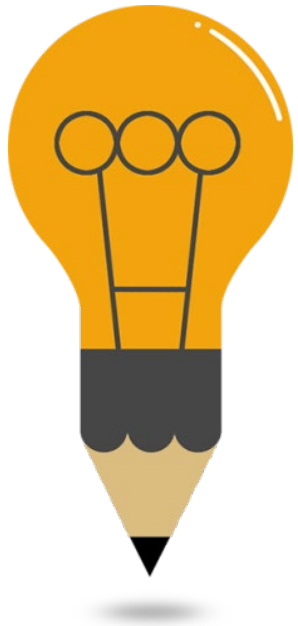
STRUKTUR DATA

Pertemuan 3

(Tim)

Agenda Pertemuan



1. Pengenalan Bahasa C
2. Struktur
3. Pointer
4. Alokasi Memori

Tipe Data di Bahasa C

Tipe Data C	Kode Format
char	%c
int	%d
float	%f
double	%f
string	%s

- **Deklarasi Variabel:** `data_type variable_name;`

  Contoh:

```
int x, y, z;
char flag;
char ch;
```

- **Initialisasi/Definisi Variabel:** `data_type variable_name = value;`

  Contoh:

```
int x = 30;
y = 100;
z = y;
char flag = 'x';
ch = 'n';
```

Tipe Data berdasarkan *Scope* dalam Program

1. **Local Variable**
   - Ruang lingkup hanya di dalam fungsi(blok program) dimana dia dideklarasikan
   - Tidak dapat diakses di luar fungsi(blok program) tersebut
2. **Global Variable**
   - Ruang lingkup berada di seluruh program
   - Didefinisikan di luar *main function* jadi bisa dipanggil di *main function* dan fungsi-fungsi lain
   - Variabel-variabel ini dapat diakses dari mana saja dalam program ini
3. **Environment Variable**
   - Variabel yang tersedia untuk semua aplikasi dan program
   - Dapat diakses dimanapun di program C tanpa harus mendeklarasikan atau mendefiniskannya
   - Akses dengan inbuilt function: `getenv()`, modifikasi: `setenv()`, assign: `putenv()`

Local & Global Variable

```
#include <stdio.h>
int m = 22, n = 44;
int a = 50, b = 80;

int main()
{
    printf("Tampilkan semua variabel dari main function:");
    printf("\nvalue: m = %d,n = %d, a = %d, b = %d", m, n, a, b);

    test();
}

void test()
{
    int x = 100, y = 200;
    printf("\n\nTampilkan semua variabel dari test function:");
    printf("\nvalue: m = %d,n = %d, a = %d, b = %d, x = %d, y = %d", m, n, a, b, x, y);

}
```

