Algoritma Matematika

Seringkali bagi kebanyakan orang permasalahan matematika merupakan suatu hal yang cukup memusingkan. Bahasa C dapat mempermudah kita dalam perhitungan matematika.

Apakah Anda sering kesulitan untuk menyelesaikan permasalahan matematika yang rumit atau yang membutuhkan kesabaran? Kalau hal itu terjadi pada Anda, Anda perlu tahu bahwa kita dapat menggunakan komputer untuk membantu mempermudah pekerjaan kita yang satu ini. Algoritma pemrograman dapat dimanfaatkan untuk memecahkan berbagai macam masalah matematika, kita menyebutnya dengan algoritma matematika. Anda perlu tahu bahwa sangat banyak fungsi matematika yang dapat implementasikan dalam bahasa pemrograman, salah satu bahasa pemrograman yang sering digunakan untuk menulis algoritma matematika yaitu bahasa pemrograman C.

Beberapa contoh algoritma matematika yang dapat ditulis dalam bahasa C misalnya algoritma bilangan prima, fibbonaci, segitiga pascal, dan penjumlahan polinom. Sebenarnya masih banyak lagi persoalan dalam matematika yang dapat dibuat algoritmanya, namun dalam artikel ini saya akan membahas algoritma untuk memecahkan masalah matematika tersebut dengan menggunakan bahasa C. Algoritma yang dapat digunakan untuk menyelesaikan persoalan matematika seperti di atas sangatlah beragam jenisnya, namun tentu saja tidak semuanya dapat saya bahas dalam kesempatan ini.

Source code yang dibahas di dalam artikel ini tidak bersifat absolut sehingga bisa Anda kembangkan menurut gaya Anda sendiri. Tidak tertutup kemungkinan nantinya Anda dapat membuat sendiri algoritma matematika yang lain setelah Anda memahami contoh yang akan kita bahas. Source code program yang ditulis di sini tidak ditujukan untuk pemrograman yang optimal, namun sengaja dibuat supaya algoritma dapat lebih mudah untuk dimengerti. Penulis menggunakan *compiler* GCC untuk menjalankan algoritma yang telah dibuat. Anda tidak perlu repot-repot untuk mencari compiler C yang satu ini, karena GCC sudah terdapat dalam paket instalasi Linux Anda.

Algoritma berikut ini akan kita buat dalam bentuk suatu fungsi sehingga untuk menggunakannya fungsi tersebut dapat dipanggil dari fungsi main. Tujuan kita membuat algoritma dalam bentuk suatu fungsi yaitu supaya program yang kita buat tidak terlihat rumit, hal itu disebabkan karena program terbagi atas modul-modul terpisah yang sederhana dan mudah untuk dipahami. Oleh karena C merupakan bahasa yang *case sensitive* maka Anda perlu ekstra hati-hati untuk tidak salah menuliskan sintaks program Anda.

Algoritma bilangan prima

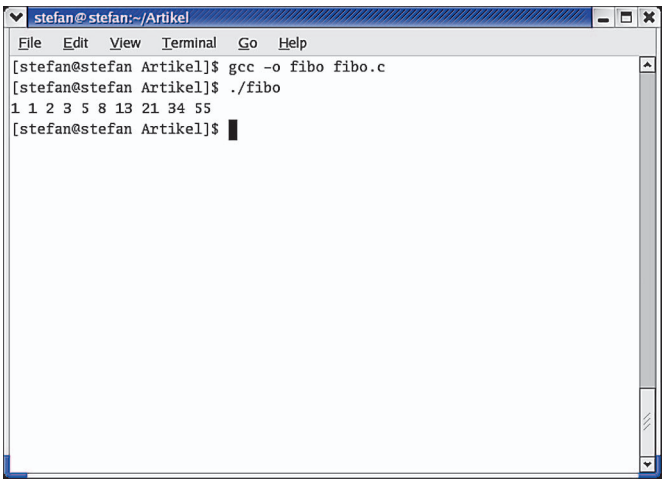
Marilah kita awali dengan algoritma yang pertama yaitu algoritma untuk membentuk fungsi yang dapat memeriksa apakah suatu angka merupakan bilangan prima atau bukan. Kita tahu bahwa bilangan prima merupakan bilangan yang hanya dapat habis dibagi oleh bilangan 1 dan oleh bilangan itu sendiri. Kita akan menyusun algoritma untuk menampilkan bilangan-bilangan tersebut pada terminal. Langkah pertama yang perlu kita lakukan yaitu membuat suatu fungsi yang dapat melaksanakan pengecekan terhadap suatu bilangan.

