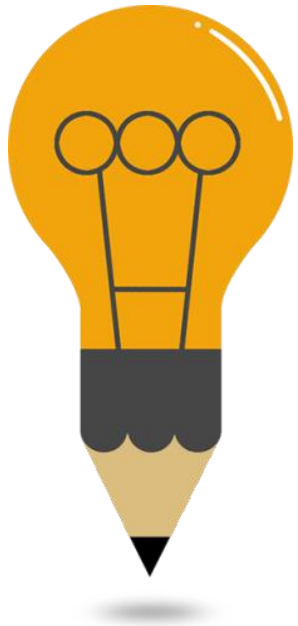
STRUKTUR DATA

Pertemuan 4

Ratih Ngestrini, Nori Wilantika

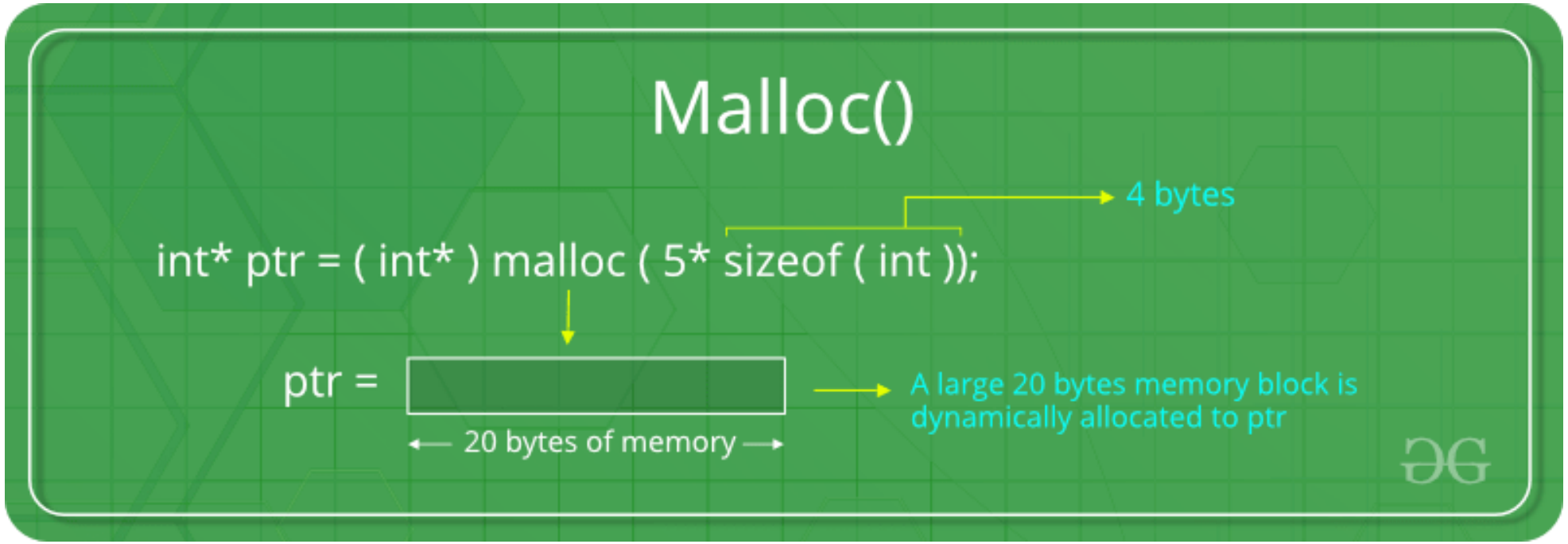
Agenda Pertemuan



1. Review Alokasi Memori Dinamis
2. Pengenalan Single Linked List
3. Pengenalan Double Linked List

REVIEW ALOKASI MEMORI DINAMIS

Fungsi `malloc()`



Fungsi **`malloc`** akan mengalokasi memory sebesar 5 x 4 byte = 20 byte, karena akan kita isi memory tersebut dengan integer, maka kita konversi dengan syntax **`(int*)`**

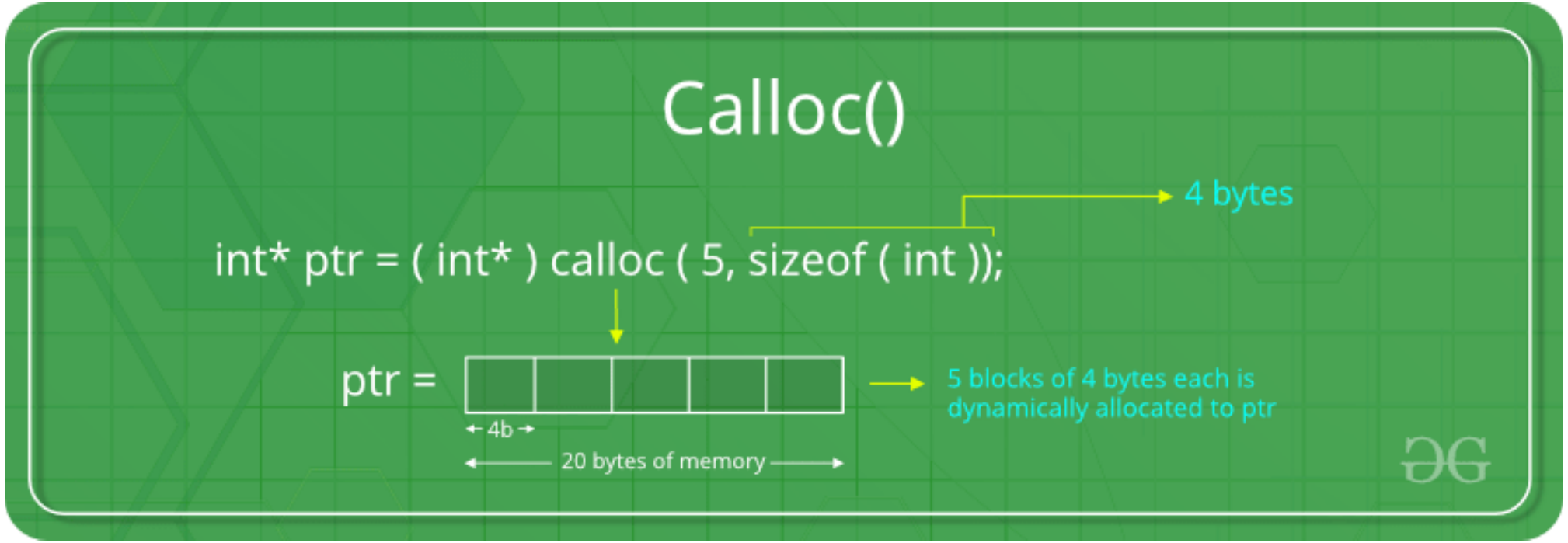
Hasilnya adalah pointer **`ptr`** yang berisi alamat byte pertama dari memory yang dialokasikan

