Pengantar Struktur Data & Algoritma

Oleh :

Entin Martiana

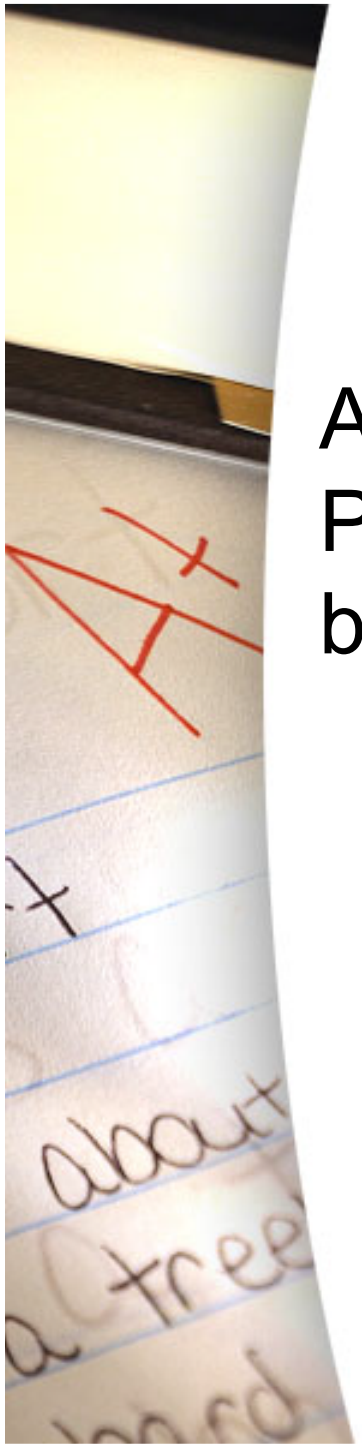
Definisi

Struktur Data:

Skema organisasi, seperti struktur dan array, yang diterapkan pada data sehingga data dapat diinterprestasikan dan sehingga operasi-operasi spesifik dapat dilaksanakan pada data tersebut

Algoritma:

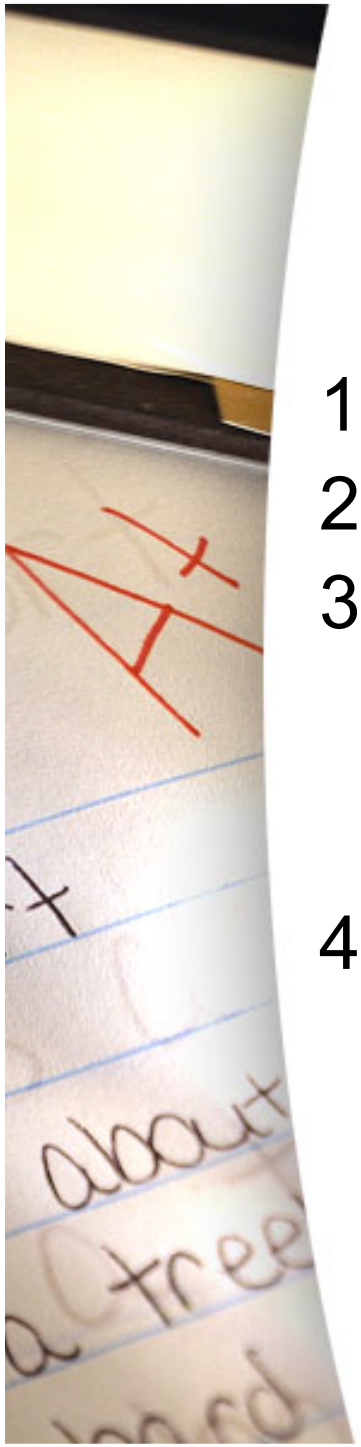
- Urutan dari instruksi-instruksi untuk menyelesaikan permasalahan.
- Urutan langkah-langkah yang merubah masukan menjadi keluaran.



