2. #include <stdio.h>  3.  4. main()  5. {  6. struct mahasiswa { char nama[20];  7. int nilai[3];  8. } nilaiMhs;  9. int i; int a = 0;  10. float total=0;  11. float rata2=0;;  12. printf("INPUT DATA MAHASISWA\n");  13. printf("Nama = "); scanf("%s",&nilaiMhs.nama);  14. a=1;  15. for(i=0;i<3;i++)  16. {  17. printf("Nilai ke-%i = ",a);  18. scanf("%i",&nilaiMhs.nilai[i]);  19. a++;  20. }  21. printf("Halo %s, nilai Anda: \n",nilaiMhs.nama);  22. for(i=0;i<3;i++)  23. {  24. printf("%d\n",nilaiMhs.nilai[i]);  25. total = total + nilaiMhs.nilai[i];  26. }  27. rata2 = total/3;  28. printf("Total Nilai = %8.2f",total);  29. printf("\nRata-rata Nilai = %8.2f",rata2);  30. getche();  31. } |

Output yang dihasilkan:



### *Array* *Record* yang mengandung *Array*

Pada sub bab-sub bab sebelumnya, telah dipelajari bagaimana menerapkan *record* dalam *array* dan sebaliknya. Pada kebanyakan kasus, seringkali kita harus menerapkan suatu *array record* yang mengandung *array*. Sebagai contoh, mahasiswa dapat memiliki lebih dari satu nilai. Tetapi jumlah mahasiswa yang ada lebih dari satu orang. Untuk kasus semacam ini , harus digunakan *array record* yang mengandung *array*. Berikut adalah contoh deklarasinya dalam bahasa C.

**struct** mahasiswa { char nama[20];

int nilai[3];

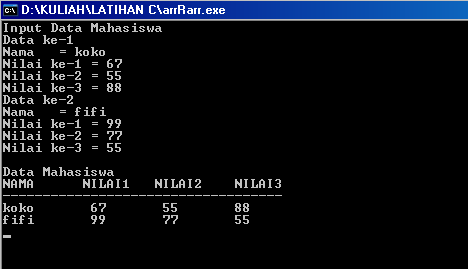
} nilaiMhs[2];

Deklarasi di atas menjelaskan bahwa terdapat *array record* nilaiMhs yang terdiri dari 2 elemen *array*, dimana masing-masing (dalam hal ini asumsikan mahasiswa) memiliki 3 buah nilai.

Contoh program penggunaan *array* *record* yang mengandung *array*.

|  |
| --- |
| 1.#include <stdio.h>  2.#include <conio.h>  3.  4.main()  5.{  6. struct mahasiswa { char nama[20];  7. int nilai[3]; } nilaiMhs[2];  8. int i,a,j,b;  9. a=1;  10. printf("Input Data Mahasiswa\n");  11. for(i=0;i<2;i++)  12. {  13. printf("Data ke-%i\n",a);  14. printf("Nama = "); scanf("%s",&nilaiMhs[i].nama);  15. b=1;  16. for(j=0;j<3;j++)  17. {  18. printf("Nilai ke-%i = ",b);  19. scanf("%i",&nilaiMhs[i].nilai[j]);  20. b++;  21. }  22. a++;  23. }  24.  25. printf("\nData Mahasiswa\n");  26. printf("NAMA NILAI1 NILAI2 NILAI3\n");  27. printf("-----------------------------------\n");  28. for(i=0;i<2;i++)  29. {  30. printf("%s",nilaiMhs[i].nama);  31. for(j=0;j<3;j++)  32. {  33. printf("%9i",nilaiMhs[i].nilai[j]);  34. }  35. printf("\n");  36. }  37. getche();  38.} |

Output yang dihasilkan:



|  |  |
| --- | --- |
|  | **Rangkuman** |

1. Array disebut juga dengan larik.
2. Bentuk umum deklarasi array satu dimensi dalam bahasa C:

type\_data nm\_var[ukuran];

1. Indeks elemen array dalam bahasa C, dimulai dengan indeks 0 (nol).
2. Bentuk umum deklarasi array dua dimensi dalam bahasa C:

type\_data nm\_var[baris][kolom];

1. Deklarasi record dalam bahasa C menggunakan sintaks **struct**.
2. Elemen-elemen data dalam *array* memiliki tipe data yang sama, sedangkan elemen-elemen dalam *record* dapat memiliki tipe data yang berbeda.

|  |  |
| --- | --- |
|  | **Latihan** |

1. Buat program dalam bahasa C untuk mencetak “\*” **sebanyak nilai array** yang diinputkan oleh user. Misal terdapat array A yang terdiri dari 3 elemen, dan diinputkan nilai-nilai sebagai berikut:

A[0] = 4

A[1] = 2

A[2] = 6

Maka output yang diinginkan adalah:

A[0] = \*\*\*\*

A[1] = \*\*

A[2] = \*\*\*\*\*\*

2. Buat program dalam bahasa C yang masukannya adalah harga-harga integer antara 0 s/d 100 yang disimpan dalam array integer ARRNILAI[0..9] dan melakukan proses sebagai berikut:

- menghitung nilai rata-rata dari harga yang disimpan dalam array tersebut

- mengkonversi setiap angka ke dalam skala 'A' s/d 'E' dengan kriteria:

>= 80 : 'A'

70 - 79 : 'B'

55 - 69 : 'C'

40 - 54 : 'D'

< 40 : 'E'

1. Diketahui record mahasiswa berisi field-field sbb :

type recmhs : < NIM : integer

Nama: string[20]

nUTS: integer

nUAS: integer

nA : real >

 dimana : NIM = Nomor Induk Mahasiswa

nUTS = nilai UTS

nUAS = nilai UAS

nA = nilai Akhir

Buatlah sebuah program dalam bahasa C untuk menentukan nilai akhir dari, asumsikanlah ada 5 mahasiswa. Adapun nilai akhir dihitung dengan rumus :

  Nilai Akhir = 40% Nilai UTS + 60% Nilai UAS

Output yang diinginkan adalah:

NIM Nama UTS UAS NA

100

:

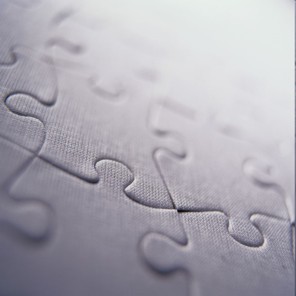
:

Rata-rata Nilai UTS : …

Rata-rata Nilai UAS : …

Rata-rata Nilai Akhir : …

# *Pemrograman Modular*



|  |  |
| --- | --- |
|  | **Overview** |
| Pemrograman modular memungkinkan perancang program menyederhanakan persoalan didalam program dengan memecah atau membagi persoalan tersebut menjadi sub-sub persoalan yang lebih kecil agar mudah diselesaikan. Secara umum dikenal dua cara yang dapat digunakan untuk memecah persoalan dalam modul-modul, yaitu dengan menggunakan struktur fungsi dan prosedur. Pemahaman tentang perbedaan dan karakteristik masing-masing struktur tersebut perlu diimbangi pula dengan kemampuan mengimplementasikannya dalam program. | |

|  |  |
| --- | --- |
|  | **Tujuan** |
| 1. Memahami dan dapat mempraktekkan konsep pemrograman modular 2. Dapat mengimplementasikan pemakaian fungsi dan procedure. 3. Dapat menentukan kapan harus menggunakan fungsi dan procedure 4. Dapat menerapkan pemanggilan subprogram dari program utama. | |

## *Variabel Lokal dan Variabel Global*

### Variabel Lokal

Variabel lokal adalah variabel yang dideklarasikan didalam fungsi atau procedur. Variabel ini hanya dapat dikenali didalam fungsi atau procedure itu sendiri. Variabel lokal tidak dapat dipanggil, diakses dan diubah oleh prosedur atau fungsi yang lain, bahkan oleh program utama sekalipun.

### Variabel Global

Variabel global adalah variabel yang didefinisikan dalam program utama dan dapat digunakan di program utama maupun sub-sub program lainnya. Nilai dari variabel ini dapat dipanggil, diakses dan diubah oleh prosedur atau fungsi apapun yang terdapat dalam program tersebut.

## *Fungsi*

Fungsi adalah subprogram yang menerima data masukan atau parameter inputan, kemudian melakukan beberapa proses, kemudian mengembalikan nilai hasil proses untuk digunakan oleh proses yang memanggil fungsi tersebut. Pendeklarasian fungsi merupakan salah satu cara memecah persoalan ke dalam beberapa sub persoalan yang lebih mudah diselesaikan.

Struktur umum sebuah fungsi :

Tipe\_data\_kembali nama\_fungsi(daftar\_parameter)

{

/\* instruksi dalam fungsi \*/

return value;

}

Jika kita melihat struktur penulisan fungsi, struktunya hampir atau bahkan sama dengan program utama. Pada dasarnya, pemprograman dengan menggunakan C++ adalah pemprograman dengan struktur fungsi, dimana setiap kode yang dituliskan harus dalam bentuk fungsi, tak terkecuali program utama. Program utama merupakan suatu fungsi dengan nama main() yang tidak memiliki nilai kembali atau nilai kembalinya adalah kosong atau 0. Oleh karenanya, kita juga dapat menuliskan program utama dengan void main() atau dengan int main(), dengan *return value*-nya 0.

Saat program pertama dijalankan, kode yang pertama dieksekusi adalah fungsi main(). Oleh karenanya, setiap program minimal harus memiliki satu fungsi yaitu main(), dimana isi dari fungsi ini adalah inti dari program.

Perhatikan contoh program untuk menghitung luas persegi berikut ini :

1. #include <stdio.h>
2. #include <conio.h>
3. void main() {
4. int p,l; // Variabel lokal
5. int luas; // Variabel lokal
6. printf(“Masukan Lebar : ”);
7. scanf(“%i”, &p);
8. printf(“Masukan Tinggi : ”);
9. scanf(“%i”, &l);
10. Luas = p \* l;
11. printf(“Luas Persegi : %i”, luas);
12. getch();
13. }

Perhatikan program menghitung luas tersebut diatas. Pada program tersebut, untuk menghitung luas persegi panjang dapat kita jadikan fungsi seperti berikut :

1. #include <stdio.h>
2. #include <conio.h>
3. int hitungLuas(int p,int l) {
4. int luas; // Variabel lokal
5. luas = p \* l;
6. return luas;
7. }
8. void main() {
9. int pj,lb;
10. int luasUtama;
11. printf(“Masukan Lebar : ”);
12. scanf(“%i”, &p);
13. printf(“Masukan Tinggi : ”);
14. scanf(“%i”, &l);
15. luasUtama = hitungLuas(pj, lb);
16. printf(“Luas Persegi : %i”, luasUtama);
17. getch();
18. }

Pada program di atas, tampak bahwa code dengan menggunakan fungsi lebih banyak dibanding yang tidak, namun jika kita menulis dengan fungsi, akan memprmudah kita membagi beberapa masalah. Selain itu, jika kita ingin menggunakan lagi perhitungan tersebut, kita hanya memanggil fungsi tersebut dengan parameter yang sesuai. Namun jika kita tidak menggunakan fungsi, maka codenya harus kita tulis ulang. Untuk memanggil fungsi, harus diisikan pada suatu nilai tertentu yang tipenya sesuai denga tipe nilai kembali fungsi tersebut. Misalnya pada contoh diatas, variabel luasUtama yang akan diisikan oleh nilai fungsi tersebut.