Fungsi `calloc()`

- **`calloc`** atau *"contiguous allocation"* digunakan untuk alokasi memory dinamis seperti `malloc`
- Sama seperti `malloc`, `calloc` juga return pointer bertipe void (**`void*`**)

`malloc()`	**`calloc()`**
1 parameter = ukuran	2 parameter = jumlah blok dan ukuran masing-masing blok
Isi/nilai dari blok memory yang dialokasikan belum terinisialisasi (belum ada nilainya)	Masing-masing blok memory telah terinisialisasi dengan 0 (nol)
malloc lebih cepat dibanding calloc (tentu saja karena selain mengalokasikan, calloc juga menginisialisasi nilai 0 ke setiap blok)	
	Mengapa calloc? menghindari buffer overflow (ketika kita alokasi memory, bisa saja alokasi memory kita sukses, tetapi sebenarnya memory fisik tidak cukup, seperti pada linux yang menerapkan Optimistic Memory Allocation), sehingga dengan kita inisialisasi 0 maka memastikan bahwa memory benar-benar tersedia. buffer overflow bisa menyebabkan crash program, karena ketika kita mau mengisi blok memory, jika tidak cukup, maka akan disimpan di memory yang berdekatan (meluap), bisa saja sedang dipakai program lain.

Fungsi `calloc()`



Fungsi **`calloc`** akan mengalokasi memory sebesar 5 blok integer masing-masing berukuran **`sizeof(int)`** yaitu 4 byte = total 20 byte, karena akan kita isi memory tersebut dengan integer, maka kita konversi dengan syntax **`(int*)`**. Kemudian, masing-masing blok tersebut akan otomatis terinisialisasi dengan nilai 0 (nol).

Fungsi **`realloc()`**

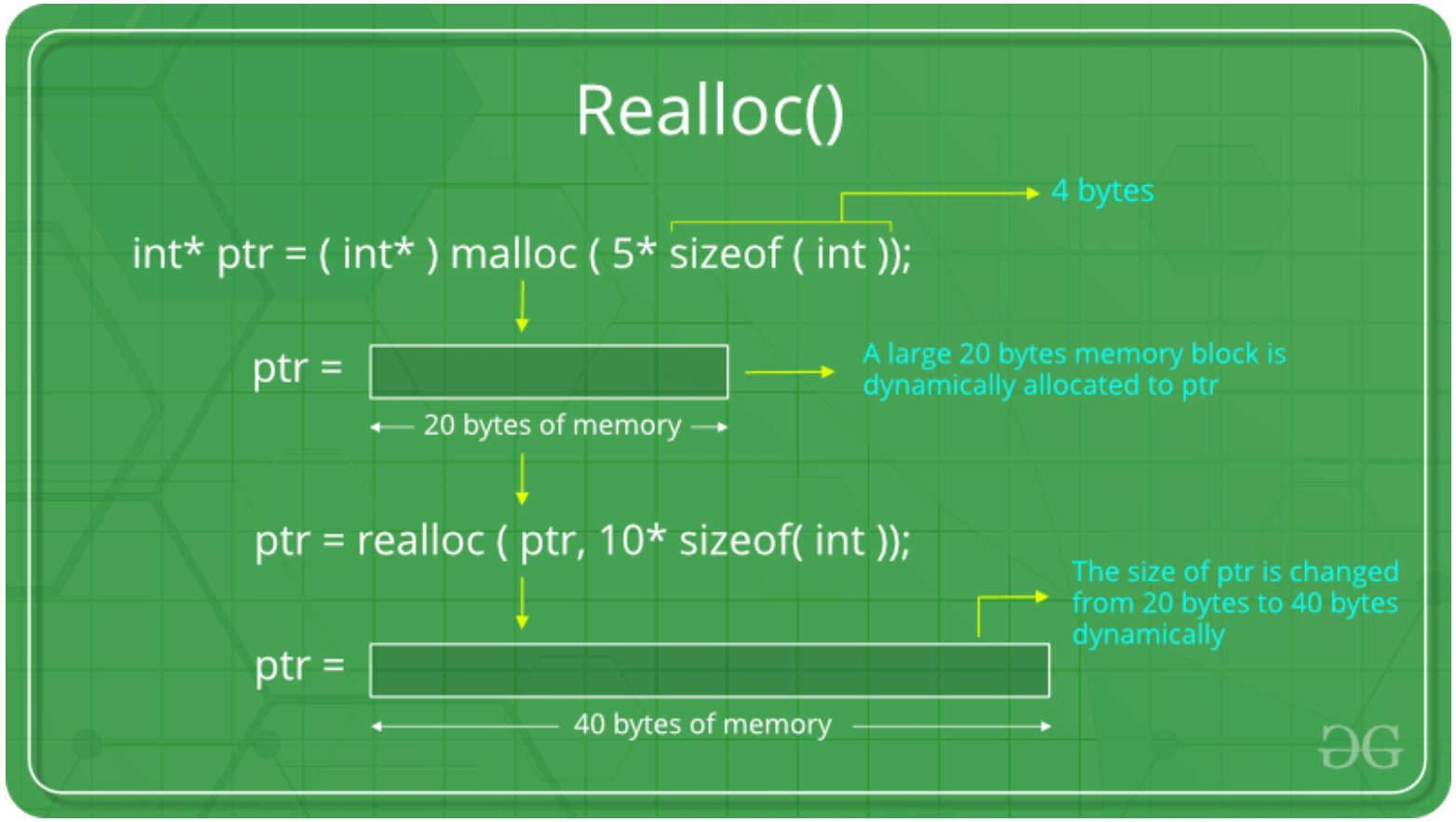
- **`realloc`** atau *"re-allocation"* digunakan untuk mengubah ukuran memori yang dialokasikan fungsi `malloc` dan `calloc`
- Jika memory yang sebelumnya dialokasikan tidak cukup/berlebih, `realloc` dapat digunakan untuk merealokasi memory secara dinamis
- **Jika berhasil**, `realloc` akan melakukan relokasi memory
- **Jika gagal**, fungsi akan return sebuah pointer NULL