Contoh Algoritma

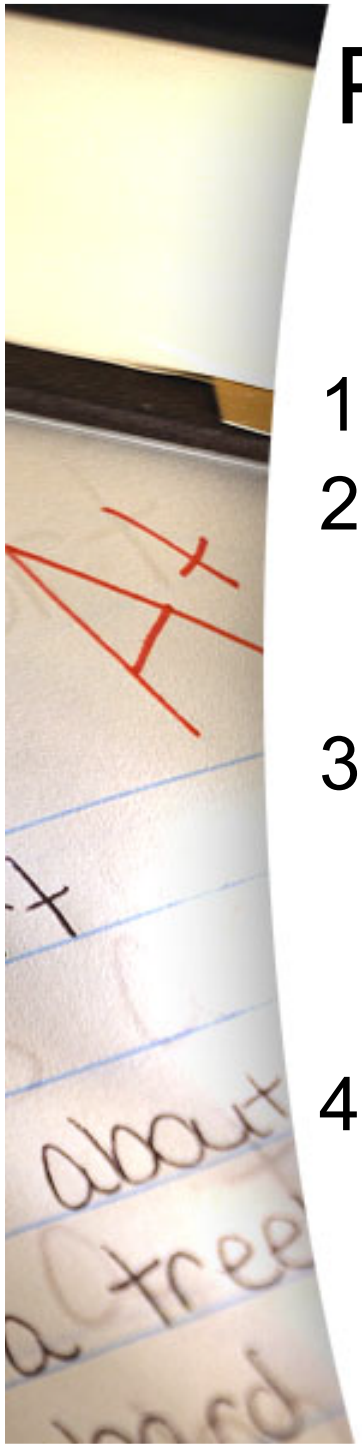
Algoritma Pencarian Faktor Persekutuan Terbesar dua buah bilangan:

- Prosedur Sekolah Menengah
- Pengecekan Integer Berurutan
- ALgoritma Euclidean



Prosedur Sekolah Menengah (m,n)

1. Definisikan faktor prima dari m
2. Definisikan faktor prima dari n
3. Definisikan faktor prima yang sama pada langkah 1 dan 2. Identifikasi p_m & p_n sebagai frekuensi terjadinya bilangan prima tsb pada m dan n
4. Hitung hasil perkalian dari bilangan prima persekutuan dari langkah 3 dengan frekuensi min sebagai hasil bilangan persekutuan terbesar.

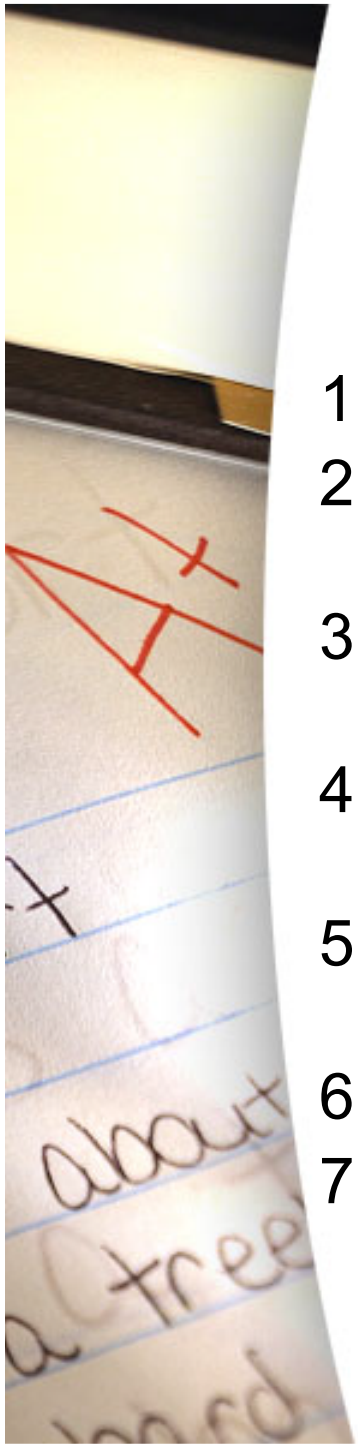


Pengecekan Integer Berurutan (m,n)

1. Masukkan nilai minimum dari {m,n} ke t.
2. Bagi m dengan t. Jika sisa hasil bagi adalah 0, menuju langkah 3; selainnya itu ke langkah 4.
3. Bagi n dengan t. Jika sisa hasil bagi adalah 0, maka kembalikan t sebagai hasil dari faktor persekutuan terbesar dan stop. Jika tidak ke langkah 4.
4. Kurangi nilai t dengan 1. Kembali ke langkah 2.