Cara kerja fungsi yang akan kita buat yaitu dengan memeriksa parameter yang telah ditransfer ke dalam fungsi. Kita namakan fungsi pertama kita ini is_prima. Fungsi tersebut akan mengecek apakah bilangan tersebut hanya habis dibagi oleh bilangan 1 dan oleh bilangan itu sendiri ataukah juga dapat habis dibagi oleh bilangan lain. Fungsi is_prima ini dipanggil oleh fungsi main dan akan mengembalikan nilai 1 (*true*) apabila bilangan yang telah ditransfer ke dalam fungsi merupakan bilangan prima dan akan mengembalikan nilai 0 (*false*) apabila bilangan tersebut bukan merupakan bilangan prima. Nilai 1 dan 0 kita sepakati untuk menunjukkan nilai true atau false, yang merupakan tanda bahwa bilangan tersebut merupakan bilangan prima atau bukan.

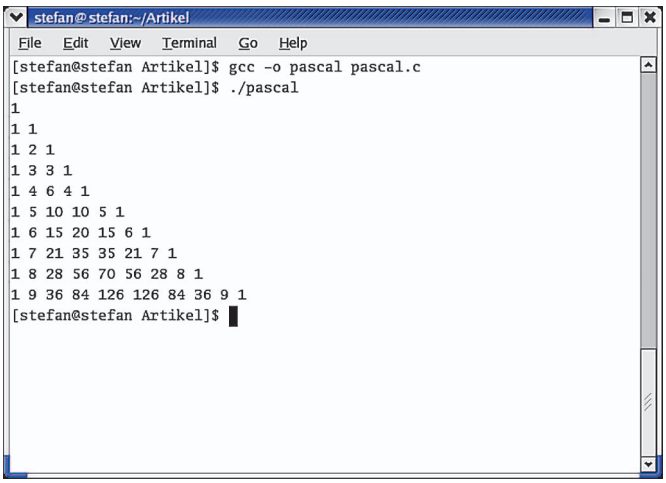
```
/* Source code fungsi is_prima */
int is_prima(int N)
{
  int x, flag;
  switch(N)
  {
    case 1: flag=0; break;
    case 2: flag=1; break;
    default:
      flag=1;
      for(x=2; x<=N-1; x++)
      {
        if((N%x)==0)
        {
          flag=0;
          break;
        }
    }
  }
  return(flag);
}
```

Penjelasan:

Nilai bilangan yang akan diuji diambil oleh fungsi is_prima untuk dilakukan pengecekan. Nilai tersebut diterima oleh fungsi is_prima dan disimpan dalam variabel N yang bertipe integer. Pertama-tama algoritma ini akan memeriksa nilai N menggunakan perintah switch case. Jika nilai N adalah 1 maka variabel flag langsung akan bernilai 0 atau false hal tersebut dikarenakan 1 bukanlah bilangan prima. Jika nilai N bernilai dua maka variabel flag akan diisi dengan 1 atau true karena 2 merupakan bilangan prima yang pertama. Tetapi jika N itu tidak bernilai 1 maupun 2 maka pada baris selanjutnya program akan menguji apakah N habis



fibo.



pascal.

dibagi oleh bilangan di antara 1 sampai bilangan itu sendiri. Sebelum pengujian ini kita telah mengisi nilai awal variabel flag bernilai 1 (*true*). Apabila dalam pengujian ditemukan bahwa bilangan itu bisa dibagi oleh bilangan diantara 1 sampai dengan bilangan itu sendiri maka variabel flag akan bernilai 0 (false) dan pengujian akan langsung berhenti tanpa perlu melanjutkan ke pengujian berikutnya, untuk itu kita menggunakan perintah *break*.