Contoh yang lain untuk penggunaan fungsi adalah penentuan index nilai mahasiswa dimana inputannya adalah nilai dengan skala 0 – 100 dan menghasilkan index A, B, C, D dan E. Seperti pada contoh berikut :

1. #include <stdio.h>
2. #include <conio.h>
3. char getIndex(int n) {
4. char inx; // Variabel lokal
5. if (n >= 75) inx = “A”;
6. else if (n >= 65) inx = “B”;
7. else if (n >= 55) inx = “C”;
8. else if (n >= 45) inx = “D”;
9. else inx = “E”;
10. return inx;
11. }
12. void main() {
13. int nilai;
14. printf(“Masukan Nilai Mahasiswa : ”);
15. scanf(“%i”, &nilai);
16. printf(“Index Nilai Mahasiswa adalah %i”, getIndex(nilai));
17. getch();
18. }

## *Prosedur*

Prosedur adalah sederetan instruksi yang diberi nama, dan melakukan tujuan tertentu. Seperti halnya pada fungsi, prosedur bekerja dengan mekanisme pemanggilan-pengembalian (*call-return mechanism*).

Struktur umum deklarasi prosedur adalah sebagai berikut:

1. void nama\_procedure (<daftar\_parameter\_input> <,daftar\_parameter\_output>)
2. {
3. /\* instruksi \*/
4. }

Untuk penulisan parameter input-ouput dengan menambahkan karakter “&” pada tipe datanya. Misalnya int& n.

Kembali pada contoh sebelumnya (menghitung luas persegi panjang), kita juga dapat mengubahnya menjadi procedure seperti pada contoh dibawah ini :

1. #include <stdio.h>
2. #include <conio.h>
3. void hitungLuas(int p, int l, int& luas) {
4. luas = p \* l;
5. return 0;
6. }
7. void main() {
8. int pj,lb; // Variabel lokal
9. int luasUtama; // Variabel lokal
10. printf(“Masukan Lebar : ”);
11. scanf(“%i”, &pj);
12. printf(“Masukan Tinggi : ”);
13. scanf(“%i”, &lb);
14. hitungLuas(pj, lb, luasUtama);
15. printf(“Luas Persegi : %i”, luasUtama);
16. getch();
17. }

Pada dasarnya code diatas tidak jauh berbeda dengan menggunakan fungsi. Pada procedure, nilai variabel luasUtama dimasukan sebagai parameter, yang nantinya nilai variabel tersebut akan dipetakan dengan variabel luas pada fungsi, dimana variabel tersebut merupakan parameter input-output. Sehingga setelah procedure dijalankan, maka nilai variabel luasUtama akan diisi oleh nilai variabel luas pada procedure. Untuk pemanggilan procedure berbeda dengan fungsi yang harus menyediakan variabel untuk menyimpan nilai kembali fungsi tersebut, pada procedure kita tidak perlu menyediakan variabel untuk menampung nilai kembaliannya, karena procedure tidak memiliki nilai kembali. Sedangkan untuk parameter input-output tidak harus ada. Jika kita mengosongkannya parameter input-output juga tidak apa-apa.

Selanjutnya kita akam mengubah kode pada contoh fungsi untuk menghitung index nilai mahasiswa diatas menjadi struktur procedure seperti berikut :

1. #include <stdio.h>
2. #include <conio.h>
3. char inx; // Variabel global
4. char getIndex(int n) {
5. if (n >= 75) inx = “A”;
6. else if (n >= 65) inx = “B”;
7. else if (n >= 55) inx = “C”;
8. else if (n >= 45) inx = “D”;
9. else inx = “E”;
10. }
11. void main() {
12. int nilai;
13. printf(“Masukan Nilai Mahasiswa : ”);
14. scanf(“%i”, &nilai);
15. getIndex(nilai)
16. printf(“Index Nilai Mahasiswa adalah %c”, inx);
17. getch();
18. }

## *Fungsi Rekursif*

Fungsi dapat dipanggil oleh program utama dan juga oleh fungsi yang lain, selain kedua metode pemanggilan tersebut, fungsi dapat juga dipanggil oleh dirinya sendiri. Yang dimaksud disini adalah pemanggilan fungsi itu didalam fungsi itu sendiri, bukan pada fungsi yang lain. Fungsi yang melakukan pemanggilan terhadap dirinya sendiri disebut dengan fungsi rekursif.

Berikut adalah contoh program dalam program yang menerapkan metode rekursif untuk menghitung nilai jumlah deret bilangan dari suatu bilangan.

Struktur umum deklarasi prosedur adalah sebagai berikut:

1. // Program Hitung\_Deret
2. #include <stdio.h>
3. int angka; // variabel Global
4. int hasil; // variabel Global
5. {===Program Utama===}
6. void main () { // program Utama
7. printf(“Masukan Angka Batas Deret :”);
8. scanf(“%i”,&angka);
9. hasil = hitungDeret(angka);
10. printf(“Jumlah deret : %i.”, hasil);
11. }
12. int hitungDeret(int bil)
13. {
14. if bil = 0 then
15. return 0;
16. else
17. return (bil + hitungDeret**(bil - 1)**);
18. }

Perhatikan pada baris ke-20 kode diatas, pemanggilan fungsi hitung\_deret tersebut berada pada bloknya sendiri degan parameter yang diubah. Hal yang harus diperhatikan dalam pembuatan rekursif adalah fungsi tersebut harus berhenti dimana didalam fungsi tersebut harus ada pengkondisian bahwa fungsi harus berhenti. Pada contoh diatas, code untuk menghentikan fungsi tersebut ada pada baris ke 17 dan 18, dimana apabila inputan parameternya 0, maka akan menghasilkan 0. Namun jika tidak, proses akan memanggil fungsi yaitu dirinya sendiri.

Sebagai ilustrasi program diatas, mari kita coba memanggil fungsi faktorial dengan hitungDeret(4), maka proses yang akan dilakukan adalah :

1. 4 + hitungDeret(3)
2. 4 + 3 + hitungDeret(2)
3. 4 + 3 + 2 + hitungDeret(1)
4. 4 + 3 + 2 + 1 + hitungDeret(0)
5. 4 + 3 + 2 + 1 + 0
6. 4 + 3 + 2 + 1
7. 4 + 3 + 3
8. 4 + 6
9. 10
10. Hasil Akhir : 10

## *Unit*

Contoh sederhana adalah kita akan membuat unit pada contoh fungsi diatas. Buatlah file dengan nama unit1.h. dengan kode sebagai berikut :

1. #include <stdio.h>
2. #include <conio.h>
3. #define Phi 3.14
4. void luasPersegi(int p, int l, int& luas) {
5. luas = p \* l;
6. return 0;
7. }
8. float luasLingkaran(int r) {
9. float ls;
10. ls = Phi \* r \* r;
11. return ls;
12. }

Selanjutnya buat program utama dengan nama prog-7.cpp. dengan kode sebagai berikut :

1. #include <unit1.h>
2. void main() {
3. int pj,lb,jari; // Variabel lokal
4. int luasP; // Variabel lokal
5. float luasL;
6. printf(“Menghitung Luas Persegi\n”);
7. printf(“Masukan Lebar : ”);
8. scanf(“%i”, &pj);
9. printf(“Masukan Tinggi : ”);
10. scanf(“%i”, &lb);
11. hitungLuas(pj, lb, luasP);
12. printf(“Luas Persegi : %i”, luasP);
13. printf(“Menghitung Luas Lingkaran\n”);
14. printf(“Masukan Jari-jari Lingkaran : ”);
15. scanf(“%i”, &jari);
16. luasL = luasLingkaran(jari);
17. printf(“Luas Lingkaran : %f”, luasL);
18. getch();
19. }

|  |  |
| --- | --- |
|  | **Latihan** |

1. Buatlah program yang menggunakan fungsi untuk perhitungan aritmetik yaitu tambah, kurang, kali dan bagi.
2. Buatlah program yang dapat menentukan nilai dari deret fibonansi. Deret fibonansi adalah sebuah deret dimana bilangan berikutnya adalah bilangan merupakan penjumlahan dari dua buah bilangan sebelumnya. Contoh deret fibonansi adalah :

1 1 2 3 5 8 12 20 32 ...

1. Buatlah sebuah fungsi biasa yang dapat mengkonfersi angka bilangan bulat menjadi huruf dengan ucapan.

Misalnya : 10 : Sepuluh

12 : Duabelas

1234 : seribu duaratus tiga puluh empat

1. Buatlah soal nomor 3 diatas kedalam fungsi rekursif.
2. Buatlah fungsi dan procedure untuk menukarkan dua buah bilangan
3. Buatlah proram dengan menggunakan beberapa fungsi untuk mengubah derajat suhu dari celsius ke Fahrenhit, Reamur dan Kelfin.
4. Buatlah fungsi untuk memeriksa apakah bilangan tersebut merupakan bilangan prima atau bukan.

Buatlah fungsi untuk menghitung nilai mata uang rupiah dari dolar, yen, ringgit dan uero dengan nilai tukar ditentukan sendiri sesuai kondisi saat ini.>

# Mesin Karakter



|  |  |  |
| --- | --- | --- |
|  | | **Overview** |
| Dalam menyelesaikan masalah yang relatif kompleks perlu dilakukan penggambaran atau analogi dengan suatu hal yang bersifat nyata. Tujuan penggambaran ini agar mempermudah menentukan kebutuhan proses atau mekanisme tertentu, sehingga pada akhirnya seluruh mekanisme dapat teridentifikasi untuk menyelesaikan masalah. | | |
|  | **Tujuan** | |
| 1. Memahami perlunya analogi dalam mendefinisikan suatu permasalahan 2. Mempelajari mekanisme mesin abstrak 3. Mempelajari mekanisme mesin pencacah (mesin integer) 4. Mempelajari mekanisme mesin karakter 5. Mempelajari penggunaan mesin integer dan mesin karakter dalam menyelesaikan suatu kasus | | |

## *Penyiapan Mesin*

Dalam mesin karakter, pita karakter berupa file teks atau dapat digunakan tipe string. Yang penting dalam kasus ini, bagaimana menggunakan mesin karakter untuk mengakses pita karakter.

Berikut contoh mesin karakter yang disimpan pada mesinkar.inc

|  |
| --- |
| Nama file: mesinkar.inc |
| char pita[]="HARI INI HARI YANG SANGAT INDAH.";  int ci,ch;  char cc;  bool EOP;  void RESET() {  ci=0;  }  void INC() {  ci=ci+1;  }  void START() {  ch=0;  cc=pita[ch];  if (cc=='.')  EOP=true;  else {  EOP=false;  }  }  void ADV() {  ch=ch+1;  cc=pita[ch];  if (cc=='.') {  EOP=true;  }  } |

## *Penggunaan Mesin*

### 1. Menghitung Jumlah Karakter

Berikut ini adalah algoritma proses penghitungan huruf:

|  |
| --- |
| Program Hitung\_Huruf  {Menghitung banyaknya huruf dalam pita karakter}  Kamus data  CI : integer, CC : char, EOP : boolean  BEGIN  START  RESET  WHILE (not EOP) DO  INC  ADV  ENDWHILE  OUTPUT(CI)  END |

|  |
| --- |
| #include <stdio.h>  #include <conio.h>  #include "mesinkar.inc"  //Program Hitung Huruf  //Menghitung banyaknya huruf dalam pita karakter  void main() {  START();  RESET();  while ( !EOP ) {  INC();  ADV();  }  printf("Banyak huruf dalam pita = %d \n",ci);  getche();  } |

### 2. Menghitung Jumlah Karakter Tertentu

Kombinasi penggunaan mesin karakter dan mesin integer dapat juga dimanfaatkan untuk menghitung jumlah karakter tertentu, misalnya untuk menghitung jumlah huruf ’A’ dalam suatu pita karakter.

Cara menangani kasus ini yaitu dengan mengkombinasikannya dengan operasi if..then..else. Jika CC menunjukkan huruf ’A’, maka tombol INC pada mesin integer akan ditekan, jika bukan huruf ’A’, maka akan memajukan pita karakter menggunakan tombol ADV.