- **`ptr = realloc(ptr, ukuran_baru)`**
  - Di mana **`ptr`** adalah pointer dari return fungsi **`malloc`** atau **`calloc`**

- Contoh:
- **`ptr = realloc(ptr, 10 * sizeof(int));`**

  → alokasi memory **`ptr`** dari **`malloc`** atau **`calloc`** akan diubah menjadi sebesar 40 byte

Fungsi `realloc()`



Misal kita punya pointer **`ptr`** hasil dari mengalokasikan memory menggunakan **`malloc`** sebesar 5 x 4 = 20 byte.

Ternyata alokasi tidak cukup, maka perlu realokasi memory (dalam hal ini menambahkan) secara dinamis. Kita ubah alokasi memory **`ptr`** dari 20 byte menjadi 10 x 4 = 40 byte.

Contoh Penggunaan `malloc()` untuk Membuat Array Dinamis

```c
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int i, n;
    printf("Enter number of elements: ");
    scanf("%d", &n);
    int* arr = (int*)malloc(n * sizeof(int));

    if (arr == NULL) {
        printf("Memory not allocated.\n");
        exit(0);
    }
    else {
        printf("Memory successfully allocated using malloc.\n");
        for (i = 0; i < n; ++i) {
            arr[i] = i + 1;
        }
        printf("The elements of the array are: ");
        for (i = 0; i < n; ++i) {
            printf("%d, ", arr[i]);
        }
    }
    return 0;
}
```

Hasil:

```
Enter number of elements: 5
Memory successfully allocated using malloc.
The elements of the array are: 1, 2, 3, 4, 5,
```

Setelah memory dialokasikan, space tersebut dapat diakses sebagai array 1 dimensi.

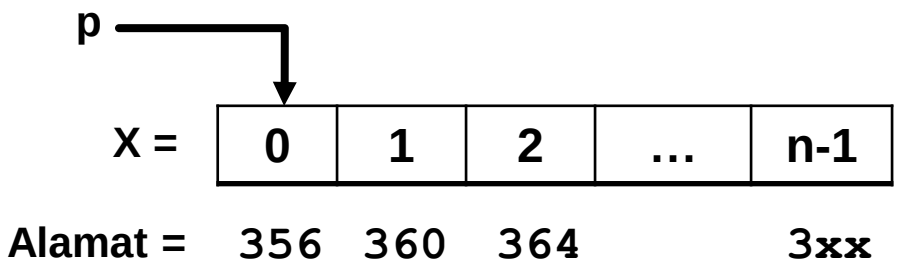
Contoh:

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

Inisialisasi nilai ke memory yg dialokasikan dengan cara:

```c
*p = 34;          atau    p[0] = 34;
*(p+1) = 23;              p[1] = 23;
*(p+2) = 10;              p[2] = 10;
```

Nama array adalah pointer yang menunjuk ke elemen ke 0 dari array.



- **<u>Untuk menampilkan nilai setiap elemen dalam array</u>**:

  Elemen ke 1 : x[0] atau *x atau *(x+0) atau *p atau *(p+0)

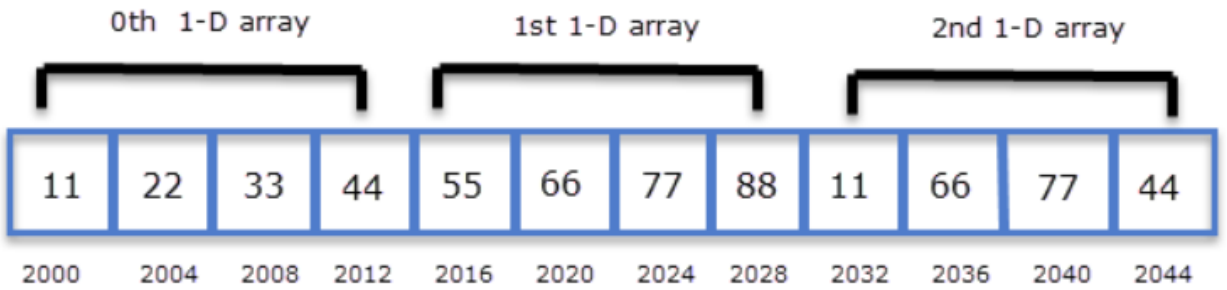
  Elemen ke 2 : x[1] atau *(x+1) atau *(p+1)

  Elemen ke 3 : x[2] atau *(x+2) atau *(p+2)

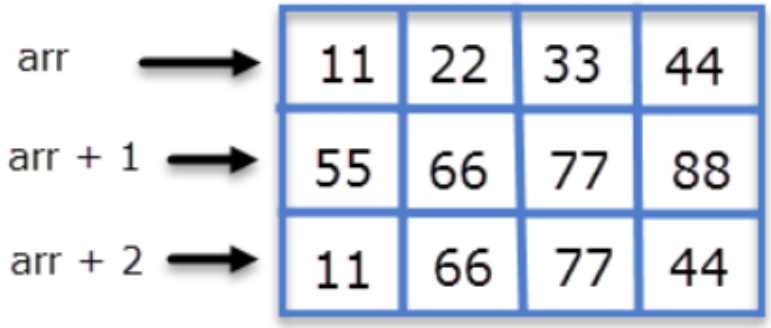
  Elemen ke n : x[n-1] atau *(x+(n-1)) atau *(p+(n-1))

- Bagaimana dengan Array 2 Dimensi?