Pengujian dalam program akan berhenti dikarenakan perintah ini. Kita tidak perlu menguji lagi apakah N habis dibagi bilangan selanjutnya. Nilai akhir dari variabel *flag* menunjukkan hasil pengujian. Untuk mengembalikan nilai 1 (true) atau 0 (false) dalam variabel flag yang menunjukkan apakah bilangan tersebut merupakan bilangan prima atau bukan maka kita dapat menggunakan perintah berikut:

```
return(flag);
```

Selanjutnya dalam fungsi main bila kita bermaksud untuk menampilkan bilangan prima dari 1-20 kita dapat memanggil fungsi is_prima yang telah dibuat dengan cara sebagai berikut:

```
int main()
{
  int i, nilai;
  for(i=1; i<=20; i++)
  {
    nilai=is_prima(i);
    if(nilai==1) printf("%d   ", i);
  }
  printf("\n");
  return 0;
}
```

Cukup mudah bukan? Fungsi main ini memanggil fungsi is_prima untuk mengetahui apakah suatu bilangan yang akan dicek merupakan bilangan prima atau bukan. Lalu isi variabel flag yang dihasilkan oleh fungsi is_prima dikembalikan oleh fungsi is_prima ke dalam variabel nilai. Kita dapat mengetahui bilangan tersebut prima atau bukan dengan melihat isi dari variabel nilai. Dalam hal ini saya misalkan variabel nilai sebagai variabel untuk menampung nilai yang dikembalikan oleh fungsi is_prima.

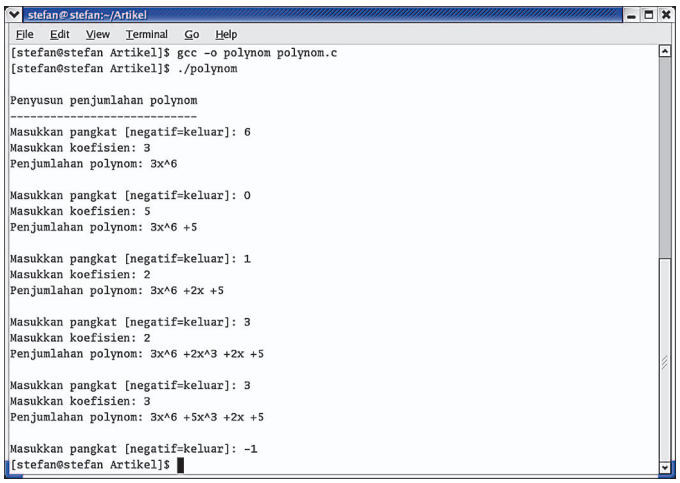
Jika nilai yang dikembalikan oleh fungsi is_prima bernilai 1, maka bilangan yang diuji akan dicetak, sehingga program hanya menampilkan bilangan prima antara dari 1 sampai 20 sesuai dengan keinginan kita. Anda sudah selesai membuat sebuah fungsi is_prima yang dapat dipanggil dan pakai sewaktu-waktu dalam program utama Anda. Saya yakin tanpa kesulitan Anda telah mengerti algoritma bilangan prima di atas. Kita telah selesai membuat algoritma bilangan prima dan akan melanjutkan ke algoritma fibonacci.