Berikut adalah algoritma dan program untuk menghitung banyaknya huruf ’A’:

|  |
| --- |
| **Algoritma** |
| Program Hitung\_Huruf\_A  {Menghitung banyaknya huruf A yang terdapat dalam pita karakter}  Kamus data  CI : integer, CC : char, EOP : boolean  BEGIN  RESET  START  WHILE (CC≠”.”) DO  IF CC = “A” THEN  INC  ENDIF  ADV  ENDWHILE  OUTPUT(CI)  END |
| **Bahasa C** |
| #include <conio.h>  #include <stdio.h>  #include "mesinkar.inc"  void main() {  START();  RESET();  while (cc!='.') {  if (cc=='A') {  INC();  }  ADV();  }  printf("Banyak huruf A = %d \n",ci);  getche();  } |

### 3. Menghitung Jumlah Kata

Sekilas terbayang menghitung jumlah kata adalah sesuatu yang amat sulit, padahal jumlah kata tidak lain adalah jumlah spasi ditambah 1.

Contoh isi pita:

|  |  |
| --- | --- |
| HARI INI HARI SENIN | 3 spasi = 4 kata |
| HARI INI HUJAN TURUN LAGI | 4 spasi = 5 kata |

Namun persoalan akan makin rumit bila dimungkinkan antar kata lebih dari 1 spasi. Untuk ini dapat dibuat prosedur mengabaikan spasi yang berlebihan.

Contoh isi pita karakter

” HARI INI HUJAN TURUN LAGI .”

Algoritma berikut prosedurnya sbb:

|  |
| --- |
| **Algoritma** |
| PROCEDURE IGNORE\_BLANK  {mengabaikan/membuang spasi berlebihan}  BEGIN  while (cc==' ') do  ADV  endwhile  ENDPROCEDURE  Algoritma Hitung\_Kata  {Menghitung banyaknya kata dalam pita karakter}  Kamus data  CI : integer, CC : char, EOP : boolean  BEGIN  RESET  START  WHILE (not EOP) DO  IF CC=' ' THEN  INC  IGNORE\_BLANK  ELSE  ADV  ENDIF  ENDWHILE  INC  OUTPUT(CI)  END |
| **Bahasa C** |
| void IGNORE\_BLANK() {  while (cc==' ') {  ADV();  }  }  int main() {  START();  RESET();  IGNORE\_BLANK();  while (cc!='.') {  if (cc==' ') {  INC();  IGNORE\_BLANK();  }  else {  ADV();  }  }  /\* banyak kata = banyak spasi + 1 \*/  INC();  printf("Banyak kata = %d \n",ci);  getche();  } |

### 4. Studi Kasus Palindrom

Palindrom adalah istilah yang digunakan untuk kata atau kalimat yang apabila dibaca dari depan ke belakang atau sebaliknya, memiliki arti yang sama.

Contoh palindrom:

KATAK

KASUR RUSAK

KASUR NABABAN RUSAK

Untuk memeriksa apakah kata yang dimasukkan merupakan palindrom maka, dapat dibuat sebuah function yang memiliki tipe data boolean. Function ini akan mengembalikan nilai TRUE jika kalimat termasuk palindrom, dan akan mengembalikan nilai FALSE untuk kondisi sebaliknya.

|  |
| --- |
| **Algoritma** |
| FUNCTION IsPalindrom (kt : string) 🡪 boolean  {akan mengembalikan nilai TRUE jika k adalah palindrom}  Kamus data  i,j : integer  temporer : string  BEGIN  {mengisi temporer dengan karaker kosong}  temporer 🡨 ””  {mengisi j dengan lebar kata}  j 🡨 length(kt)  WHILE (j>0) DO  {operasi konkatenasi}  temporer 🡨 temporer + kt[j]  j 🡨 j – 1  ENDWHILE  {membandingkan isi temporer dengan kt}  IF temporer = kt THEN  return TRUE  ELSE  return FALSE  ENDIF  ENDFUNCTION |

Karena adanya perbedaan akses terhadap karakter dan string pada bahasa C, untuk mengimplementasi kasus ini akan lebih mudah bila mesin karakter sedikit dimodifikasi (dengan cc bertipe string) sebagai berikut :

|  |
| --- |
| **Modifikasi Mesin Karakter** |
| char cc[2];  bool EOP;  void START() {  ci=0;  cc[1]=0;  \*cc=pita[ci];  if (\*cc=='.')  EOP=true;  else {  EOP=false;  }  }  void ADV() {  ci++;  \*cc=pita[ci];  if (\*cc=='.') {  EOP=true;  }  } |

Misalkan program diawali dengan menyalin isi pita karakter ke dalam sebuah variabel string, kemudian memeriksa isi variabel tersebut apakah palindrom atau tidak.

|  |
| --- |
| **Bahasa C** |
| void COPY(char \*st) {  START();  strcpy(st,"");  while (!EOP) {  strcat(st,cc);  ADV();  }  }  //fungsi cek palindrom  bool IsPalindrom(char kt[]){  char ss[]={0,0};  int i,j;  char temp[30];  strcpy(temp,"");  j=strlen(kt)-1;  while (j>=0) {  \*ss=kt[j];  strcat(temp,ss);  j=j-1;  }  if (strcmp(temp,kt)==0)  return true;  else {  return false;  }  }  int main() {  char st[30];  COPY(st);  if (IsPalindrom(st))  printf("String pada pita Palindrom\n");  else {  printf("String pada pita Tidak Palindrom\n");  }  getche();  } |

**8.3. Latihan Tambahan**

1. Buatlah program untuk menghitung frekuensi huruf ”A” yang terdapat pada pita karakter yang berisi :

”ada apa dengan cinta. ” outputnya: 6/20.

(petunjuk: gunakan variabel bantu lain bila diperlukan)

1. Buatlah program untuk menentukan frekuensi huruf hidup (vokal) mana yang paling banyak muncul :

”hari ini hari libur. ” outputnya: vokal terbanyak ”i”.

(petunjuk: hitung frekuensi tiap vokal & pilih terbanyaknya)

1. Buatlah program untuk menentukan panjang dari kata terpanjang:

”hari ini hari libur. ” outputnya: kata terpanjang 5 karakter.

(petunjuk: setiap awal kata mulai dengan hitungan 1)

1. Buatlah program untuk menentukan panjang dari kata terpendek:

”hari ini hari libur. ” outputnya: kata terpendek 3 karakter.

1. Buatlah program untuk menentukan panjang rata-rata tiap kata:

”hari ini hari libur. ” outputnya: panjang kata rata-rata 4 karakter.

1. Buatlah program untuk menampilkan kata pertama:

”hari ini hari libur. ” outputnya: kata pertama ”hari”.

1. Buatlah program untuk menampilkan kata terakhir:

”hari ini hari libur. ” outputnya: kata terpanjang ”libur”.

1. Buatlah program untuk menentukan kata terpanjangnya, bila ada dua kata yang panjangnya sama maka yang dipilih adalah kata yang lebih dulu:

”hari ini hari libur. ” outputnya: kata terpanjang ”libur”.

1. Buatlah program untuk membalik posisi kata:

”hari ini hari libur. ” outputnya: kata terpanjang ”libur hari ini hari”.

1. Buatlah prosedur untuk melakukan invers kata atau kalimat. Invers adalah kebalikan, artinya fungsi akan menghasilkan kata atau kalimat secara terbalik dari semula.

Contoh:

”POLITEKNIK” setelah diinvers akan menjadi ”KINKETILOP”.

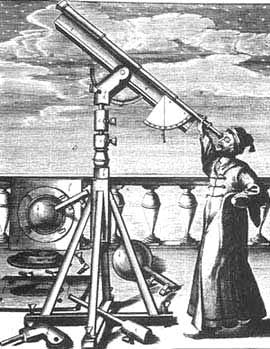
1. Buatlah fungsi baru, yang berbeda dengan contoh, untuk memeriksa sebuah kata atau kalimat termasuk kategori palindrom atau bukan! (petunjuk: tidak menggunakan temporer dan operasi konkatenasi, tetapi membandingkan huruf pertama dengan terakhir, lalu kedua dengan kedua dari akhir dst., jika ada yang beda berarti bukan palindrom)
2. Buatlah program yang menerima masukan berupa dua buah kata, kemudian memeriksa apakah kedua kata tadi termasuk anagram atau tidak (anagram artinya huruf-hurufnya sama tetapi diacak).

Contoh:

”SEBAB” dan ”BEBAS” 🡪 anagram

## ”KAPAS” dan ”PASAK” 🡪 anagram

# Pencarian



|  |  |
| --- | --- |
|  | **Overview** |
| Pencarian merupakan sebuah algoritma dasar yang sering diperlukan dalam pembuatan program. Berbagai algoritma pencarian telah diciptakan dan dapat digunakan. Pemahaman tentang beberapa algoritma pencarian dasar perlu diketahui, termasuk cara penggunaannya dalam program. Di dalam bab ini akan dijelaskan penggunaan program pencarian dalam array. | |

|  |  |
| --- | --- |
|  | **Tujuan** |
| 1. Mengenal beberapa program pencarian 2. Menerapkan program pencarian untuk menyelesaikan suatu masalah | |



## Konsep Pencarian

Pencarian adalah proses menemukan nilai (data) tertentu dari dalam sekumpulan nilai yang bertipe sama (tipe dasar maupun tipe bentukan). Dengan kata lain, algoritma pencarian adalah algoritma yang mengambil input berupa persoalan dan mengembalikan penyelesaian berupa penemuan nilai yang dicari dalam persoalan inputan.

Proses pencarian seringkali diperlukan pada saat program perlu mengubah atau menghapus nilai tertentu (sebelum bisa mengubah atau menghapus, perlu mencari dulu apakah nilai tersebut ada dalam kumpulan nilai tersebut). Kasus lain yang memerlukan algoritma pencarian adalah penyisipan data ke dalam kumpulan data (perlu dimulai dengan pencarian apakah data tersebut telah ada sehingga terhindar dari duplikasi data).

## *Pencarian Sekuensial*

Pencarian sekuensial (*sequential search*) adalah proses membandingkan setiap elemen larik (array) satu persatu dengan nilai yang dicari secara beruntun, mulai dari elemen pertama sampai elemen yang dicari sudah ditemukan, atau sampai seluruh elemen sudah diperiksa.

Pencarian sekuensial ini cocok untuk pencarian nilai tertentu pada sekumpulan data terurut maupun tidak. Keunggulan pencarian ini adalah dalam mencari sebuah nilai dari sekumpulan kecil data. Pencarian sekuensial termasuk pencarian yang sederhana dan cepat karena tidak memerlukan proses persiapan data (misalnya: pengurutan).

Untuk penerapan dalam program, pencarian sekuensial akan dibuat dalam suatu prosedur yang akan dipanggil dari main atau dari prosedur yang lainnya. Prosedur pencarian akan digunakan untuk mencari data pada suatu array, dimana di dalam array terdapat sekumpulan data angka. Berikut adalah penerapan pencarian sekuensial dalam bahasa pemrograman C :

1. Pencarian Sekuensial yang mengembalikan Boolean (ketemu atau tidak ketemu)

|  |
| --- |
| 1. #include <stdio.h> 2. #define TRUE 1 3. #define FALSE 0 4. typedef int bool; 5. void seqSearch(int L[10],int N,int X,bool \*ketemu); 6. void main() 7. { 8. int arr[10]= {6,7,3,8,2,5,4,1,8,10}; 9. bool pos; 10. seqSearch(arr,10,11,&pos); 11. if (pos) 12. printf("Ketemu"); 13. else 14. printf("Tidak Ketemu"); 15. } 16. void seqSearch(int L[10],int N,int X,bool \*ketemu) 17. { 18. int k; 19. k=0; 20. while ((k<N) && (L[k] != X)) 21. { 22. k = k+1; 23. } 24. if ((L[k] == X) && (k<10)) 25. \*ketemu=TRUE; 26. else 27. \*ketemu=FALSE; 28. } |

Pada contoh diatas, telah diinisialisasikan sebuah array dengan 10 elemen, kemudian dilakukan pencarian angka 8. Program akan menampilkan “Ketemu” jika angka yang dicari terdapat di dalam array dan menampilkan “Tidak Ketemu” jika angka yang dicari tidak terdapat di dalam array.