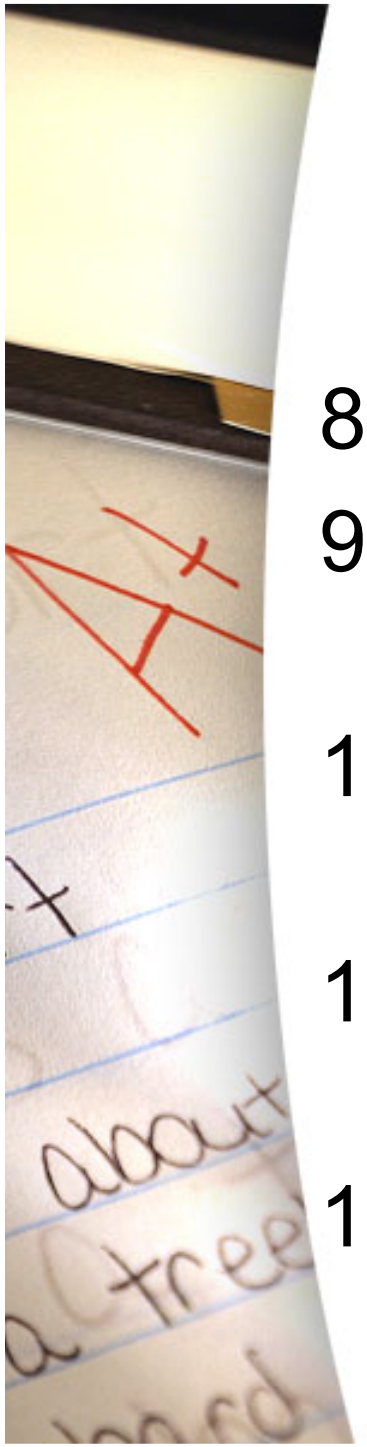
Algoritma Euclidean (m,n)

1. Jika n=0, kembalikan nilai m sebagai hasil dan stop, selainnya itu proses langkah 2
2. Bagi m dengan n dan masukkan nilai bagi hasil ke r.
3. Masukkkan nilai n ke m dan nilai r ke n. Kembali ke langkah 1.



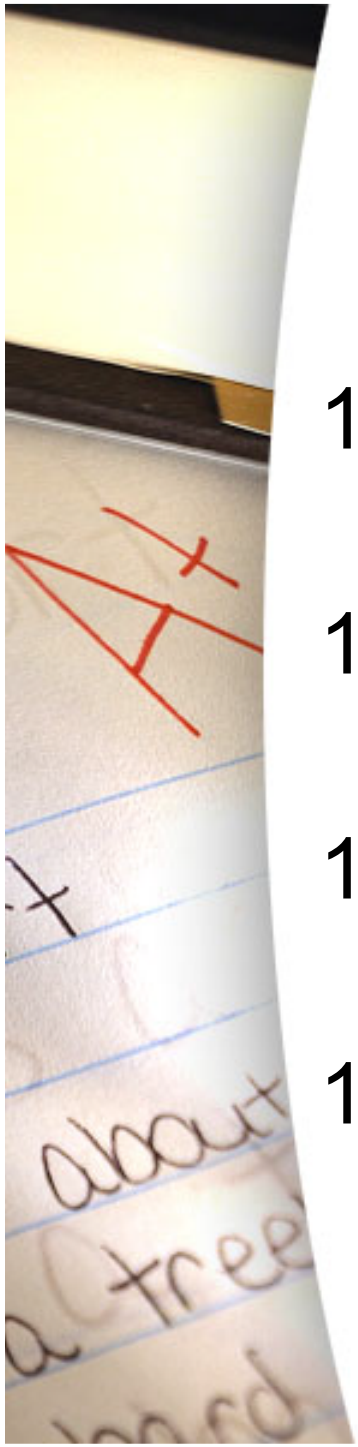
Materi

1. Review (Array, Pointer, Struktur)
2. Single Linked List (Membuat, Tampil. Cari, Hapus)
3. Single Linked List (Sisip di akhir, Sisip sebelum, Sisip sesudah simpul tertentu)
4. Double Linked List (Membuat, Tampil. Cari, Hapus)
5. Double Linked List (Sisip di akhir, Sisip sebelum, Sisip sesudah simpul tertentu)
6. Stack (Tumpukan)
7. Queue (Antrian)