Algoritma Fibonacci

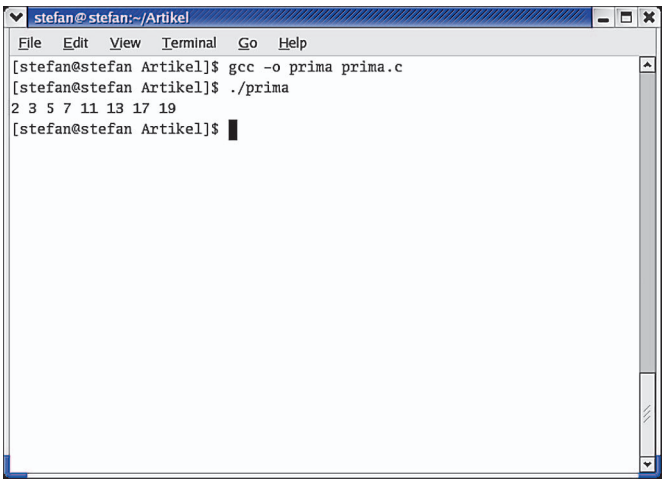
Sekarang kita masuk ke pokok pembahasan yang kedua yaitu algoritma untuk menghasilkan deret fibbonaci. Berikut ini adalah beberapa bilangan fibbonaci yang pertama, yaitu 1 1 2 3 5 8 13 dan seterusnya. Setiap bilangan dalam deret tersebut merupakan hasil penjumlahan dari dua bilangan sebelumnya, deret dimulai dari dua buah bilangan 1. Saya mengasumsikan Anda telah mengerti hubungan masing-masing bilangan di dalam deret tersebut. Setelah Anda bisa membuat algoritma bilangan prima, tentu saja untuk membuat algoritma yang menghasilkan bilangan fibbonaci menggunakan bahasa C bukanlah suatu hal yang sulit bagi Anda. Namun karena kita akan mulai mencoba membuat algoritma program yang bersifat rekursif, maka Anda perlu memahami proses rekursif yang terdapat dalam fungsi ini. Saya akan menguraikan langkah-langkah untuk membuat algoritma tersebut. Fungsi yang akan kita buat seperti biasanya akan dipanggil melalui fungsi main dengan disertai parameter yang serupa dengan algoritma program kita sebelumnya.

```
/* Source code fungsi fibonacci */
long int fib(int N)
{
  if(N==0||N==1) return n;
  else return fib(N-2)+fib(N-1);
}
```

Fungsi fibbonaci yang akan kita buat merupakan fungsi yang bersifat rekursif dikarenakan fungsi tersebut dalam menjalankan prosesnya membutuhkan pemanggilan fungsi yang seperti dirinya sendiri, karena dalam bahasa C sebuah fungsi diperbolehkan untuk memanggil dirinya sendiri maka kita akan memanfaat-



polynom.



prima.

kan hal ini. Sekilas tampak bahwa fungsi ini lebih singkat dari fungsi sebelumnya namun sesungguhnya fungsi ini melaksanakan prosedur yang cukup panjang. Mari kita telusuri jalannya algoritma tersebut!

Bila N bernilai 0 atau 1 maka fungsi akan mengembalikan nilai N itu sendiri, namun apabila variabel N pada fungsi bernilai lebih dari 1 (misalkan N) maka fungsi akan memanggil fungsi fib(N-2) dan fungsi fib(N-1), hasil yang dikembalikan oleh fungsi fib(N-2) akan ditambahkan dengan hasil yang dikembalikan oleh fungsi fib(N-1). Fungsi tersebut pada akhirnya akan mengembalikan nilai:

```
f(N) = f(N-2) + f(N-1);
sedangkan
f(0) = 0;
f(1) = 1;
```

Agar lebih jelasnya, Anda dapat memperhatikan contoh berikut. Misalkan kita mencari bilangan ke-4 dari deret fibonacci maka jalannya fungsi:

1. f(N) = f(N-2) + f(N-1)
2. f(4) = f(2) + f(3)
3. f(4) = {f(0) + f(1)} + {f(1) + f(2)}
4. f(4) = {0 + 1} + {1 + [f(0) + f(1)]}
5. f(4) = {0 + 1} + {1 + [0 + 1]}
6. f(4) = 3

Maka deret fibonacci yang ke-4 bernilai 3.

```
1   1   2   3
            ^
```

Seperti pada fungsi prima, kita juga memanggil fungsi fib melalui fungsi main. Untuk menampilkan 8 bilangan pertama dari deret fibonacci maka dalam fungsi main fungsi fibo dapat dipanggil dengan cara:

```
int main()
{
  int n=10, i;
  clrscr();
  for(i=1; i<=n; i++) printf ("%ld  ",
  fib(i));
  getch();
  return 0;
}
```

Maka program akan mencetak tampilan seperti di bawah ini:

1 1 2 3 5 8 13 21 34

Dengan ini telah selesailah algoritma fibonacci Anda. Saya yakin apabila Anda menyimak dengan baik, Anda dapat mengerti bagaimana jalannya program sehingga bilangan-bilangan tersebut dapat tampil pada layar monitor Anda.