- Saat membahas array, kita menggunakan istilah seperti baris dan kolom. Konsep ini hanya teoritis, karena memori komputer linier dan tidak ada baris dan kolom.
- Gambar berikut menunjukkan bagaimana array 2-D disimpan dalam memori:



- Array 2-D sebenarnya adalah array 1-D di mana setiap elemen itu sendiri adalah array 1-D.



```c
#include<stdio.h>

int main()
{
    int arr[3][4] = {
                        {11,22,33,44},
                        {55,66,77,88},
                        {11,66,77,44}
                    };

    int i, j;

    for(i = 0; i < 3; i++)
    {
        printf("Address of %d th array %u \n",i , (arr + i));
        for(j = 0; j < 4; j++)
        {
            printf("arr[%d][%d]=%d\n", i, j, *( *(arr + i) + j) );
        }
        printf("\n\n");
    }

    return 0;
}
```

Mengakses elemen array 2D melalui pointer.

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int row = 3, col = 4, i, j, count;

    int* arr[row];
    for (i = 0; i < row; i++)
        arr[i] = (int*)malloc(col * sizeof(int));

    // Putting 1 to 12 in the array
    count = 0;
    for (i = 0; i < row; i++)
        for (j = 0; j < col; j++)
            arr[i][j] = ++count; // Or *(*(arr+i)+j) = ++count

    //Accessing the array values
    for(i = 0; i < 3; i++)
    {
        printf("Address of %d th array %u \n",i , (arr + i));
        for(j = 0; j < 4; j++)
        {
             printf("arr[%d][%d]=%d\n", i, j, *( *(arr + i) + j)); //or arr[i][j]
        }
        printf("\n\n");
    }

    /* free the dynamically allocated memory */
        for (int i = 0; i < row; i++)
            free(arr[i]);

        return 0;
    }
```

Mengakses elemen array 2D melalui pointer.

Penggunaan `malloc()` untuk Membuat Structure Dinamis

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

//membuat struct
typedef struct Mahasiswa {
   char  name[20];
   char  address[20];
   int   age;
} mhs_struct;

int main( ) {

   mhs_struct *ptr_mhs;
   int i, n;

   // allocating memory untuk n struct mahasiswa
   ptr_mhs = (mhs_struct*)malloc(sizeof(mhs_struct));

   //mengisi nilai elemen struct
   strcpy( ptr_mhs->name, "Dian");
   strcpy( ptr_mhs->address, "Mataram");
   ptr_mhs->age = 22;

   // Mencetak isi elemen pada struct
   printf( "## Mahasiswa ##\n");
   printf( "Nama: %s\n", ptr_mhs->name);
   printf( "Alamat: %s\n", ptr_mhs->address);
   printf( "Umur: %d\n\n", ptr_mhs->age);

   return 0;
}
```

Pointers sebagai Structure Field

```c
#include <stdio.h>
#include <string.h>

//membuat struct
typedef struct Mahasiswa {
   char  name[20];
   char  address[20];
   int   age;
} mhs_struct;

int main( ) {
   //menggunakan struct
   mhs_struct mhs1;

   //mengisi nilai elemen struct
   strcpy( mhs1.name, "Dian");
   strcpy( mhs1.address, "Mataram");
   mhs1.age = 22;

   // Mencetak isi elemen pada struct
   printf( "## Mahasiswa ##\n");
   printf( "Nama: %s\n", mhs1.name);
   printf( "Alamat: %s\n", mhs1.address);
   printf( "Umur: %d\n\n", mhs1.age);

   return 0;
}
```

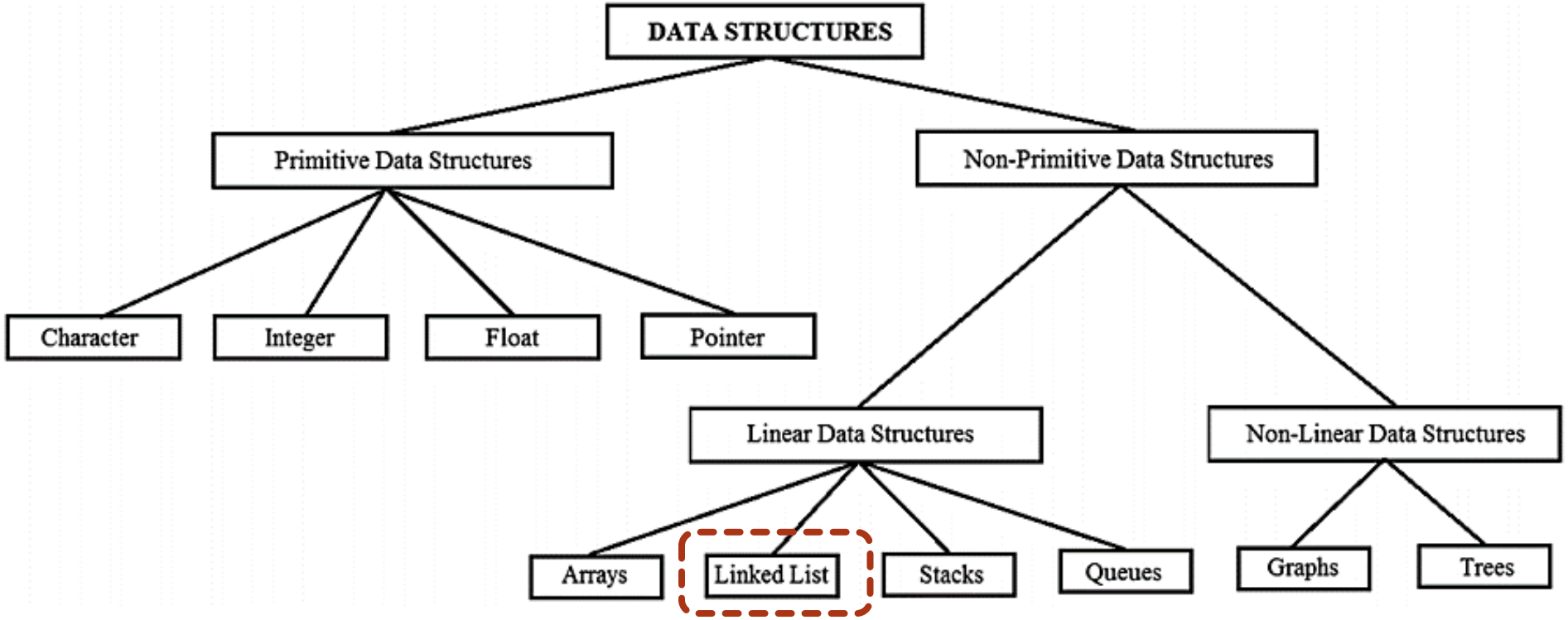
```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

//membuat struct
typedef struct Mahasiswa {
   char  name[20];
   char  address[20];
   int   *age;
} mhs_struct;

int main( ) {
    //menggunakan struct
   mhs_struct mhs1;

   //mengisi nilai elemen struct
   strcpy( mhs1.name, "Dian");
   strcpy( mhs1.address, "Mataram");
   mhs1.age = (int*)malloc (sizeof(int));
   *mhs1.age = 22;

   // Mencetak isi elemen pada struct
   printf( "## Mahasiswa ##\n");
   printf( "Nama: %s\n", mhs1.name);
   printf( "Alamat: %s\n", mhs1.address);
   printf( "Umur: %d\n\n", *mhs1.age);

   return 0;
}
```