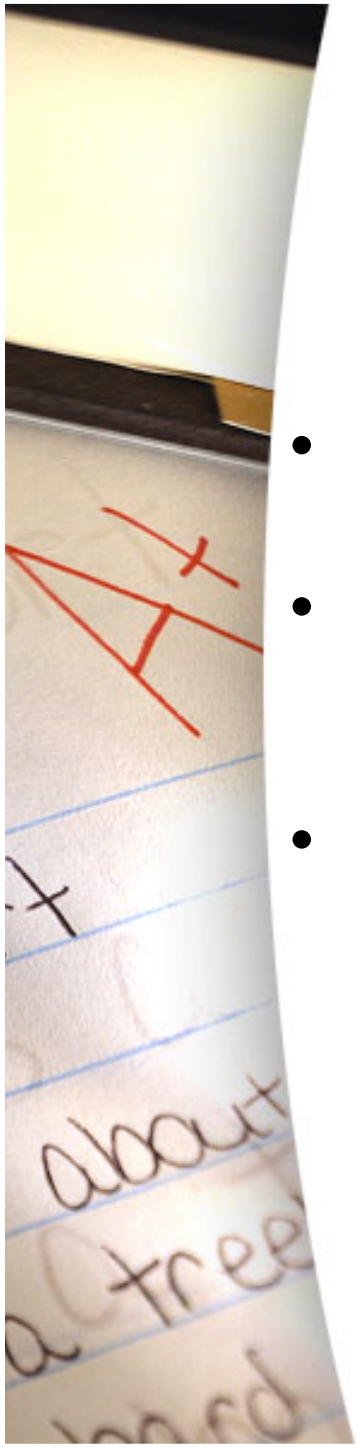
Materi

8. Rekursi
9. Sorting (Pengurutan) – Insertion, Selection
10. Sorting (Pengurutan) – Bubble, Shell
11. Sorting (Pengurutan) – Quick, Merge
12. Searching (Pencarian)



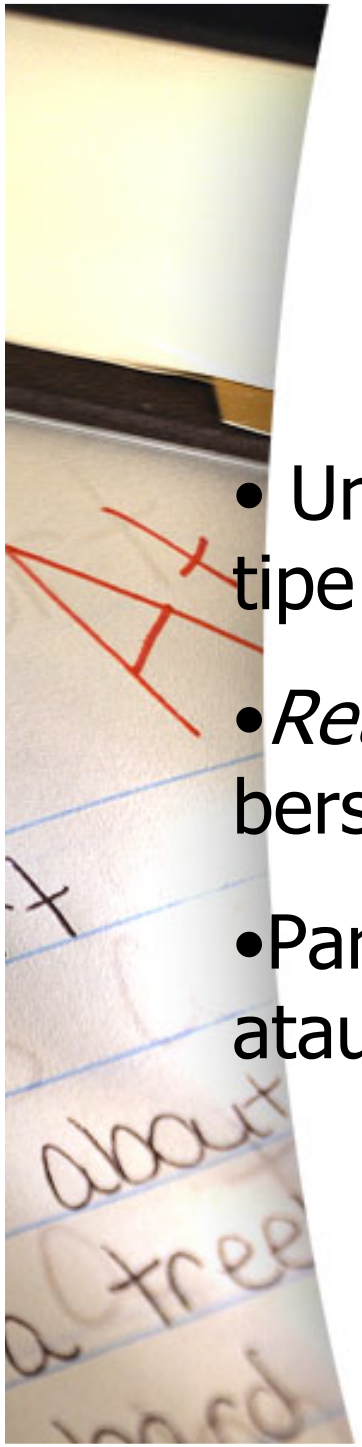
Materi

13. Graph – Konsep, Jenis Graph, Representasi dengan Array
14. Graph – Representasi dengan Linked List
15. Tree – Konsep Tree, Membangun Binary Tree, Metode Traversal
16. Tree – Spanning Tree