Algoritma segitiga Pascal

Algoritma ketiga berikut ini akan menampilkan baris-baris yang membentuk segitiga pascal yang seringkali diperlukan dalam perhitungan matematika. Cara kerja algoritma tersebut yaitu dengan meng-inputkan jumlah baris dalam segitiga Pascal yang ingin ditampilkan. akan diproses oleh fungsi ini sekaligus menam-pilkannya di layar. Untuk menghasilkan segitiga pascal kali ini kita akan sekalian menggunakan algoritma faktorial dan kombinasi. Untuk melaksanakan proses faktorial nantinya kita dapat dengan mudah memanggil fungsi *fact*, sedangkan untuk melaksanakan fungsi kombinasi, kita dapat memanggil fungsi komb. Kedua fungsi tersebut dapat digunakan untuk menampilkan baris-baris segitiga Pascal di terminal Linux Anda. Fungsi faktorial tersebut sebagai berikut:

```
unsigned long int fact(int f)
{
  if(f<2) return 1; else return f*fact(f-1);
}
```

Seperti fungsi rekursif yang terdapat pada algoritma fibonacci, fungsi ini memfaktorialkan variabel f. Proses yang terjadi dalam fungsi fact bila diuraikan adalah sbb:

1. fact(f) = f*fact(f-1)
2. fact(4) = 4*fact(3)
3. fact(4) = 4*{3*fact(2)}
4. fact(4) = 4*{3*[2*1]}
5. fact(4) = 24

Kemudian kita perlu membuat fungsi untuk perhitungan kombinasi komb(n,r). Fungsi ini sederhana dan tidak sulit, hanya mengembalikan nilai perhitungan kombinasi dengan memanfaatkan fungsi faktorial sebelumnya. Kita hanya perlu mengem-balikan nilai kombinasi menggunakan rumus kombinasi C(n,r) = n! / (r! * (n-r)!).

```
int komb(int n, int r)
{
  return(fact(n)/(fact(r)*fact(n-r)));
}
```

Selanjutnya dalam fungsi main kita mencetak baris-baris segitiga Pascal yang dihasilkan dengan rumus kombinasi:
C(0,0)
C(1,0) C(1,1)
C(2,0) C(2,1) C(2,2)
dst...

Lalu setelah kedua fungsi tersebut beres, kita panggil fungsi komb ke dalam fungsi utama. Untuk mencetak 10 baris segitiga pascal, seperti biasa kita tulis fungsi utama kita yaitu main untuk memanggil fungsi kombinasi sebagai berikut:

```
int main()
{
  int i, j, n=10;
  for(i=0; i<n; i++)
  {
    for(j=0;j<=i;j++)
    {
      printf("%d ",komb(i,j));
    }
    printf("\n");
  }
  return 0;
}
```

Output dari program segitiga Pascal ini adalah sebagai berikut:
1
11
121
1331
dst...

Anda telah menyelesaikan 3 algoritma yang pertama, sekarang Anda dapat melanjutkan langkah Anda menuju algoritma yang lebih menantang.

Algoritma penjumlahan Polinom

Algoritma yang terakhir yaitu algoritma untuk menjumlahkan bilangan-bilangan polinom. Algoritma yang akan kita buat ini menggunakan *single linked list*. Saya mengasumsikan anda sebelumnya telah mengetahui konsep penggunaan linked list dalam bahasa C. Dalam fungsi tersebut kita akan menampilkan penjumlahan polinom yang diinputkan oleh *user*.