1. Pencarian yang mengembalikan posisi angka yang dicari

|  |
| --- |
| 1. #include <stdio.h> 2. void seqSearch(int L[10],int N,int X, int \*idx); 3. void main() 4. { 5. int pos, arr[10]= {6,7,3,8,2,5,4,1,8,10}; 6. seqSearch(arr,10,5,&pos); 7. if (pos!=-1) 8. printf("Ketemu di posisi %d",pos); 9. else 10. printf("Tidak Ketemu"); 11. } 12. void seqSearch(int L[10],int N,int X, int \*idx) 13. { 14. int k; 15. k=0; 16. while ((k<N) && (L[k] != X)) 17. { 18. k = k+1; 19. } 20. if ((L[k] == X) && (k<N)) 21. \*idx=k+1; 22. else 23. \*idx=-1; 24. } |

Program diatas akan mengembalikan posisi data yang dicari. Pencarian dimulai dari elemen pertama (k 🡨 0). Kemudian pencarian dilakukan dengan perulangan yang membandingkan setiap elemen larik secara berurutan dari elemen pertama hingga terakhir dengan nilai data yang diinginkan. Perulangan berakhir jika elemen yang dibandingkan bernilai sama dengan data yang dicari (mengembalikan posisi data), atau jika elemen larik telah dibandingkan semua namun data yang dicari tidak ditemukan (mengembalikan nilai -1).

1. Pencarian data berdasar inputan user

|  |
| --- |
| 1. #include <stdio.h> 2. void seqSearch(int L[10],int N,int X,int \*idx); 3. void inputData(int arr[]); 4. void main() 5. { 6. int i,pos,cari,arr[10]; 7. //panggil function inputData 8. inputData(arr); 9. //user input elemen yang dicari 10. printf("\nElemen yang dicari : "); 11. scanf("%d",&cari); 12. //menjalankan pencarian 13. seqSearch(arr,10,cari,&pos); 14. if (pos!=-1) 15. printf("Ketemu di posisi %d",pos); 16. else 17. printf("Tidak Ketemu"); 18. } 19. void inputData(int arr[]) 20. { 21. int i; 22. printf("\nMassukkan Elemen Array\n"); 23. //user input elemen array 24. for(i=0;i<10;i++) 25. { 26. printf("Array ke %d : ",i+1); 27. scanf("%d",&arr[i]); 28. } 29. } 30. void seqSearch(int L[10],int N,int X,int \*idx) 31. { 32. int k; 33. k=0; 34. while ((k<N) && (L[k] != X)) 35. { 36. k = k+1; 37. } 38. if ((L[k] == X) && (k<10)) 39. \*idx=k+1; 40. else 41. \*idx=-1; 42. } |

Pada program diatas, akan dilakukan pencarian data yang telah diinputkan oleh user. Pertama user harus memasukkan data ke dalam larik, hal ini ditangani di dalam prosedur inputData. Kemudian user akan diminta untuk memasukkan data yang ingin dicari. Prosedur seqSearch akan dijalankan dengan parameter berupa array dan nilai yang dicari yang telah dimasukkan oleh user. Prosedur ini akan mengembalikan posisi data yang dicari jika ketemu.

1. Implementasi pencarian ke data bentukan

|  |
| --- |
| 1. #include <stdio.h> 2. struct dataMahasiswa 3. { 4. char nim[10]; 5. char nama[20]; 6. }; 7. void inputDataMhs(struct dataMahasiswa mhs[]); 8. void seqSearch(struct dataMahasiswa mhs[],int N,char X[10],int \*idx); 9. void main() 10. { 11. int pos; 12. char pil; 13. char nimCari[10]; 14. struct dataMahasiswa mhs[5]; 15. //panggil function input data 16. inputDataMhs(mhs); 17. while(1) 18. { 19. //user input elemen yang dicari 20. printf("\nNim yang dicari : "); 21. scanf("%s",&nimCari); 22. //menjalankan pencarian 23. seqSearch(mhs,5,nimCari,&pos); 24. if (pos!=-1) 25. { 26. printf("\nData Mahasiwa Ketemu"); 27. printf("\nNim siswa : %s",mhs[pos].nim); 28. printf("\nNama siswa : %s",mhs[pos].nama); 29. } 30. else 31. printf("\nData Mahasiwa Tidak Ketemu"); 32. printf("\nCari data lagi [y/n] ? "); 33. scanf("%s",&pil); 34. if ((pil=='y')||(pil=='Y')) 35. continue; 36. else 37. break; 38. } 39. } 40. void inputDataMhs(struct dataMahasiswa mhs[]) 41. { 42. int i; 43. printf("\nMasukkan Data Mahasiswa\n"); 44. //user input elemen mahasiswa 45. for(i=0;i<5;i++) 46. { 47. printf("\nData Mahasiswa ke %d : \n",i+1); 48. printf("Nim : "); 49. scanf("%s",&mhs[i].nim); 50. printf("Nama : "); 51. scanf("%s",&mhs[i].nama); 52. } 53. } 54. void seqSearch(struct dataMahasiswa L[],int N,char X[10],int \*idx) 55. { 56. int k; 57. int i; 58. k=0; 59. while ((k<N) && (strcmp(L[k].nim,X)!=0)) 60. { 61. k = k+1; 62. } 63. if ((strcmp(L[k].nim,X)==0) && (k<N)) 64. \*idx=k; 65. else 66. \*idx=-1; 67. } |

Pada program diatas, akan dilakukan pencarian terhadap data bentukan. Data bentukan adalah dataMahasiswa dimana terdiri dari nim dan nama. Data mahasiswa tersebut akan diinputkan oleh user. Kemudian user juga diminta untuk memasukkan data yang akan dicari yaitu data nim. Jika nim mahasiswa ditemukan maka akan ditampilkan nama dan nim mahasiswa tersebut, jika tidak ada maka akan ditampilkan bahwa data mahasiswa tidak ditemukan.

Berikut adalah hasil output program pada contoh no 4 :

|  |
| --- |
| Masukkan Data Mahasiswa  Data Mahasiswa ke 1 :  Nim : 101 {inputan user}  Nama : Michael {inputan user}  Data Mahasiswa ke 2 :  Nim : 102 {inputan user}  Nama : Linda {inputan user}  Data Mahasiswa ke 3 :  Nim : 103 {inputan user}  Nama : Leonardo {inputan user}  Data Mahasiswa ke 4 :  Nim : 104 {inputan user}  Nama : Jason {inputan user}  Data Mahasiswa ke 5 :  Nim : 105 {inputan user}  Nama : Jennifer {inputan user}  Nim yang dicari : 103 {inputan user}  Data Mahasiwa Ketemu  Nim siswa : 103  Nama siswa : Leonardo  Cari data lagi [y/n] ? y {inputan user}  Nim yang dicari : 106 {inputan user}  Data Mahasiwa Tidak Ketemu  Cari data lagi [y/n] ? y {inputan user}  Nim yang dicari : 105 {inputan user}  Data Mahasiwa Ketemu  Nim siswa : 105  Nama siswa : Jennifer  Cari data lagi [y/n] ? n {inputan user} |

Pencarian sequential tidak efektif jika digunakan pada data yang banyak atau data yang dicari berada pada posisi terakhir pencarian.

## *Pencarian Biner*

Pencarian biner adalah proses mencari data dengan membagi data atas dua bagian secara terus menerus sampai elemen yang dicari sudah ditemukan, atau indeks kiri lebih besar dari indeks kanan.

Algoritma ini lebih efisien daripada algoritma pencarian sekuensial, tetapi pencarian ini mempunyai syarat yaitu bahwa kumpulan data yang harus dilakukan pencarian harus sudah **terurut** terlebih dahulu, baik terurut secara menaik (ascendant) atau menurun (descendant). Karena data sudah terurut, algoritma dapat menentukan apakah nilai data yang dicari berada sebelum atau sesudah elemen larik yang sedang dibandingkan pada suatu saat. Dengan cara ini, algoritma dapat lebih menghemat waktu pencarian.

Pencarian dalam data terurut bermanfaat misalnya pada penyimpanan data dengan beberapa komponen, program dapat mencari sebuah indeks terurut. Setelah menemukan indeks yang dicari, program dapat membaca data lain yang bersesuaian dengan indeks yang ditemukan tersebut.

Keterurutan data menentukan algoritma biner yang digunakan, karena algoritma biner untuk data terurut menaik sedikit berbeda dengan algoritma biner untuk data terurut menurun, tetapi intinya sama yaitu membagi data atas dua bagian.

Untuk penerapan dalam program, pencarian biner akan dibuat dalam suatu prosedur yang akan dipanggil dari main atau dari prosedur yang lainnya. Prosedur pencarian akan digunakan untuk mencari data pada suatu array, dimana di dalam array terdapat sekumpulan data angka.

Berikut adalah penerapan pencarian biner dalam bahasa pemrograman C untuk data terurut **menaik** :

1. Pencarian Biner untuk data terurut menaik yang mengembalikan posisi data

|  |
| --- |
| #include <stdio.h>  #define TRUE 1  #define FALSE 0  typedef int bool;  void BinSearch(int L[7],int N,int X,int \*idx);  void main()  {  int pos,arr[7] = {0,4,6,7,12,45,67};  BinSearch(arr,7,7,&pos);  if(pos!=-1)  printf("Ketemu di posisi ke-%d",pos);  else  printf("Tidak Ketemu");  }  void BinSearch(int L[7],int N, int X,int \*idx)  {  int i,j,k;  bool ketemu;  i=0;  j=N;  ketemu = FALSE;  while ((!ketemu) && (i<=j))  {  k=(i+j)/2;  if (L[k]==X)  ketemu=TRUE;  else  {  if(L[k]<X)  i=k+1;  else  j=k-1;  }  }  if (ketemu)  \*idx = k+1;  else  \*idx =-1;  } |

Prosedur BinSearch pada program di atas dapat bekerja pada larik yang telah terurut menaik (*ascending*). Prosedur memerlukan parameter input berupa:

* L yaitu sebuah larik (array) tempat menyimpan data yang diinginkan,
* N yaitu jumlah elemen larik,
* X yaitu nilai data yang ingin dicari

serta sebuah parameter output berupa nilai idx (mengembalikan posisi nilai jika nilai ditemukan, dan mengembalikan -1 jika nilai tidak ditemukan).

Prosedur dimulai dengan inisialisasi pencacah (i🡨0) dan menyimpan jumlah elemen larik dalam variabel j. Variabel ketemu akan diberi nilai false, hal ini memberitahukan bahwa data yang dicari belum ditemukan. Perulangan diawali dengan membandingkan elemen tengah (elemen dengan indeks k) dengan nilai data yang dicari.

* Jika elemen larik yang dibandingkan bernilai sama dengan data yang dicari, maka variabel boolean ketemu bernilai true dan prosedur mengembalikan nilai indeks elemen tersebut.
* Jika elemen larik bernilai lebih kecil dari data yang dicari, maka pencarian akan bergeser ke kanan dengan cara mengubah nilai i (awal pencacah) dengan indeks di sebelah kanan nilai tengah (i🡨k+1).
* Jika elemen larik bernilai lebih besar dari data yang dicari, maka pencarian akan bergeser ke kiri dengan cara mengubah nilai i (awal pencacah) dengan indeks di sebelah kiri nilai tengah (j🡨k-1).