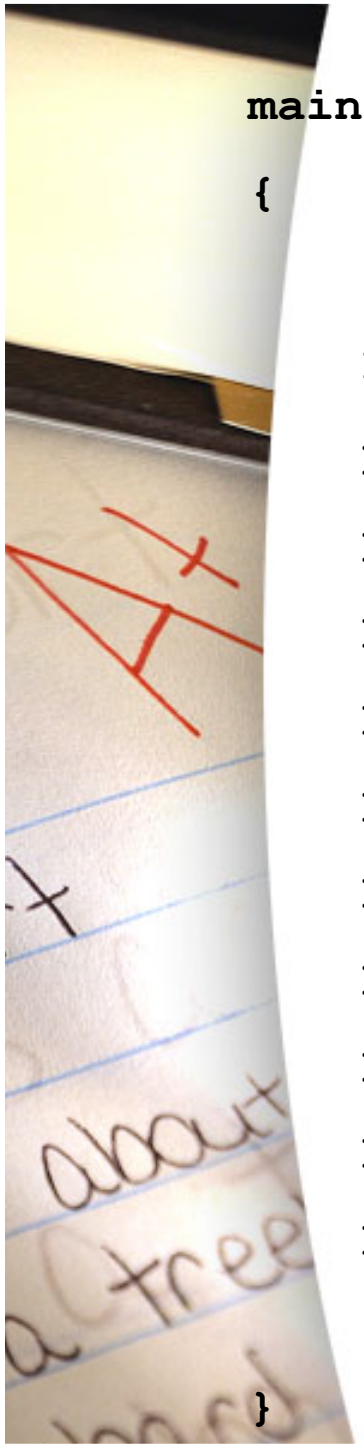
Latar Belakang

- Tipe data array cenderung menggunakan ukuran yang tetap
- Pada beberapa kasus, ukuran dari sebuah obyek tidak bisa dipastikan sampai dengan waktu dieksekusi *(run time)*
- Alokasi memori (memory allocation) menyediakan fasilitas untuk membuat ukuran buffer dan array secara dinamik. Dinamik artinya bahwa ruang dalam memori akan dialokasikan ketika program dieksekusi.



sizeof()

- Untuk mendapatkan ukuran dari berbagai tipe data, variabel ataupun struktur.
- *Return value* : ukuran dari obyek yang bersangkutan dalam byte.
- Parameter dari sizeof() : sebuah obyek atau sebuah tipe data



```
typedef struct employee_st {
    char name[40];
    int id;
} Employee;
```

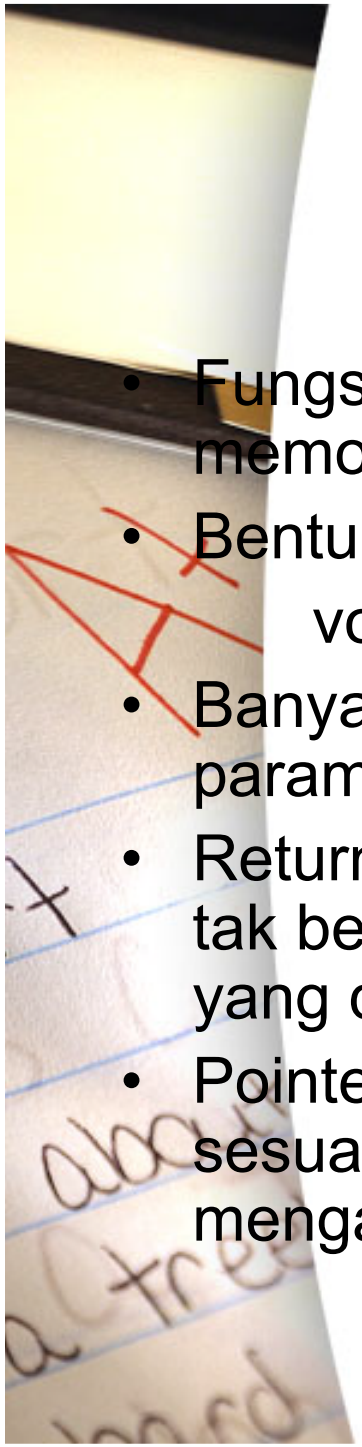
```
main()
{
    int myInt;
    Employee john;
    printf("Size of int is %d\n",sizeof(myInt));
    printf("Size of int is %d\n",sizeof(int));
    printf("Size of Employee is %d\n",sizeof(Employee));
    printf("Size of john is %d\n",sizeof(john));
    printf("Size of char is %d\n",sizeof(char));
    printf("Size of short is %d\n",sizeof(short));
    printf("Size of int is %d\n",sizeof(int));
    printf("Size of long is %d\n",sizeof(long));
    printf("Size of float is %d\n",sizeof(float));
    printf("Size of double is %d\n",sizeof(double));
    return 0;
}
```