Kita memerlukan file include stdio.h untuk menjalankan algoritma ini. Kemudian untuk mempersiapkan linked list kita membuat sebuah *struct* yang saya misalkan diberi nama *tnode*.

```
#include<stdio.h>
struct tnode
{
  int pangkat, koef;
  struct tnode *next;
};
```

Lalu setelah struct dibuat, di bawahnya kita deklarasikan variabel pointer global yang akan digunakan dalam linked list sebagai node (data baru), head (awal linked list), dan curr (penunjuk node aktif).

```
struct tnode *node, *head=NULL, *curr;
```

Supaya algoritma yang kita buat lebih mudah dipahami maka kita akam membuat beberapa fungsi, yaitu fungsi 'tambah' untuk meminta masukan data baru, fungsi 'linking' untuk menempatkan data baru ke dalam linked list pada posisi yang tepat, dan fungsi 'tampil' untuk menampilkan data yang terdapat dalam linked list. Fungsi linking merupakan fungsi yang cukup rumit bila dibandingkan dengan fungsi tambah dan fungsi tampil. Namun apabila Anda mau sedikit berkonsentrasi saya yakin Anda dapat menangkap logika algoritma tersebut.

Kita mulai dengan fungsi yang pertama yaitu fungsi 'tambah'. Fungsi ini akan dipanggil dari fungsi main untuk meminta data baru yang akan dimasukkan dalam linked list.

```
int tambah()
{
  node=(struct
  tnode*)malloc(sizeof(struct tnode));
  printf("Masukkan pangkat
  [negatif=keluar]: ");
  scanf("%d",&node->pangkat);
  if(node->pangkat<0)
  {
    curr=head;
    while(curr!=NULL)
    {
      node=curr;
      curr=curr->next;
      free(node);
    }
    exit(0);
  }
  printf("Masukkan koefisien: ");
  scanf("%d",&node->koef);
  return 0;
}
```

Fungsi di atas akan digunakan berulang-ulang untuk meminta masukan data baru. Setiap kali fungsi dipanggil, kita harus memesan tempat pada memory komputer untuk menampung data baru. Dalam hal ini alamat memory yang dipesan disimpan dalam variabel pointer node, program akan meminta kita untuk memasukkan data baru yang akan dimasukkan ke memory melalui variabel pointer node.

Kita membuat pengecekan terhadap nilai node->pangkat, apabila pangkat diisi dengan bilangan negatif, program akan berakhir dengan dieksekusinya perintah exit(0). Sebelum perintah exit(0), program harus melakukan pembebasan alokasi memori yang telah dipakai program. Hal ini penting sekali untuk dilakukan karena tanpa disertai oleh pembebasan memory, maka alamat memory yang telah kita pakai tidak dapat digunakan oleh program lain yang dapat menyebabkan terjadinya error karena komputer kehabisan memory.

Berikutnya kita akan membuat fungsi untuk memasukkan data yang telah didapat dari fungsi tambah() ke dalam linked list. Kita tidak bisa sembarangan memasukkan data ini ke dalam linked list. Kita perlu melakukan pengecekan untuk menemukan posisi yang tepat di mana data akan diletakkan dalam linked list, sehingga data yang kita peroleh merupakan data dalam posisi terurut secara menurun (*descending*) berdasarkan pangkat.

```
int linking()
{
  if(head==NULL) //belum ada data
  {
    head=node;
    curr=head;
    head->next=NULL;
  }
  else //sudah ada data
  {
    curr=head;
    while((curr->pangkat > node-
    >pangkat) && curr!=NULL)
      curr=curr->next;
    if((curr->pangkat==node->pangkat)
```

```
    && curr!=NULL)
      curr->koef+=node->koef;
    else if(curr==head) //data sebagai
    head
    {
      node->next=curr;
      head=node;
    }
    else if(curr!=head && curr!=NULL) //
    sebagaidata diantara
    {
      node->next=curr;
      curr=head;
      while(curr->next!=node->next)
      curr=curr->next;
      curr->next=node;
    }
    else if(curr==NULL) //sebagai data
    terakhir
    {
      curr=head;
      while(curr->next!=NULL) curr=curr-
      >next;
      curr->next=node;
      node->next=NULL;
    }
  }
  return 0;
}
```

Untuk memasukkan data ke dalam linked list tidak semua data diperlakukan sama. Karena kita ingin memasukkan data sambil mengurutkan, maka kita perlu memeriksa pada posisi mana node akan dimasukkan ke dalam linked list.