Pointers ke Structure

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

//membuat struct
typedef struct Mahasiswa {
    char  name[20];
    char  address[20];
    int   *age;
} mhs_struct;

int main( ) {
     //menggunakan struct
    mhs_struct *ptr_mhs1, mhs1;

    //mengisi nilai elemen struct
    strcpy( mhs1.name, "Dian");
    strcpy( mhs1.address, "Mataram");
    mhs1.age = (int*)malloc (sizeof(int));
    *mhs1.age = 22;

    ptr_mhs1 = &mhs1;

    // Mencetak isi elemen pada struct
    printf( "## Mahasiswa ##\n");
    printf( "Nama: %s\n", ptr_mhs1->name);
    printf( "Alamat: %s\n", ptr_mhs1->address);
    printf( "Umur: %d\n\n", *ptr_mhs1->age);

    return 0;
}
```

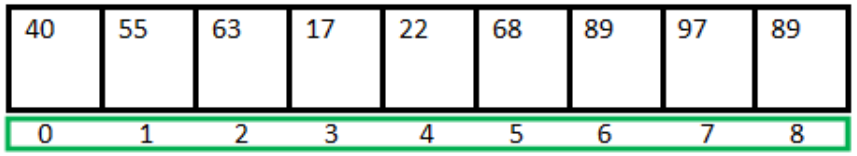
PENGENALAN
SINGLE LINKED LIST

Jenis-Jenis Struktur Data



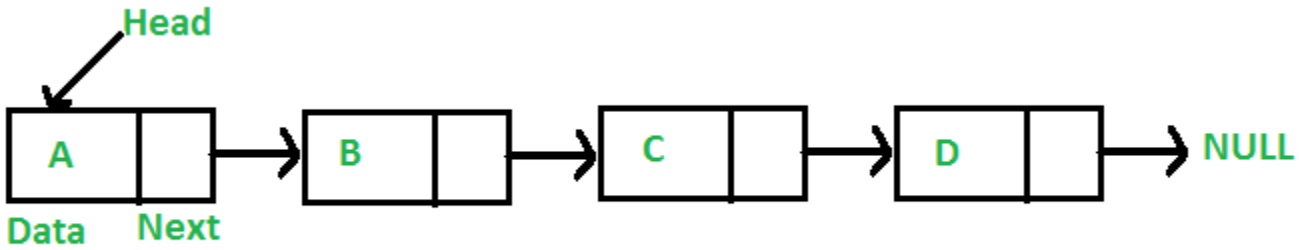
Linked List

- Sebuah struktur data seperti array yang berupa **sekumpulan node (simpul) yang saling terhubung secara linear dengan node lain melalui sebuah pointer**
- Node-node tersebut tidak disimpan secara berdampingan seperti array, tetapi terpencar-pencar di dalam memory → membutuhkan pointer yang menghubungkan satu node ke node berikutnya (**pointer bertugas menyimpan address node selanjutnya**)



Array Length = 9
First Index = 0
Last Index = 8

(Array)



(Linked List)

<u>Representasi</u>

Elemen dalam array = node dalam linked list

Linked List vs Array

Mengapa menggunakan linked list daripada array untuk menyimpan data?

- Ukuran array adalah tetap (tidak dinamis). Alokasi memory terbuang jika array tidak diisi penuh, dan bermasalah ketika harus menambah ukuran yang ditetapkan di awal

  → linked list akan mengalokasikan memory untuk setiap elemennya secara terpisah dan hanya ketika dibutuhkan

- Insert (sisip) elemen baru di awal ataupun tengah array membutuhkan usaha/komputasi yang besar (apalagi untuk array dengan jumlah elemen yang besar)

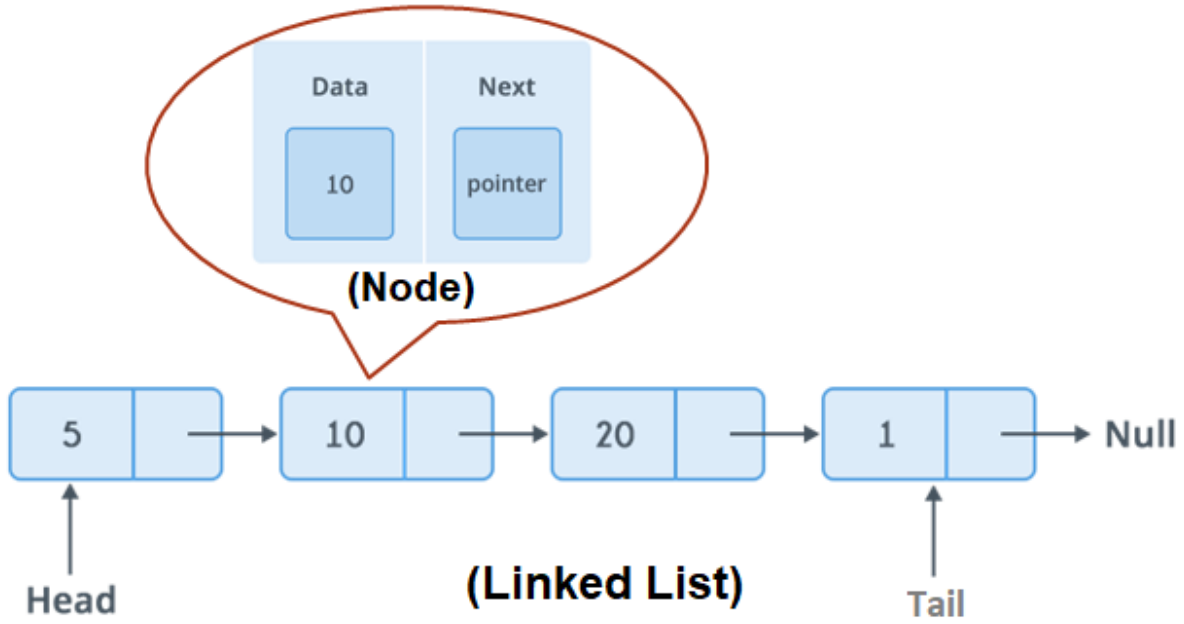
  → Misal: kita punya array nama mahasiswa berukuran 100, kita ingin sisipkan elemen di index ke 50, berarti kita harus menggeser satu per satu nilai di setiap index ke 51 sampai 100.

Linked List

- Setiap node terdiri dari 2 bagian:
  - **Data** berisi elemen data dalam node tersebut
  - **Pointer Next** berisi alamat memory node selanjutnya (untuk menghubungkan)

- Diawali dengan sebuah node **head** untuk menyimpan alamat awal dan diakhiri dengan node **tail** dengan pointer mengarah ke Null (menunjukan akhir dari sebuah list)

- Setiap node diimplementasikan secara dinamis (memory dialokasikan pada saat *runtime*)



Tipe Linked List

- **Single Linked List**
  - Pointer **Next** menyimpan alamat dari node berikutnya
- **Double Linked List**
  - Dua Pointer **Prev** dan **Next,** menyimpan alamat dari node sebelumnya dan node berikutnya
- **Circular Linked List**

Deklarasi Single Linked List

- Setiap node akan berbentuk **`struct`** dan memiliki satu buah field bertipe **`struct`** yang sama yang berfungsi sebagai pointer
- **Ingat**: cara mendeklarasikan **structure**