Hasil Program

lesson12

Auto

```
Size of int is 4
Size of int is 4
Size of Employee is 44
Size of john is 44
Size of char is 1
Size of short is 2
Size of int is 4
Size of long is 4
Size of float is 4
Size of double is 8
Press any key to continue_
```



malloc()

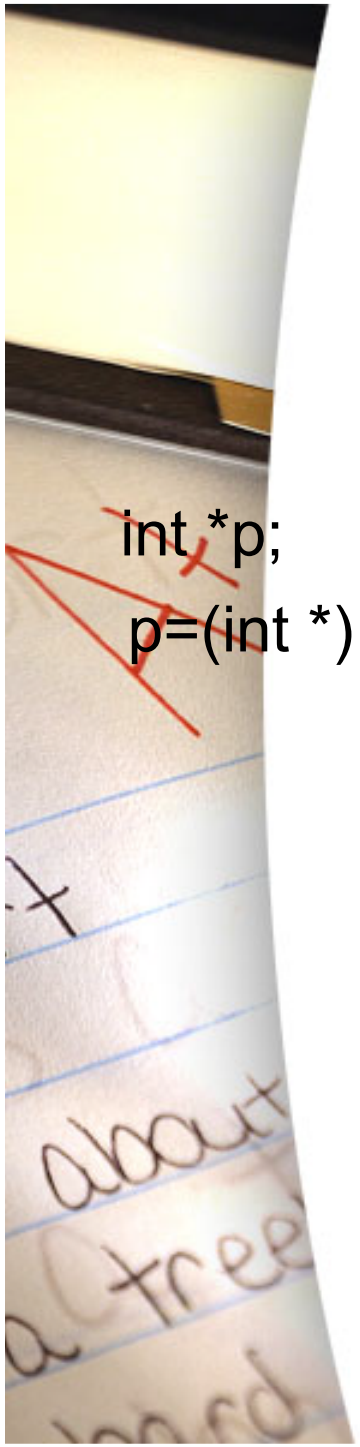
- Fungsi standar yang digunakan untuk mengalokasikan memori
- Bentuk prototypenya adalah

  void *malloc(int jml_byte)
- Banyaknya byte yang akan dipesan dinyatakan sebagai parameter fungsi.
- Return value dari fungsi ini adalah sebuah pointer yang tak bertipe *(pointer to void)* yang menunjuk ke buffer yang dialokasikan.
- Pointer tersebut haruslah dikonversi kepada tipe yang sesuai (dengan menggunakan *type cast*) agar bisa mengakses data yang disimpan dalam buffer.

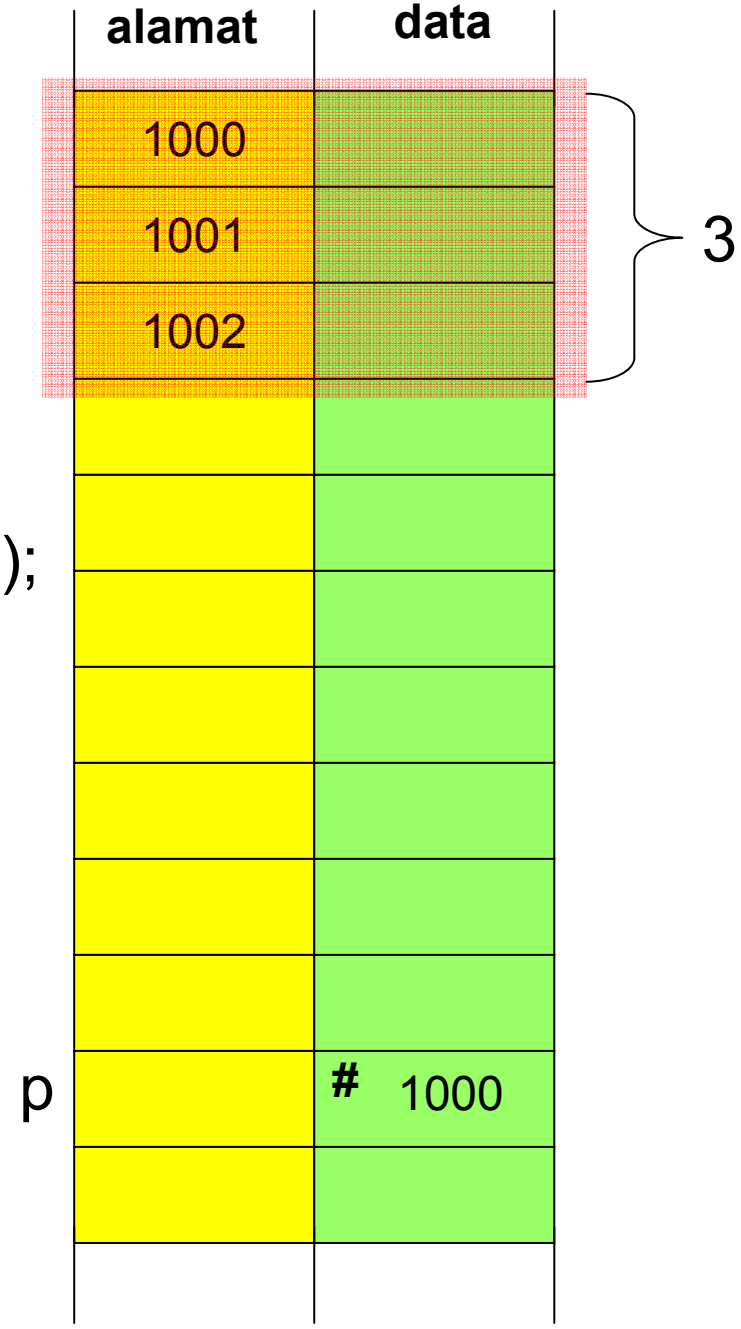
malloc()