Selanjutnya, perulangan terus dilakukan sampai ketemu bernilai true atau nilai i>j (semua elemen larik sudah dibandingkan dengan nilai yang dicari). Jika semua elemen larik sudah dibandingkan dengan nilai yang dicari tetapi nilai tidak ditemukan, maka nilai ketemu akan tetap bernilai false sehingga akan dikembalikan nilai index= -1 (penanda bahwa nilai yang dicari tidak ditemukan dalam larik).

1. Pencarian Biner untuk data terurut menaik berdasar inputan user

|  |
| --- |
| #include <stdio.h>  #define TRUE 1  #define FALSE 0  typedef int bool;  void BinSearch(int L[10],int N,int X,int \*idx);  void inputData(int arr[]);  void bubbleSort(int arr[],int N);  void main()  {  int pos,cari,arr[10];  //panggil fungsi input Data  inputData(arr);  //panggil fungsi sort  bubbleSort(arr,10);  //user input elemen yang dicari  printf("\nElemen yang dicari : ");  scanf("%d",&cari);  //menjalankan pencarian  BinSearch(arr,10,cari,&pos);  if(pos!=-1)  printf("Ketemu di posisi ke-%d",pos);  else  printf("Tidak Ketemu");  }  void inputData(int arr[])  {  int i;  printf("\nMassukkan Elemen Array\n");  //user input elemen array  for(i=0;i<10;i++)  {  printf("Array ke %d : ",i+1);  scanf("%d",&arr[i]);  }  }  void bubbleSort(int arr[],int N)  {  int a,b,temp;  //proses sortir dengan bubble sort  for(a=0;a<=(N-2);a++)  {  for(b=(N-1);b>=(a+1);b--)  {  if (arr[b-1] > arr[b])  {  temp = arr[b-1];  arr[b-1]= arr[b];  arr[b] = temp;  } //endif  } //end loop j  } //end loop i  //tampil data urut  printf("\nData terurut");  for(a=0;a<N;a++)  {  printf("\nArray ke-%d : ",a+1);  printf("%d",arr[a]);  }  }  void BinSearch(int L[10],int N, int X,int \*idx)  {  int i,j,k;  bool ketemu;  i=0;  j=N;  ketemu = FALSE;  while ((!ketemu) && (i<=j))  {  k=(i+j)/2;  if (L[k]==X)  ketemu=TRUE;  else  {  if(L[k]<X)  i=k+1;  else  j=k-1;  }  }  if (ketemu)  \*idx = k+1;  else  \*idx =-1;  } |

Pada program diatas terdapat procedure sorting, hal ini diperlukan karena kemungkinan user akan memasukkan nilai yang tidak terurut, padahal syarat pencarian biner adalah data harus terurut terlebih dahulu.

1. Implementasi Pencarian Biner untuk data terurut menaik ke data bentukan

|  |
| --- |
| #include <stdio.h>  #define TRUE 1  #define FALSE 0  typedef int bool;  struct dataMahasiswa  {  char nim[10];  char nama[20];  };  void inputDataMhs(struct dataMahasiswa mhs[]);  void BinSearch(struct dataMahasiswa L[],int N, char X[10],int \*idx);  void bubbleSort(struct dataMahasiswa mhs[],int N);  void main()  {  int pos;  char pil;  char nimCari[10];  struct dataMahasiswa mhs[5];  //panggil function input data  inputDataMhs(mhs);  //panggil fungsi sorting  bubbleSort(mhs,5);  while(1)  {  //user input elemen yang dicari  printf("\nNim yang dicari : ");  scanf("%s",&nimCari);  //menjalankan pencarian  BinSearch(mhs,5,nimCari,&pos);  if (pos!=-1)  {  printf("\nData Mahasiwa Ketemu");  printf("\nNim siswa : %s",mhs[pos].nim);  printf("\nNama siswa : %s",mhs[pos].nama);  }  else  printf("\nData Mahasiwa Tidak Ketemu");  printf("\nCari data lagi [y/n] ? ");  scanf("%s",&pil);  if ((pil=='y')||(pil=='Y'))  continue;  else  break;  }  }  void inputDataMhs(struct dataMahasiswa mhs[])  {  int i;  printf("\nMasukkan Data Mahasiswa\n");  //user input elemen mahasiswa  for(i=0;i<5;i++)  {  printf("\nData Mahasiswa ke %d : \n",i+1);  printf("Nim : ");  scanf("%s",mhs[i].nim);  printf("Nama : ");  scanf("%s",mhs[i].nama);  }  }  void bubbleSort(struct dataMahasiswa mhs[],int N)  {  int a,b;  struct dataMahasiswa temp;  //proses sortir dengan bubble sort  for(a=0;a<=(N-2);a++)  {  for(b=(N-1);b>=(a+1);b--)  {  if (strcmp(mhs[b-1].nim,mhs[b].nim) >0)  {  temp = mhs[b-1];  mhs[b-1]= mhs[b];  mhs[b] = temp;  } //endif  } //end loop j  } //end loop i  }  void BinSearch(struct dataMahasiswa L[],int N, char X[10],int \*idx)  {  int i,j,k;  bool ketemu;  i=0;  j=N;  ketemu = FALSE;  while ((!ketemu) && (i<=j))  {  k=(i+j)/2;  if (strcmp(L[k].nim,X)==0)  ketemu=TRUE;  else  {  if(strcmp(L[k].nim,X) < 0)  i=k+1;  else  j=k-1;  }  }  if (ketemu)  \*idx = k;  else  \*idx =-1;  } |

Berikut adalah penerapan pencarian biner dalam bahasa pemrograman C untuk data terurut **menurun** :

1. Pencarian Biner untuk data terurut menurun yang mengembalikan posisi data

|  |
| --- |
| #include <stdio.h>  #define TRUE 1  #define FALSE 0  typedef int bool;  void BinSearch(int L[7],int N, int X, int \*idx);  void main()  {  int pos,arr[7] = {45,22,16,10,6,2,0};  BinSearch(arr,7,16,&pos);  if(pos!= -1)  printf("Ketemu di posisi ke-%d",pos);  else  printf("Tidak Ketemu");  }  void BinSearch(int L[7],int N, int X, int \*idx)  {  int i,j,k;  bool ketemu;  i=0;  j=N;  ketemu = FALSE;  while ((!ketemu) && (i<=j))  {  k=(i+j)/2;  if (L[k]==X)  ketemu=TRUE;  else  {  if(L[k]<X)  j=k-1;  else  i=k+1;  }  }  if (ketemu)  \*idx = k+1;  else  \*idx =-1;  } |

Prosedur di atas bekerja untuk pencarian data di dalam larik, dimana data terurut menurun (dari besar ke kecil). Jika nilai ditemukan dalam larik, prosedur mengembalikan nilai idx (indeks elemen larik yang nilainya sama dengan nilai yang dicari). Jika semua elemen larik sudah dibandingkan dengan nilai yang dicari namun tidak ditemukan, prosedur akan mengembalikan nilai idx🡨-1 (penanda bahwa nilai yang dicari tidak ditemukan dalam larik).

Prosedur dimulai dengan inisialisasi pencacah (i🡨0) dan menyimpan jumlah elemen larik dalam variabel j. Variabel ketemu akan diberi nilai false, hal ini memberitahukan bahwa data yang dicari belum ditemukan. Perulangan diawali dengan membandingkan elemen tengah (elemen dengan indeks k) dengan nilai data yang dicari.

* Jika elemen larik yang dibandingkan bernilai sama dengan data yang dicari, maka variabel boolean ketemu bernilai true dan prosedur mengembalikan nilai indeks elemen tersebut.
* Jika elemen larik bernilai lebih kecil dari data yang dicari, maka pencarian akan bergeser ke kiri dengan cara mengubah nilai i (awal pencacah) dengan indeks di sebelah kanan nilai tengah (j🡨k-1).
* Jika elemen larik bernilai lebih besar dari data yang dicari, maka pencarian akan bergeser ke kanan dengan cara mengubah nilai i (awal pencacah) dengan indeks di sebelah kiri nilai tengah (i🡨k+1).

Selanjutnya, perulangan terus dilakukan sampai ketemu bernilai true atau nilai i>j (semua elemen larik sudah dibandingkan dengan nilai yang dicari). Jika semua elemen larik sudah dibandingkan dengan nilai yang dicari tetapi nilai tidak ditemukan, maka nilai ketemu akan tetap bernilai false sehingga akan dikembalikan nilai index= -1 (penanda bahwa nilai yang dicari tidak ditemukan dalam larik).

1. Pencarian Biner untuk data terurut menurun berdasar inputan user

|  |
| --- |
| #include <stdio.h>  #define TRUE 1  #define FALSE 0  typedef int bool;  void BinSearch(int L[10],int N,int X,int \*idx);  void inputData(int arr[]);  void bubbleSort(int arr[],int N);  void main()  {  int pos,cari,arr[10];  inputData(arr);  //panggil fungsi sort  bubbleSort(arr,10);  //user input elemen yang dicari  printf("\nElemen yang dicari : ");  scanf("%d",&cari);  //menjalankan pencarian  BinSearch(arr,10,cari,&pos);  if(pos!=-1)  printf("Ketemu di posisi ke-%d",pos);  else  printf("Tidak Ketemu");  }  void inputData(int arr[])  {  int i;  printf("\nMassukkan Elemen Array\n");  //user input elemen array  for(i=0;i<10;i++)  {  printf("Array ke %d : ",i+1);  scanf("%d",&arr[i]);  }  }  void bubbleSort(int arr[],int N) //sort descending  {  int a,b,temp;  //proses sortir dengan bubble sort  for(a=0;a<=(N-2);a++)  {  for(b=(N-1);b>=(a+1);b--)  {  if (arr[b-1] < arr[b])  {  temp = arr[b-1];  arr[b-1]= arr[b];  arr[b] = temp;  } //endif  } //end loop j  } //end loop i  //tampil data urut  printf("\nData terurut");  for(a=0;a<N;a++)  {  printf("\nArray ke-%d : ",a+1);  printf("%d",arr[a]);  }  }  //search descending  void BinSearch(int L[10],int N, int X,int \*idx)  {  int i,j,k;  bool ketemu;  i=0;  j=N;  ketemu = FALSE;  while ((!ketemu) && (i<=j))  {  k=(i+j)/2;  if (L[k]==X)  ketemu=TRUE;  else  {  if(L[k]<X)  j=k-1;  else  i=k+1;  }  }  if (ketemu)  \*idx = k+1;  else  \*idx =-1;  } |