Pertama-tama kita harus memeriksa apakah pada linked list sudah terdapat data dengan fungsi if(head == NULL). Jika belum ada data pada linked list maka data baru dianggap sebagai data pertama sekaligus data terakhir. Oleh karena itu, pointer head menunjuk pada node dan head->next menunjukk ke NULL (akhir data).

Lain halnya apabila sudah terdapat data pada linked list, kita masih perlu melakukan pemeriksaan apakah data baru akan disisipkan sebagai data pertama, di antara atau sebagai data terakhir. Namun, sebelumnya kita perlu melakukan pengecekan terhadap pangkat pada data baru. Apabila dalam linked list sudah terdapat data dengan pangkat yang sama maka kita hanya perlu menambahkan koefisien data baru dengan koefisien data yang sudah terdapat dalam linked list.

```
if(curr->pangkat == node->pangkat)
  ... /* tambahkan koefisien */
else if((curr==head) && (curr->pangkat
< node->pangkat))
  ... /* data pertama */
else if(curr->next != NULL)
  ... /* data di antara */
else
  ... /* 2 data terakhir */
```

Sedangkan untuk menampilkan data, kita perlu membuat sebuah fungsi lagi. Kita akan membuat fungsi tampil() untuk menampilkan data yang terdapat dalam linked list.

```
int tampil()
{
  printf("Penjumlahan polynom: ");
  curr=head;
  while(curr!=NULL)
  {
    if(curr==head) //jika pointer curr pada
    awal data
    {
      if(curr->pangkat==0) printf("%d
      ",curr->koef);
      else if(curr->pangkat==1)
      printf("%dx ",curr->koef);
      else printf("%dx^%d ",curr-
      >koef,curr->pangkat);
    }
    else //jika pointer curr bukan pada awal
    data
    {
      if(curr->pangkat==0) printf("%+d
      ", curr->koef);
      else if(curr->pangkat==1)
      printf("%+dx ",curr->koef);
      else printf("%+dx^%d ",curr-
      >koef,curr->pangkat);
    }
    curr=curr->next;
  }
  printf("\n\n");
  return 0;
}
```

Untuk mencetak data pada linked list kita membedakan cara pencetakan *data head* (awal data) dengan data berikutnya. Dalam mencetak data pertama apabila koefisien bernilai positif kita tidak perlu mencetak tanda tambah di depannya, sedangkan pada data seterusnya tanda plus maupun minus harus ditampilkan. Bagian yang akan mencetak awal data yaitu bagian if(curr==head) sedangkan bagian yang akan mencetak data selain head ditempatkan pada bagian else. Tanda (+) dan (-) pada proses pencetakan data selain head digunakan untuk menampilkan tanda plus maupun minus sebelum koefisien ditampilkan.

Untuk membentuk algoritma penjumlahan polynom semua fungsi diatas kita satukan dalam fungsi main sebagai berikut:

```
int main()
{
  printf("\nPenyusun penjumlahan
  polynom\n");
  printf("----------------\n");
  do{
    tambah();
    linking();
    tampil();
  }while(1);
  return 0;
}
```

statement while(1) digunakan untuk melakukan pengulangan tanpa akhir (*looping forever*). Seperti yang telah kita lihat pada fungsi tambah, program dapat dihentikan apabila pangkat yang dimasukkan dalam fungsi tambah bernilai negatif.

Sebagai catatan, untuk mengkompilasi program yang sudah anda buat, pada terminal gunakan perintah:

```
gcc -o namafile namafile.c
```

Perhatikan bahwa file harus berada pada direktori yang sedang aktif. Dengan perintah tersebut GCC akan mengompilasi file Anda sehingga dapat dieksekusi secara langsung dengan mengetikkan

```
./namafile
```

untuk menjalankan program yang telah dikompilasi dengan gcc.

Demikianlah pembahasan kita tentang algoritma matematika. Saya sangat berharap apa yang telah kita bahas di atas dapat berguna untuk Anda. Akhir kata saya ucapkan selamat mencoba!

Stefan *(stefan_ay@plasa.com)*