- Jika proses alokasi gagal dilakukan, fungsi ini akan memberikan return value berupa sebuah pointer NULL.
- Sebelum dilakukan proses lebih lanjut, perlu terlebih dahulu dipastikan keberhasilan proses pemesanan memori, sebagaimana contoh berikut:

```
int *x;
x = (int *) malloc(3 * sizeof(int));
if (x== NULL) {
    printf("Error on malloc\n");
    exit(0);
} else {
    lakukan operasi memori dinamis…
}
```



```
int *p;
p=(int *) malloc(3 * sizeof(int));
```



Membebaskan kembali memori dengan fungsi free()

Jika bekerja dengan menggunakan memori yang dialokasikan secara dinamis, maka seorang programmer haruslah membebaskan kembali memori yang telah selesai digunakan untuk dikembalikan kepada sistem. Setelah suatu ruang memori dibebaskan, ruang tersebut bisa dipakai lagi untuk alokasi variabel dinamis lainnya. Untuk itu digunakan fungsi `free()` dengan prototype sebagai berikut :

```
void free(void *pblok);
```

dengan pblok adalah pointer yang menunjuk ke memori yang akan dibebaskan.

Membebaskan kembali memori dengan fungsi free()

```
/* File program : alokasi2.cMenggunakan fungsi free() utk
   membebaskan kembali memori */
#include <stdio.h>
#include <stdlib.h>
main()
{
   char *pblok;
   pblok = (char *) malloc(500 * sizeof(char));

   if (pblok == NULL)
        puts("Error on malloc");
   else {
        puts("OK, alokasi memori sudah dilakukan");
        puts("------");
        free(pblok);
        puts("Blok memori telah dibebaskan kembali");
   }
 }
```

Mengalokasikan ulang memori dengan fungsi realloc()

- Bisa jadi terjadi ketika hendak mengalokasikan memori, user tidak yakin berapa besar lokasi yang dibutuhkannya. Misalnya user tersebut memesan 500 lokasi, ternyata setelah proses pemasukan data kebutuhannya melebihi 500 lokasi menjadi 600.
- Maka user tersebut dapat mengalokasikan ulang memori yang dipesannya dengan menggunakan fungsi realloc().
- Fungsi ini akan mengalokasikan kembali pointer yang sebelumnya telah diatur untuk menunjuk sejumlah lokasi, memberinya ukuran yang baru (bisa jadi lebih kecil atau lebih besar).

Mengalokasikan ulang memori dengan fungsi realloc()

- Sebagai contoh, adalah pblok adalah pointer yang menunjuk kepada 500 lokasi char, maka user bisa mengalokasikan ulang agar pointer pblok menunjuk kepada 600 lokasi char sebagai berikut :

```
...
pblok = (char *) malloc(500 *
sizeof(char));
...
pblok = realloc(pblok, 600 *
sizeof(char));
```