1. Implementasi Pencarian Biner untuk data terurut menurun ke data bentukan

|  |
| --- |
| #include <stdio.h>  #define TRUE 1  #define FALSE 0  typedef int bool;  struct dataMahasiswa  {  char nim[10];  char nama[20];  };  void inputDataMhs(struct dataMahasiswa mhs[]);  void BinSearch(struct dataMahasiswa L[],int N, char X[10],int \*idx);  void bubbleSortDown(struct dataMahasiswa mhs[],int N);  void main()  {  int pos;  char pil;  char nimCari[10];  struct dataMahasiswa mhs[5];  //panggil function input data  inputDataMhs(mhs);  //panggil fungsi sorting  bubbleSortDown(mhs,5);  while(1)  {  //user input elemen yang dicari  printf("\nNim yang dicari : ");  scanf("%s",&nimCari);  //menjalankan pencarian  BinSearch(mhs,5,nimCari,&pos);  if (pos!=-1)  {  printf("\nData Mahasiwa Ketemu");  printf("\nNim siswa : %s",mhs[pos].nim);  printf("\nNama siswa : %s",mhs[pos].nama);  }  else  printf("\nData Mahasiwa Tidak Ketemu");  printf("\nCari data lagi [y/n] ? ");  scanf("%s",&pil);  if ((pil=='y')||(pil=='Y'))  continue;  else  break;  }  }  void inputDataMhs(struct dataMahasiswa mhs[])  {  int i;  printf("\nMasukkan Data Mahasiswa\n");  //user input elemen mahasiswa  for(i=0;i<5;i++)  {  printf("\nData Mahasiswa ke %d : \n",i+1);  printf("Nim : ");  scanf("%s",mhs[i].nim);  printf("Nama : ");  scanf("%s",mhs[i].nama);  }  }  void bubbleSortDown(struct dataMahasiswa mhs[],int N)  {  int a,b;  struct dataMahasiswa temp;  //proses sortir dengan bubble sort  for(a=0;a<=(N-2);a++)  {  for(b=(N-1);b>=(a+1);b--)  {  if (strcmp(mhs[b-1].nim,mhs[b].nim) <0)  {  temp = mhs[b-1];  mhs[b-1]= mhs[b];  mhs[b] = temp;  } //endif  } //end loop j  } //end loop i  }  void BinSearch(struct dataMahasiswa L[],int N, char X[10],int \*idx)  {  int i,j,k;  bool ketemu;  i=0;  j=N;  ketemu = FALSE;  while ((!ketemu) && (i<=j))  {  k=(i+j)/2;  if (strcmp(L[k].nim,X)==0)  ketemu=TRUE;  else  {  if(strcmp(L[k].nim,X) < 0)  j=k-1;  else  i=k+1;  }  }  if (ketemu)  \*idx = k;  else  \*idx =-1;  } |

## *Pencarian Lain*

Pencarian sekuensial dan pencarian biner merupakan algoritma pencarian dasar yang termasuk ke dalam kelompok pencarian daftar (*list search*). Terdapat pula beberapa algoritma lain yang termasuk pula dalam kelompok pencarian daftar, antara lain:

* pencarian interpolasi (*interpolation search*): melakukan pencarian lebih baik daripada pencarian biner pada larik berukuran besar dengan distribusi seimbang, tapi waktu pencariannya buruk
* pencarian Grover (*Grover’s search*): melakukan pencarian dalam waktu singkat, yang merupakan pengembangan dari pencarian linier biasa pada larik dengan elemen tidak berurut

Selain algoritma pencarian dalam kelompok pencarian daftar, terdapat pula beberapa kelompok algoritma lain. Beberapa di antaranya adalah sebagai berikut:

|  |  |
| --- | --- |
| **Kelompok Algoritma** | **Penjelasan dan Contoh Algoritma** |
| Pencarian tanpa informasi (*uninformed search*) | Algoritma ini mencari data tanpa ada batasan tipe data persoalan. |
| Pencarian pohon  (*tree search*) | Algoritma ini mencari data dengan bantuan struktur pohon (eksplisit maupun implisit).   * *breadth-first search* (mencari level demi level) * *depth-first search* (mencari dengan meraih kedalaman pohon terlebih dahulu) * *iterative-deepening search* * *depth-limited search* * *bidirectional search* * *uniform-cost search* |
| Pencarian grafik  (*graph search*) | Algoritma ini mencari data dengan algoritma penelurusuran grafik, misalnya   * *Djikstra’s algorithm* * *Kruskal’s algorithm* * *nearest neighbour algorithm* * *Prim’s algorithm* |
| Pencarian dengan informasi  (*informed search*) | Algoritma ini mencari data dengan fungsi heuristik yang spesifik pada persoalan tertentu.   * *best-first search* * *A\** |
| Jenis lain | * Algoritma pencarian string * Algoritma genetik * Algoritma *minimax* |

|  |  |
| --- | --- |
|  | **Rangkuman** |

1. Algoritma pencarian adalah algoritma yang mengambil input berupa persoalan dan mengembalikan penyelesaian berupa penemuan nilai yang dicari dalam persoalan inputan.
2. Dua contoh algoritma pencarian dasar adalah pencarian sekuensial dan pencarian biner.
3. Pencarian sekuensial (*sequential search*) membandingkan setiap elemen larik (array) satu persatu dengan nilai yang dicari secara beruntun, mulai dari elemen pertama sampai elemen yang dicari sudah ditemukan, atau sampai seluruh elemen sudah diperiksa.
4. Pencarian biner adalah proses mencari data dengan membagi data atas dua bagian secara terus menerus sampai elemen yang dicari sudah ditemukan.
5. Pencarian biner mempunyai prasyarat yaitu data harus terurut baik menaik atau menurun.

|  |  |
| --- | --- |
|  | **Latihan** |

1. Buatlah program kamus bahasa inggris – bahasa indonesia, minimal 10 kata. Inputan berupa kata dalam bahasa inggris dan output berupa arti kata dalam bahasa indonesianya. Gunakan proses pencarian dengan Sequential Search.

Hint : Gunakan tipe data bentukan

|  |
| --- |
| struct kamus  {  char inggris[20];  char indonesia[20];  }; |

1. Pecahkan persoalan pada no.1 dengan menggunakan Binary Search
2. Buatlah program untuk pencarian data Barang pada suatu supermarket. Data barang akan diinputkan oleh user. Tiap barang mempunyai atribut : Kode Barang, Nama Barang, dan Harga Barang. Pencarian dilakukan dengan user input Kode Barang kemudian ditampilkan data Nama Barang dan Harganya.
3. Tambahkan pada soal no 3, yaitu fitur untuk update harga barang. Setelah dilakukan pencarian, user diminta untuk memasukkan harga barang yang baru, kemudian harga yang baru disimpan.
4. Tambahkan pada soal no 3, yaitu fitur untuk delete barang. Setelah dilakukan pencarian kode barang kemudian hapus data barang tersebut.
5. Gabungkan soal no 3, 4, dan 5 sehingga program tersebut akan menampilkan menu :

MENU :

1. Input Data
2. Update Data
3. Delete Data
4. Search Data

Menu input data digunakan untuk memasukkan data barang pada posisi yang masih kosong.

Menu update data digunakan untuk mengubah data harga barang sesuai kodenya

Menu delete data digunakan untuk menghapus data barang

Menu search data digunakan untuk menampilkan data barang.

# *Pengurutan (Sorting)*

|  |  |  |
| --- | --- | --- |
|  | | **Overview** |
| Proses *sorting* (pengurutan data) merupakan proses yang seringkali dilakukan dalam pengolahan data. Terdapat beberapa algoritma yang dapat digunakan dalam proses pengurutan data. Dalam bab ini akan diberikan contoh beberapa penerapan algoritma *sorting* dalam *array* satu dimensi, menggunakan bahasa pemrograman C. Adapun penerapan beberapa algoritma *sorting* ini mencakup penerapan algoritma *bubble sort*, *selection dan insertion sort* dengan bahasa C. | | |
|  | | |
|  | **Tujuan** | |
| 1. Memahami konsep pengurutan 2. Mengenal beberapa algoritma pengurutan 3. Menerapkan beberapa algoritma pengurutan dalam program dengan menggunakan bahasa C. | | |



## Pengertian Sort

*Sorting* atau proses *sort* adalah proses mengurutkan nilai baik urut menaik (*ascending*) maupun urut menurun (*descending*). Nilai yang diurut dapat bertipe numerik maupun *character* (*alphanumeric*). Data yang akan diurutkan dapat berada dalam memori (*internal sort*) atau dapat juga dalam *external storage* (*external sort*).

Berikut ini akan diberikan beberapa contoh penerapan algoritma pengurutan (*bubble sort*, *selection sort* dan *insertion sort*) dalam bahasa C.

## Bubble Sort

Algoritma *bubble sort*, membandingkan elemen yang sekarang dengan elemen berikutnya. Jika elemen sekarang > dari elemen berikutnya, maka tukar. Pengecekan dapat dimulai dari data paling awal atau paling akhir. Dalam contoh berikut ini, pengecekan dimulai dari data yang paling akhir. Data paling akhir dibandingkan dengan data didepannya, jika ternyata lebih kecil maka ditukar. Dan pengecekan yang sama dilakukan terhadap data yang selanjutnya sampai dengan data yang paling awal.

**Proses *Bubble Sort***

Misal *array* L memiliki 5 elemen data (N=5).

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Data awal: | | 22 | 10 | 15 | 3 | 8 |
|  | 0 | 1 | 2 | 3 | 4 |

Dari data awal, bandingkan nilai pada indeks (N-1) dengan nilai pada indeks (N-2). Jika nilai pada indeks (N-2) > (N-1) maka tukar tempat. Kemudian pembandingan dilanjutkan untuk nilai pada indeks (N-2) dengan nilai pada indeks (N-3) jika nilai pada indeks (N-3) > (N-2) maka tukar tempat. Proses dilanjutkan sampai nilai terkecil sudah berada pada tempatnya, yaitu pada indeks 0

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Hasil Pass 1: | | | 3 | 22 | 10 | 15 | 8 |
|  | 0 | | 1 | 2 | 3 | 4 |

Dari hasil setelah proses Pass 1 selesai dilakukan, nilai pada indeks (N-1) dibandingkan dengan nilai pada indeks (N-2), dan seterusnya proses pembandingan dilakukan sampai nilai terkecil kedua berada pada tempatnya, yaitu indeks 1.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Hasil Pass 2: | | 3 | 8 | 22 | 10 | 15 |
|  | 0 | 1 | 2 | 3 | 4 |

Dari hasil setelah proses Pass 1 selesai dilakukan, nilai pada indeks (N-1) dibandingkan dengan nilai pada indeks (N-2), dan seterusnya proses pembandingan dilakukan sampai nilai terkecil kedua berada pada tempatnya, yaitu indeks 2. Proses ini terus dilakukan sampai pass 4, dan data terurut secara *ascending*.

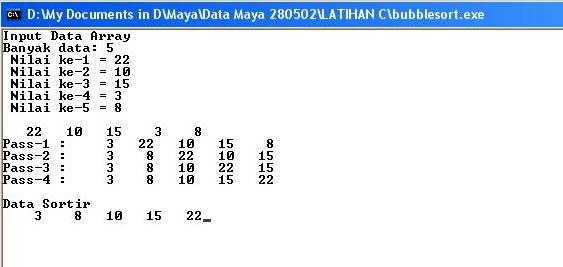
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Hasil Pass 3: | | 3 | 8 | 10 | 22 | 15 |
|  | 0 | 1 | 2 | 3 | 4 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Hasil Pass 4: | | 3 | 8 | 10 | 15 | 22 |
|  | 0 | 1 | 2 | 3 | 4 |

Berikut ini adalah contoh penerapan algoritma *Bubble Sort Ascending* dalam bahasa C. **Perhatikan** pola **perubahan urutan nilai** yang terjadi dalam setiap pass-nya dari output yang dihasilkan.

|  |
| --- |
| 1. #include <stdio.h>  2. #include <conio.h>  3. main()  4. { int L[5];  5. int i,j,temp,a;  6. int N;  7. void v\_tampil(int A[5],int N); //deklarasi procedure  8.  9. printf("Input Data Array\n");  10. printf("Banyak data: "); scanf("%i",&N);  11. for(i=0;i<N;i++)  12. { printf(" Nilai ke-%i = ",i+1);  13. scanf("%i",&L[i]); } //end loop i  14.  15. printf("\n");  16. v\_tampil(L,N); //panggil procedure v\_tampil  17.  18. for(i=0;i<=(N-2);i++)  19. { for(j=(N-1);j>=(i+1);j--)  20. { if (L[j-1] > L[j])  21. { temp = L[j-1];  22. L[j-1]= L[j];  23. L[j] = temp; } //endif  24. } //end loop j  25. printf("\nPass-%i : ",i+1);  26. v\_tampil(L,N); //panggil procedure v\_tampil  27. } //end loop i  28.  29. printf("\n\nData Sortir\n");  30. v\_tampil(L,N); //panggil procedure v\_tampil  31. getche();  32.} //end main program  33.  34. void v\_tampil(int A[5],int N)  35. { int p;  36. for(p=0;p<N;p++)  37. { printf("%5i",A[p]); }  38. } //end procedure v\_tampil |

Output yang dihasilkan:



## Selection Sort

## Algoritma *Selection sort* memilih elemen maksimum/minimum *array*, lalu menempatkan elemen maksimum/minimum itu pada awal atau akhir *array* (tergantung pada urutannya *ascending*/*descending*).