```
#include <stdio.h>

struct mahasiswa {
 char nim[25];
 char nama[25];
 int usia;
};
```

Deklarasi dan Akses:

```
struct mahasiswa mhs1;
struct mahasiswa mhs1 = {100, "Adi", 18};
printf("%s", mhs1.nama);
```

Variabel pointer : (yang menyimpan alamat memory struct)

```
struct mahasiswa mhs1, *p_mhs1;
struct mahasiswa mhs1 = {100, "Adi", 18};

p_mhs1 = &mhs1;

printf("%s", p_mhs1->nim);
printf("%s", p_mhs1->nama);
```

Deklarasi Single Linked List

- Setiap node akan berbentuk **`struct`** dan memiliki satu buah field bertipe **`struct`** yang sama berfungsi sebagai pointer

```
struct node{
        int data;
        struct node *next;
} ;
```

menyimpan alamat node setelahnya yang juga bertipe **`struct node`**, maka pointer **`next`** juga harus bertipe sama

(**Ingat**: pointer harus bertipe sama dengan nilai yang disimpan dalam alamat yang ditunjuk)

Membuat Node yaitu menggunakan Alokasi Memory Dinamis

```
struct mynode{
    int data;
    struct mynode *next;
};
struct mynode* head = NULL;
struct mynode* second = NULL;

head = (struct mynode*)malloc(sizeof(struct mynode));
second = (struct mynode*)malloc(sizeof(struct mynode));

head->data = 1;
head->next = second;

second->data = 2;
second->next = NULL;
```

Nama structure suatu node. Structure ini bisa disimpan sebagai global atau local.

Node `head = NULL` menunjukkan linked list masih kosong

Alokasikan memory secara dinamis

Isi elemen data dan pointer next di setiap node

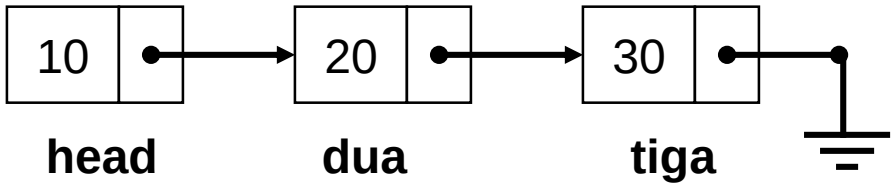
Akhir sebuah list (pointer node terakhir mengarah ke NULL)

Create Linked List

```c
#include <stdio.h>
#include <stdlib.h>

struct Node {
    int data;
    struct Node* next;
};

int main()
{
    struct Node* head = NULL;
    struct Node* dua = NULL;
    struct Node* tiga = NULL;
    head = (struct Node*)malloc(sizeof(struct Node));
    dua = (struct Node*)malloc(sizeof(struct Node));
    tiga = (struct Node*)malloc(sizeof(struct Node));

    head->data = 10;
    head->next = dua;

    dua->data = 20;
    dua->next = tiga;

    tiga->data = 30;
    tiga->next = NULL;

    printf("Isi dari linked list :\n");
    struct Node* n = head;
    while (n != NULL) {
        printf("%d\n", n->data);
        n = n->next;
    }
    return 0;
}
```

head, dua, tiga berisi alamat memory pertama node

```
Isi dari linked list :
10
20
30
```



Iterasi setiap node dalam sebuah linked list:

```c
Node* n;
n = head;
while(n != NULL){
    ....
    n = n->next;
}
```

node n adalah pointer bantuan. Lakukan **printf() field data** dari setiap node dari head, node berikutnya, dst sampai node tersebut NULL

Create Linked List

```
struct Node* head = NULL;
struct Node* dua = NULL;
struct Node* tiga = NULL;

head = (struct Node*)malloc(sizeof(struct Node));
dua = (struct Node*)malloc(sizeof(struct Node));
tiga = (struct Node*)malloc(sizeof(struct Node));
```

Contoh statis (tidak dinamis):
Dideklarasikan secara statis 3 buah node dalam linked list tersebut

- Jika jumlah node dalam sebuah linked list ditentukan secara dinamis (misal dari input user), tidak ditentukan di awal. Bagaimana cara membentuk linked list?

  Hal-hal yang harus dilakukan:

  1. Create node head
  2. Insert node-node berikutnya sampai selesai
  3. Delete node jika diperlukan

Create Linked List

1. Deklarasikan structure node yang berisi data dan pointer next