Terdapat dua pendekatan dalam metode pengurutan dengan *Selection Sort* :

1. **Algoritma pengurutan maksimum** (*maximum selection sort*), yaitu memilih **elemen maksimum sebagai basis pengurutan**.
2. **Algoritma pengurutan minimum** (*minimum selection sort*), yaitu memilih **elemen minimum sebagai basis pengurutan**.

### *Maximum* *Selection Sort Ascending*

Algoritma maksimum *selection sort ascending*, memilih elemen dengan nilai terbesar sebagai basis pengurutannya. Proses program pertama kali dilakukan dengan mencari nilai terbesar (dari indeks 0..N-1)dalam *array*, kemudian elemen terbesar tersebut dipindahkan pada urutan terakhir (indeks terakhir, yaitu N-1) dalam *array*. Proses ke-2 dilakukan dengan mencari nilai terbesar dari indeks 0..N-2, dan elemen terbesar yang ditemukan dipindahkan pada indeks N-2 dalam array, dan demikian seterusnya sampai seluruh data selesai dibandingkan dan terurut.

**Proses *Maximum* *Selection Sort Ascending***

Misal *array* L memiliki 5 elemen data (N=5).

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Data awal: | | 24 | 10 | 25 | 34 | 12 |  |
|  | 0 | 1 | 2 | 3 | 4 |  |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Hasil Pass 1: | | | 24 | 10 | 25 | 12 | 34 | Dari data awal, cari nilai terbesar pada indeks 0-4, pindahkan ke indeks N-1 (indeks 4) |
|  | 0 | | 1 | 2 | 3 | 4 |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Hasil Pass 2: | | 24 | 10 | 12 | 25 | 34 | Dari hasil pass 1, cari nilai terbesar pada indeks 0..3, lalu pindahkan ke indeks N-2 (indeks 3) |
|  | 0 | 1 | 2 | 3 | 4 |  |

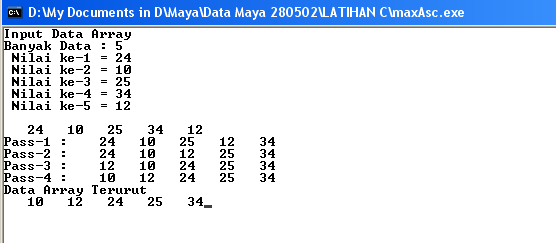
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Hasil Pass 3: | | 12 | 10 | 24 | 25 | 34 | Dari hasil pass 2, cari nilai terbesar pada indeks 0-2, lalu pindahkan ke indeks N-3 (indeks 2) |
|  | 0 | 1 | 2 | 3 | 4 |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Hasil Pass 4: | | 10 | 12 | 24 | 25 | 34 | Dari hasil pass 3, cari nilai terbesar pada indeks  0-1, lalu pindahkan ke indeks N-4 (indeks 1) |
|  | 0 | 1 | 2 | 3 | 4 |  |

Berikut ini adalah contoh penerapan algoritma *Maximum Selection Sort Ascending* dalam bahasa C. **Perhatikan** pola **perubahan urutan nilai** yang terjadi dalam setiap pass-nya dari output yang dihasilkan.

|  |
| --- |
| 1. #include <stdio.h>  2. #include <conio.h>  3. main()  4. { int A[5];  5. int i,k,j,temp,maks,a,N;  6. void v\_tampil(int A[5],int N); //deklarasi procedure  7.  8. printf("Input Data Array\n");  9. printf("Banyak Data : ");scanf("%i",&N);  10. for(i=0;i<N;i++)  11. { printf(" Nilai ke-%i = ",i+1);  12. scanf("%i",&A[i]); } //end loop i  13. a=0;  14. printf("\n");  15. v\_tampil(A,N); //panggil procedure v\_tampil  16. for(k=(N-1);k>0;k--)  17. { maks=0;  18. for(j=0;j<=k;j++)  19. { if (A[j] > A[maks])  20. { maks=j; } //endif  21. } //end loop j  22. temp = A[k];  23. A[k] = A[maks];  24. A[maks] = temp;  25.  26. printf("\nPass-%i : ",a+1);  27. v\_tampil(A,N); //panggil procedure v\_tampil  28. a++;  29. } //end loop k  30.  31. printf("\nData Array Terurut\n");  32. v\_tampil(A,N);  33. getche();  34.} //end main program  35.  36. void v\_tampil(int A[5],int N)  37. { int i;  38. for(i=0;i<N;i++)  39. { printf("%5i",A[i]); } //end loop i  40. } //end procedure v\_tampil |

Output yang dihasilkan:



### *Maximum* *Selection Sort Descending*

Algoritma maksimum *selection sort descending*, memilih elemen dengan nilai terbesar sebagai basis pengurutannya. Proses program pertama kali dilakukan dengan mencari nilai terbesar (dari indeks 0..N-1) dalam *array*, kemudian elemen terbesar tersebut dipindahkan pada urutan pertama (indeks pertama, yaitu 0) dalam *array*. Proses ke-2 dilakukan dengan mencari nilai terbesar dari indeks 1..N-1, dan elemen terbesar yang ditemukan dipindahkan pada indeks 1 dalam array, dan demikian seterusnya sampai seluruh data selesai dibandingkan dan terurut.

**Proses *Maximum* *Selection Sort Descending***

Misal *array* L memiliki 5 elemen data (N=5).

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Data awal: | | 24 | 10 | 25 | 34 | 12 |
|  | 0 | 1 | 2 | 3 | 4 |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Hasil Pass 1: | | | 34 | 10 | 25 | 24 | 12 | Dari data awal, cari nilai terbesar pada indeks 0-4, lalu pindahkan ke indeks 0 |
|  | 0 | | 1 | 2 | 3 | 4 |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Hasil Pass 2: | | 34 | 25 | 10 | 24 | 12 | Dari hasil pass 1, cari nilai terbesar pada indeks 1-4, lalu pindahkan ke indeks 1 |
|  | 0 | 1 | 2 | 3 | 4 |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Hasil Pass 3: | | 34 | 25 | 24 | 10 | 12 | Dari hasil pass 2,cari nilai terbesar pada indeks 2-4, lalu pindahkan ke indeks 2 |
|  | 0 | 1 | 2 | 3 | 4 |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Hasil Pass 4: | | 34 | 25 | 24 | 12 | 10 | Dari hasil pass 3,cari nilai terbesar pada indeks 3-4, lalu pindahkan ke indeks 3 |
|  | 0 | 1 | 2 | 3 | 4 |  |

|  |
| --- |
| Berdasarkan dua contoh program sorting sebelumnya yang telah diberikan, coba Anda buat sendiri penerapan algoritma *Maximum Selection Sort Descending* dalam bahasa C !. |

### *Minimum* *Selection Sort Ascending*

Algoritma minimum *selection sort ascending*, memilih elemen dengan nilai terkecil sebagai basis pengurutannya. Proses program pertama kali dilakukan dengan mencari nilai terkecil (dari indeks 0..N-1) dalam *array*, kemudian elemen terkecil tersebut dipindahkan pada urutan pertama (indeks pertama, yaitu 0) dalam *array*. Proses ke-2 dilakukan dengan mencari nilai terkecil dari indeks 1..N-1, dan elemen terkecil yang ditemukan dipindahkan pada indeks 1 dalam array, dan demikian seterusnya sampai seluruh data selesai dibandingkan dan terurut.

**Proses *Minimum* *Selection Sort Ascending***

Misal *array* L memiliki 5 elemen data (N=5).

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Data awal: | | 7 | 1 | 6 | 2 | 4 |
|  | 0 | 1 | 2 | 3 | 4 |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Hasil Pass 1: | | | 1 | 7 | 6 | 2 | 4 | Dari data awal, cari nilai terkecil pada indeks 0-4, lalu pindahkan ke indeks 0 |
|  | 0 | | 1 | 2 | 3 | 4 |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Hasil Pass 2: | | 1 | 2 | 6 | 7 | 4 | Dari hasil pass 1, cari nilai terkecil pada indeks 1-4, lalu pindahkan ke indeks 1 |
|  | 0 | 1 | 2 | 3 | 4 |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Hasil Pass 3: | | 1 | 2 | 4 | 7 | 6 | Dari hasil pass 2, cari nilai terkecil pada indeks 2-4, lalu pindahkan ke indeks 2 |
|  | 0 | 1 | 2 | 3 | 4 |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Hasil Pass 4: | | 1 | 2 | 4 | 6 | 7 | Dari hasil pass 3,cari nilai terkecil pada indeks 3-4, lalu pindahkan ke indeks 3 |
|  | 0 | 1 | 2 | 3 | 4 |  |

|  |
| --- |
| Berdasarkan dua contoh program sorting sebelumnya yang telah diberikan, coba Anda buat sendiri penerapan algoritma *Minimum Selection Sort Ascending* dalam bahasa C !. |

### *Minimum* *Selection Sort Descending*

Algoritma minimum *selection sort descending*, memilih elemen dengan nilai terkecil sebagai basis pengurutannya. Proses program pertama kali dilakukan dengan mencari nilai terkecil (dari indeks 0..N-1) dalam *array*, kemudian elemen terkecil tersebut dipindahkan pada urutan terakhir (indeks terakhir, yaitu N-1) dalam *array*. Proses ke-2 dilakukan dengan mencari nilai terkecil dari indeks 0..N-2, dan elemen terkecil yang ditemukan dipindahkan pada indeks N-2 dalam array, dan demikian seterusnya sampai seluruh data selesai dibandingkan dan terurut.

**Proses *Minimum* *Selection Sort Descending***

Misal *array* L memiliki 5 elemen data (N=5).

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Data awal: | | 7 | 1 | 6 | 2 | 4 |
|  | 0 | 1 | 2 | 3 | 4 |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Hasil Pass 1: | | | 7 | 4 | 6 | 2 | 1 | Dari data awal,cari nilai terkecil pada indeks 0-4, lalu pindahkan ke indeks N-1 (indeks 4) |
|  | 0 | | 1 | 2 | 3 | 4 |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Hasil Pass 2: | | 7 | 4 | 6 | 2 | 1 | Dari hasil pass 1,cari nilai terkecil pada indeks 0-3, lalu pindahkan ke indeks N-2 (indeks 3) |
|  | 0 | 1 | 2 | 3 | 4 |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Hasil Pass 3: | | 7 | 6 | 4 | 2 | 1 | Dari hasil pass 2,cari nilai terkecil pada indeks 0-2, lalu pindahkan ke indeks N-3 (indeks 2) |
|  | 0 | 1 | 2 | 3 | 4 |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Hasil Pass 4: | | 7 | 6 | 4 | 2 | 1 | Dari hasil pass 3,cari nilai terkecil pada indeks 0-1, lalu pindahkan ke indeks N-4 (indeks 1) |
|  | 0 | 1 | 2 | 3 | 4 |  |

|  |
| --- |
| Berdasarkan dua contoh program sorting sebelumnya yang telah diberikan, coba Anda buat sendiri penerapan algoritma *Minimum Selection Sort Descending* dalam bahasa C !. |

## *Insertion Sort*

Metode penyisipan (i*nsertion sort*) bertujuan untuk menjadikan bagian sisi kiri

*array* terurutkan sampai dengan seluruh *array* berhasil diurutkan. Metode ini mengurutkan bilangan-bilangan yang telah dibaca; dan berikutnya secara berulang akan menyisipkan bilanganbilangan dalam *array* yang belum terbaca ke sisi kiri *array* yang telah terurut.

**Proses *Insertion Sort Ascending***

Misal *array* L memiliki 5 elemen data (N=5).

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Data awal: | | 45 | 12 | 40 | 15 | 9 |
|  | 0 | 1 | 2 | 3 | 4 |

Nilai paling kiri pada indeks 0 (45) bisa dikatakan telah terurut secara relatif thd dirinya sendiri. Cek, untuk melihat apakah nilai kedua (indeks satu,yaitu 12) lebih kecil dari pada yang pertama (45). Jika ya, tukarkan kedua bilangan ini.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Hasil Pass 1: | | | 12 | 45 | 40 | 15 | 9 |  |
|  | 0 | | 1 | 2 | 3 | 4 |  |

Bagian yang diarsir (dua bilangan pertama) sekarang dalam keadaan terurut secara relatif. Berikutnya, kita perlu menyisipkan nilai ketiga (pada indeks 2,yaitu 40) ke dalam bagian abu-abu sehingga setelah penyisipan tersebut, bagian abu-abu tetap dalam keadaan terurut secara relatif. Hal ini dilakukan dengan cara: ambil nilai ketiga pada indeks 2, yaitu 40, kemudian geser bilangan kedua (45) sehingga ada ruang untuk disisipi. Sisipkan bilangan 40 pada posisi yang tepat.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Hasil Pass 2: | | 12 | 40 | 45 | 15 | 9 |  |
|  | 0 | 1 | 2 | 3 | 4 |  |

Sekarang, tiga bilangan pertama sudah terurut secara relatif dan kita sisipkan bilangan keempat kepada tiga bilangan pertama tsb. Setelah penyisipan, empat bilangan pertama haruslah dalam keadaan terurut secara relatif. Ulangi proses sampai bilangan terakhir selesai disisipkan.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Hasil Pass 3: | | 12 | 15 | 40 | 45 | 9 |  |
|  | 0 | 1 | 2 | 3 | 4 |  |

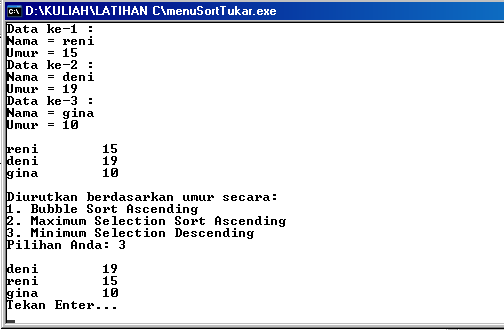
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Hasil Pass 4: | | 9 | 12 | 15 | 40 | 45 |  |
|  | 0 | 1 | 2 | 3 | 4 |  |

|  |
| --- |
| Berdasarkan dua contoh program sorting sebelumnya yang telah diberikan, coba Anda buat sendiri penerapan algoritma *Insertion Sort Ascending* dalam bahasa C !. |

Berikut ini adalah contoh penerapan algoritma *sorting* dengan menggunakan beberapa pilihan metoda *sorting* yang akan digunakan. Program menggunakan struktur record yang terdiri dari 2 buah field, yaitu nama dan umur. Program akan memunculkan hasil *sorting* apakah secara *ascending* maupun *descending* berdasarkan umur tergantung dari pilihan user.

|  |
| --- |
| 1. #include <stdio.h>  2. #include <conio.h>  3. struct siswa { int umur;  4. char nama[10]; };  5.  6. void v\_BubAsc(struct siswa s[3],int N);  7. void v\_tampil(struct siswa s[3],int N);  8. void v\_MaxAsc(struct siswa s[3],int N);  9. void v\_MinDesc(struct siswa s[3],int N);  10. void v\_Tukar(struct siswa \*P,struct siswa \*M);  11. main()  12. { struct siswa ss[3];  13. int i,pil,N;  14. bool jwb;  15. jwb=true;  16. pil=0;  17.  18. while(jwb)  19. { printf("INPUT DATA ARRAY\n");  20. printf("Banyak Data: "); scanf("%i",&N);  21. for(i=0;i<N;i++)  22. { printf("Data ke-%i :\n",i+1);  23. printf("Nama = ");scanf("%s",&ss[i].nama);  24. printf("Umur = ");scanf("%i",&ss[i].umur);  25. } //end loop i  26.  27. v\_tampil(ss,N);  28.  29. printf("\nDiurutkan berdasarkan umur secara:\n");  30. printf("1. Bubble Sort Ascending\n");  31. printf("2. Maximum Selection Sort Ascending\n");  32. printf("3. Minimum Selection Descending\n");  33. printf("Pilihan Anda: ");scanf("%i",&pil);  34. switch (pil)  35. { case 1:  36. v\_BubAsc(ss,N);  37. break;  38. case 2:  39. v\_MaxAsc(ss,N);  40. break;  41. case 3:  42. v\_MinDesc(ss,N);  43. break;  44. } //end switch  45.  46. v\_tampil(ss,N);  47. printf("Tekan Enter...\n");  48. getche();  49. jwb=false;  50. } //endwhile  51. } //end main program  52. void v\_BubAsc(struct siswa s[3],int N)  53. { int a,b;  54. struct siswa temp;  55. //proses sortir dengan bubble sort  56. for(a=0;a<=(N-2);a++)  57. { for(b=(N-1);b>=(a+1);b--)  58. { if (s[b-1].umur > s[b].umur)  59. { v\_Tukar(&s[b-1],&s[b]); }//endif  60. } //end loop b  61. } //end loop a  62. } //end procedure v\_BubAsc  63.  64. void v\_Tukar(**struct** siswa \*P,**struct** siswa \*M)  65. { **struct** siswa temp;  66. temp = \*P;  67. \*P = \*M;  68. \*M = temp;  69. } //end procedure v\_Tukar  70.  71. void v\_tampil(struct siswa s[3],int N)  72. { int i;  73. printf("\n");  74. for(i=0;i<N;i++)  75. { printf("%s%10i\n",s[i].nama,s[i].umur); }  76. } //end procedure v\_tampil  77.  78. void v\_MaxAsc(struct siswa s[3],int N)  79. { int maks,k,j;  80. struct siswa temp;  81. for(k=(N-1);k>=0;k--)  82. { maks=0;  83. for(j=0;j<=k;j++)  84. { if (s[j].umur > s[maks].umur)  85. { maks=j; } //endif  86. } //end loop j  87. v\_Tukar(&s[k],&s[maks]); //panggil v\_Tukar  88. } //end loop k  89. } //end procedure v\_MaxAsc  90.  91. void v\_MinDesc(struct siswa s[3],int N)  92. { int k,j,min;  93. struct siswa temp;  94. //minimum selection sort descending  95. for(k=(N-1);k>=1;k--)  96. { min = 0;  97. for(j=0;j<=k;j++)  98. { if (s[j].umur < s[min].umur)  99. min=j; } //end loop j  100. v\_Tukar(&s[k],&s[min]); //panggil v\_Tukar  101. } //end loop k  102. } //end procedure v\_MinDesc |

Output yang dihasilkan:



|  |  |
| --- | --- |
|  | **Rangkuman** |

1. ***Proses Sorting*** merupakan proses mengurutkan data yang berada dalam suatu tempat penyimpanan, dengan urutan tertentu baik urut menaik (*ascending*) dari nilai terkecil sampai dengan nilai terbesar, atau urut menurun (*descending*) dari nilai terbesar sampai dengan nilai terkecil
2. Terdapat dua macam proses pengurutan, yaitu pengurutan internal (*internal sort*) dan pengurutan eksternal (*external sort*).
3. *Bubble sort* adalah proses pengurutan sederhana yang bekerja dengan cara berulang kali membandingkan dua elemen data pada suatu saat dan menukar elemen data yang urutannya salah.
4. Algoritma *Selection sort* memilih elemen maksimum/minimum *array*, lalu menempatkan elemen maksimum/minimum itu pada awal atau akhir *array* (tergantung pada urutannya *ascending*/*descending*).
5. Algoritma *Insertion Sort*, mencari tempat yang "tepat" untuk setiap elemen array, dengan cara *sequential search*. Proses ini kemudian menyisipkan sebuah elemen array yang diproses ke tempatnya yang seharusnya.

|  |  |
| --- | --- |
|  | **Latihan** |

1. Suatu sekolah ingin mengkomputerisasikan sistem penilaiannya, dan ingin membuat laporan mengenai peringkat 3 nilai terbesar. Data siswa disimpan dalam record sbb :

NIM (Nomor Induk Mahasiswa), Nama Siswa, Nilai UTS, Nilai Quiz, Nilai UAS, Nilai Akhir

**Ketentuan proses sbb :**

1. Asumsikan data siswa tidak lebih dari 5 orang.
2. Nilai akhir didapat dari perhitungan 30 % Nilai UTS + 20 % Nilai Quiz + 50 % Nilai UAS
3. Buat program dalam bahasa C lengkap untuk menampilkan peringkat 3 besar berdasarkan nilai akhir siswa. Dan nilai **diurutkan secara *descending*** berdasarkan **nilai akhir (NA)**.
4. Program yang Anda bangun **harus** menggunakan *procedure* atau *function* berparameter.
5. Output akhir yang diinginkan:

**DAFTAR NILAI MAHASISWA**

**PERINGKAT 3 BESAR**

NO NIM NAMA UTS UAS QUIZ NA

1.

2.

3.

Rata-rata Nilai: 999 999 999 999

2. Suatu Pabrik Kendaraan Bermotor mempunyai beberapa Distributor di berbagai kota. Pabrik tersebut ingin mengkomputerisasikan data pemakaian kredit para distributornya. Data Distributor disimpan dalam record sbb:

Kode Distributor, Nama Distributor, Kota Distributor, Batas Kredit, Pemakaian Kredit

**Ketentuan Proses** :

* Persentasi (%) = Pemakaian Kredit / Batas Kredit x 100 %
* Jika Persentasi (%) > 100 % maka ditandai dengan \*
* Buat program C untuk **mencari dan menampilkan** data distributor **berdasarkan** kota yang **diinput**. Misalkan ingin mencari data distributor yang tinggal di kota Bandung, maka akan tampil **seluruh** data distributor yang tinggal di kota Bandung saja ATAU ingin mencari data distributor yang tinggal di kota Jakarta, maka akan tampil **seluruh** data distributor yang tinggal di kota Jakarta, dsb……(SEARCHING DATA berdasarkan KOTA !!) dan data yang dimunculkan **terurut secara ascending** berdasarkan **persentasi** (%).
* Program harus menggunakan *procedure* berparameter !! Data Distributor dialihkan melalui parameter !! Asumsikan terdapat 5 distributor.

Contoh output akhir yang diinginkan:

DATA DISTRIBUTOR

Kota : Bandung

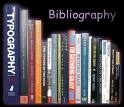
No Kode Nama Kota Batas Kredit Pemakaian Persen Ket

1. D1 aaa Bandung 1000000 800000 80

:

:

5. D5 eee Bandung 1000000 1500000 150 \*



|  |  |
| --- | --- |
|  | Daftar Pustaka |

* + - * 1. Algorithm Data Structures and Problem Solving with C++. 1997. Addison Wesley.
        2. Moh. Sjukani, Algoritma dan Struktur Data. Mitra Wacana Media
        3. Inggriani Liem, Diktat Catatan Singkat Bahasa C Agustus 2003. ITB
        4. Inggriani Liem, Diktat Kuliah Dasar Pemrograman April 2007. ITB
        5. Rinaldi Munir, Algoritma dan Pemrograman. Informatika Bandung
        6. Schaum, Programming with C++ 2nd. 2000. McGraw-Hill
        7. Schaum. Teach yourself C++ in 21 days. 2007. McGraw-Hill