```
struct node{
    int value;
    struct node* next;
};

typedef struct node* mynode;
```

Untuk selanjutnya akan dipakai sampai slide terakhir sebagai global variable

`typedef`: untuk mendefiniskan tipe data baru atau memberi alias/nama baru suatu tipe data.

- Coding lebih rapi/bersih (menyederhanakan tipe data yang panjang dan complex)
- Tidak perlu menuliskan `struct` di semua tempat

Contoh lain penggunaan `typedef`:

```
typedef unsigned char HURUF;
HURUF b1, b2;

typedef long long int LLI;
int x = sizeof(LLI);
```

Create Linked List

2. Buat fungsi untuk membuat node (dibuat fungsi sendiri karena akan dipanggil berkali-kali)

```
mynode createNode(int nilai){
     mynode p;
     p = (mynode)malloc(sizeof(struct node));
     p->value = nilai;
     p->next = NULL ;
     return(p);
}
```

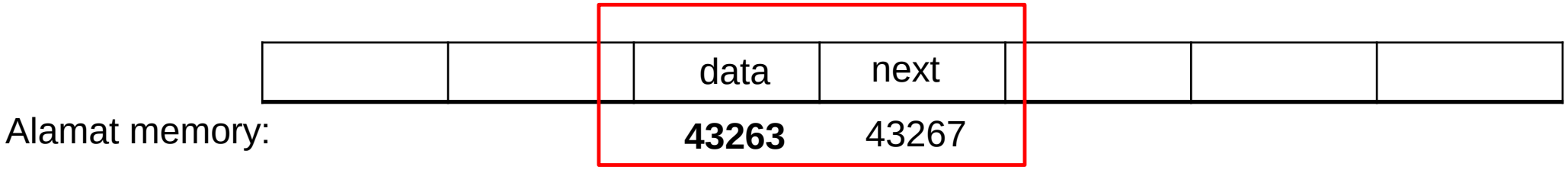
Bagaimana jika tidak mendeklarasikan **`typedef struct node *mynode`**?
Jawab:

```
struct node* createNode(int nilai){
    struct node* p;
    p = (struct node*)malloc(sizeof(struct node));
    p->value = nilai;
    p->next = NULL ;
    return(p);
}
```

Cara Kerja Fungsi `createNode()`

Apa yang terjadi jika kita mendeklarasikan sebuah node?

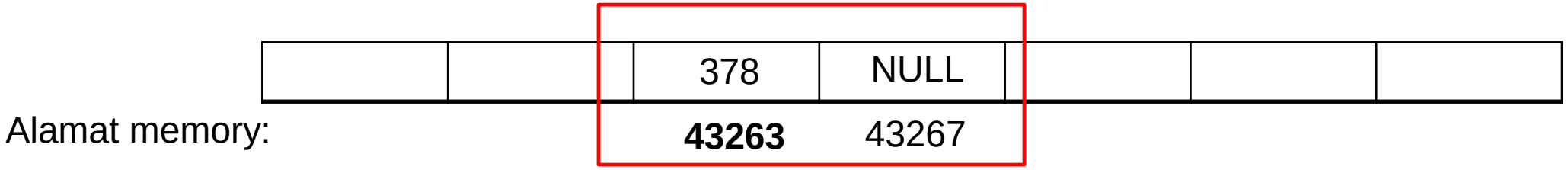
```
struct node{
    int data;
    struct node *next;
};

struct node* head = createNode(378);
struct node * p = (struct node *)malloc(sizeof(struct node));
```



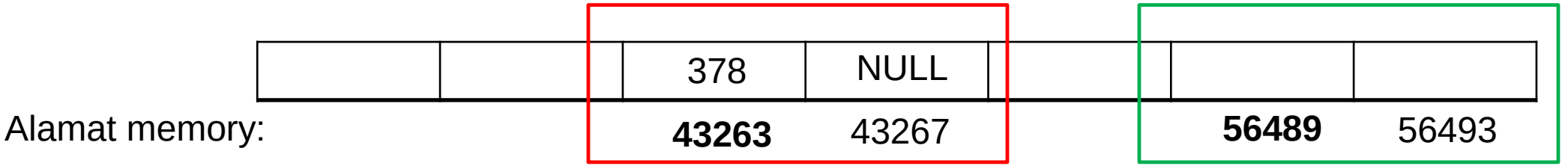
p akan berisi alamat pertama dari memory yang dialokasikan (kotak merah) yaitu **43263**

Cara Kerja Fungsi `createNode()`

```
p->data = 378;
p->next = NULL ;
```




```
struct node* new_node = createNode(500);
struct node * p = (struct node *)malloc(sizeof(struct node));
```



Alokasi **head**

Alokasi **new_node**

Sama seperti **head, new_node = 56489**

Menampilkan Isi Linked List

Untuk mengakses array, kita memakai nama variabel array dan indexnya

Untuk mengakses linked list (node-node di dalamnya) yang diketahui adalah **node/pointer head**
(karena dari head kita bisa baca seluruh elemen dalam linked list)

Menampilkan Isi Linked List

```c
#include<stdio.h>
#include<stdlib.h>

struct node{
    int value;
    struct node *next;
};

typedef struct node *mynode;

void display_list(mynode head){
    mynode tmp = head;
    while(tmp != NULL){
        printf("%d\n", tmp->value);
        tmp = tmp->next;
    }
    printf("selesai");
}
```

Fungsi untuk iterasi dari head sampai node terakhir dan **printf** value dari setiap node

```c
int main(){
    mynode head = NULL;
    mynode dua = NULL;
    mynode tiga = NULL;
    mynode empat = NULL;
    head = (mynode)malloc(sizeof(struct node));
    dua = (mynode)malloc(sizeof(struct node));
    tiga = (mynode)malloc(sizeof(struct node));
    empat = (mynode)malloc(sizeof(struct node));

    head->value = 10;
    head->next = dua;

    dua->value = 20;
    dua->next = tiga;

    tiga->value = 30;
    tiga->next = empat;

    empat->value = 40;
    empat->next = NULL;

    display_list(head);

    return 0;
}
```

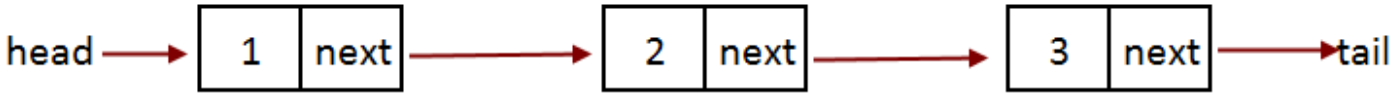
Output:

```
10
20
30
40
selesai
```

PENGENALAN DOUBLE LINKED LIST

Double Linked List

- **Single Linked List**
  - Pointer **Next** menyimpan alamat dari node berikutnya
- **Double Linked List**
  - Pointer **Prev** dan **Next** menyimpan alamat dari node sebelumnya dan node berikutnya



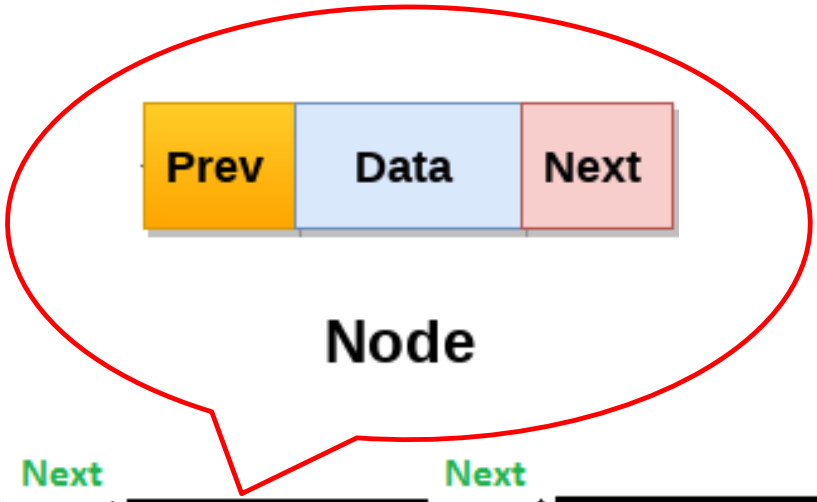
Singly Linked List

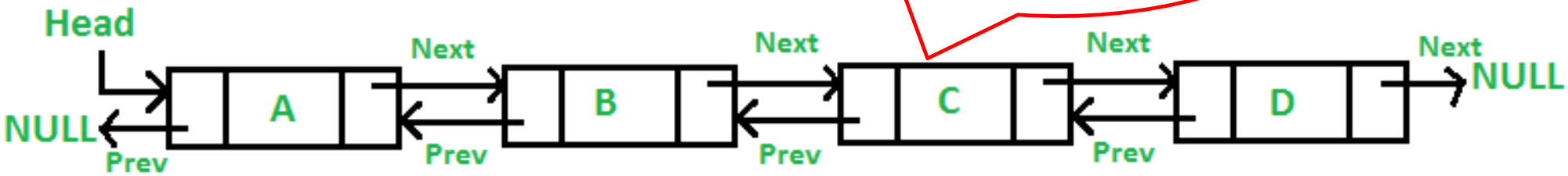
Doubly Linked List

Double Linked List

- Setiap node terdiri dari 3 bagian:
  - **Data** yang berisi elemen data pada node tersebut
  - **Pointer Next** yang menunjuk ke node berikutnya
  - **Pointer Prev** yang menunjuk ke node sebelumnya

- Pointer **Prev** dari node head (elemen pertama) menunjuk NULL
- Pointer **Next** dari node tail (elemen terakhir) menunjuk NULL





Deklarasi Double Linked List

- Sama seperti single linked list, setiap node akan berbentuk **`struct`** dan memiliki **dua** buah pointer bertipe **`struct`** yang sama yang berfungsi sebagai pointer **Prev** dan **Next**

```
struct node {
        int data;
        struct node* next;
        struct node* prev;
};
```

Menyimpan alamat node setelahnya

Menyimpan alamat node sebelumnya

Pointer harus bertipe sama dengan nilai yang disimpan dalam alamat yang ditunjuk

Buat Node dengan Alokasi Memory Dinamis

```
struct node{
    int data;
    struct node *next;
    struct node *prev;
};

struct node* head = (struct node*)malloc(sizeof(struct node));
struct node* tail = (struct node*)malloc(sizeof(struct node));

head->data = 40;
head->next = tail;
head->prev = NULL;

tail->data = 50;
tail->next = NULL;
tail->prev = head;
```

Pointer Prev untuk elemen pertama menunjuk ke NULL

Pointer Next untuk elemen terakhir menunjuk ke NULL

Create Double Linked List

1. Deklarasikan structure node yang berisi data, pointer next, dan pointer prev

```
struct node{
    int data;
    struct node *next;
    struct node *prev;
};
typedef struct node* mynode;
```

Untuk selanjutnya akan dipakai pada slide-slide selanjutnya sebagai global variable

- Structure node tersebut kemudian bisa didefinisikan dengan **`typedef`**

  → **OPTIONAL** (boleh pakai, boleh tidak, digunakan untuk menyederhanakan)

Create Double Linked List

2. Buat fungsi create node dan panggil fungsi tersebut untuk membentuk sebuah double linked list

```
mynode createNode(int nilai){
    mynode temp;
    temp =(mynode)malloc(sizeof(struct node));
    temp->data = nilai;
    temp->prev = NULL;
    temp->next = NULL;

    return(temp);
}
```

Jika kita punya fungsi di atas, bagaimana memanggilnya di fungsi main?

Menelusuri Double Linked List (Traversal)

- **Traversal** : membaca elemen-elemen dalam double linked list
- **Forward Traversal**
  - Mulai dari node pertama dan lewati semua node sampai node menunjuk NULL
- **Backward Traversal**
  - Mulai dari node terakhir dan lewati semua node sampai node menunjuk NULL

```
void traverse_beg(mynode head){
    mynode tmp = head;
    while(tmp != NULL){
        printf("%d\n", tmp->data);
        tmp = tmp->next;
    }
    printf("selesai");
}
```

```
void traverse_end(mynode tail){
    mynode tmp = tail;
    while(tmp != NULL){
        printf("%d\n", tmp->data);
        tmp = tmp->prev;
    }
    printf("selesai");
}
```

TERIMA KASIH