Global Variables

Fungsi Utama

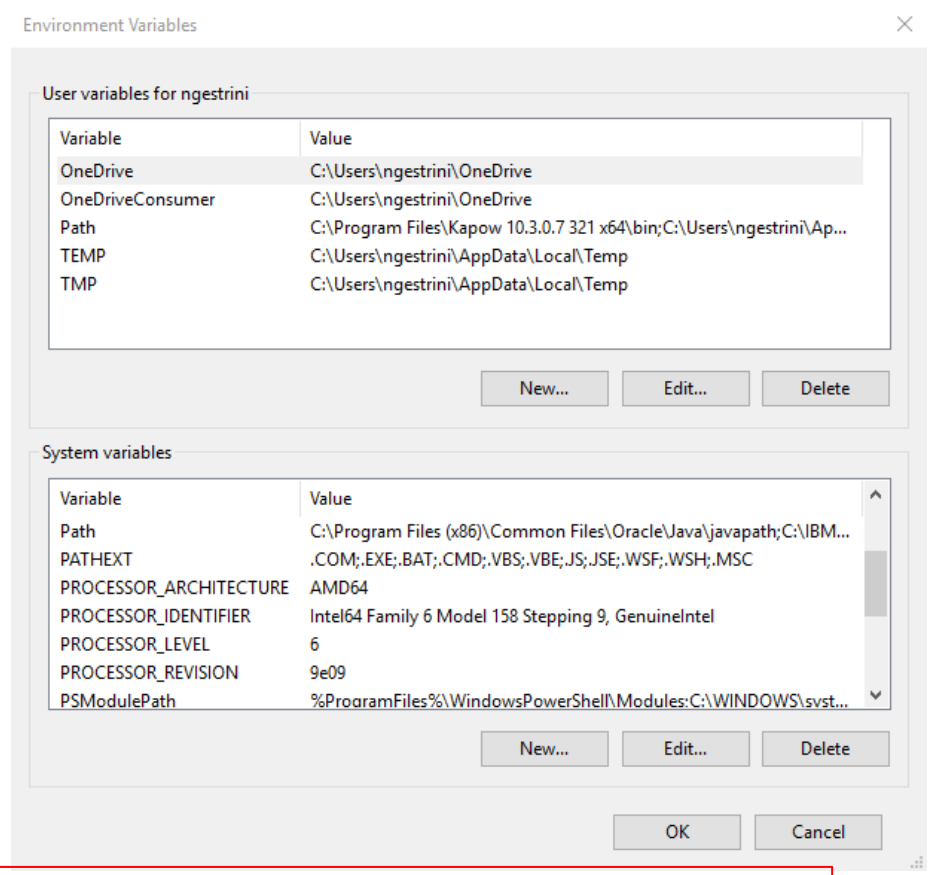
Fungsi **test()**

Local Variables

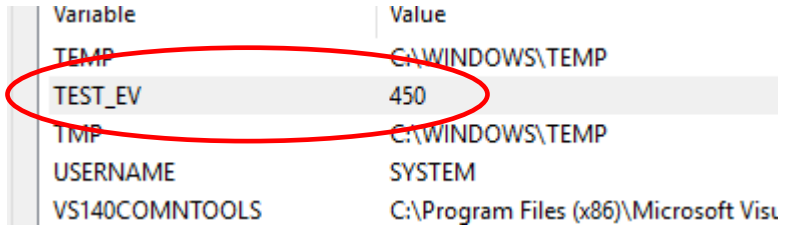
Hasil:

```
Tampilkan semua variabel dari main function:
value: m = 22,n = 44, a = 50, b = 80

Tampilkan semua variabel dari test function:
value: m = 22,n = 44, a = 50, b = 80, x = 100, y = 200
```

Environment Variable



Untuk Windows di control panel - system properties – environment variables

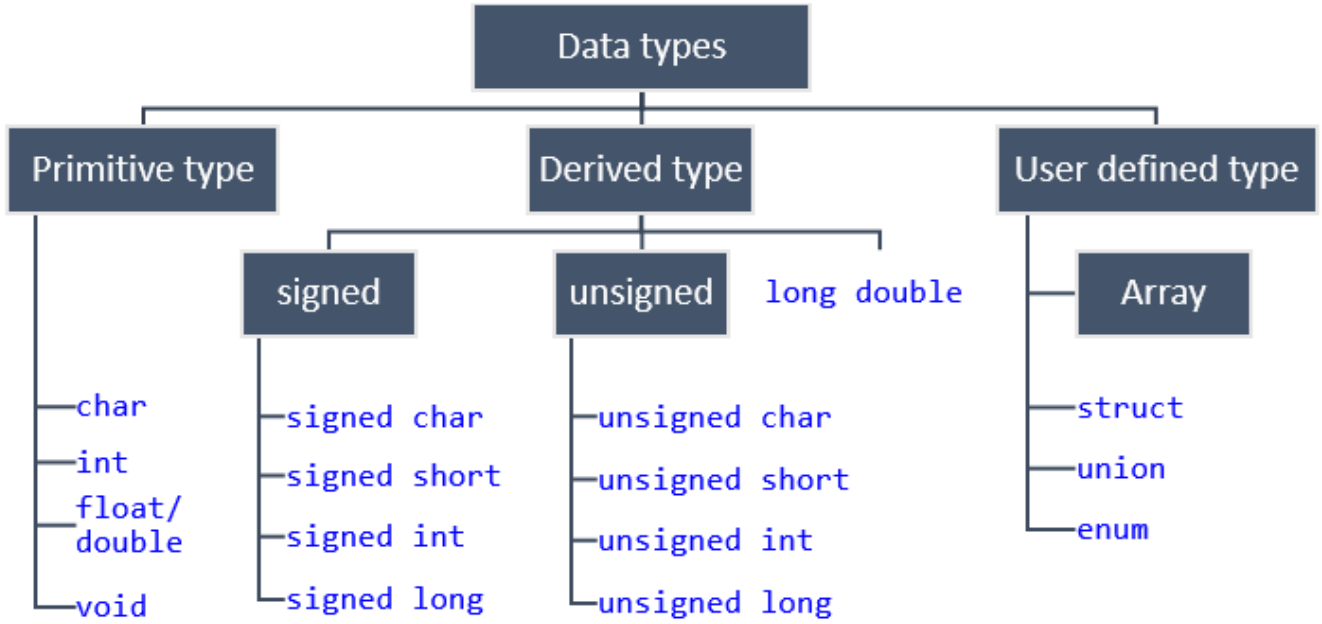



```c
#include <stdio.h>
#include <stdlib.h>

int main()
{

    printf("Nilai awal TEST_EV = %d\n", getenv("TEST_EV"));
    setenv("TEST_EV",300,1);
    printf("Nilai akhir TEST_EV = %d\n", getenv("TEST_EV"));
    return 0;
}
```

`getenv()` = mengambil nilai env variabel

`setenv()` = mengisi nilai env variabel

Tipe Data dalam Bahasa C



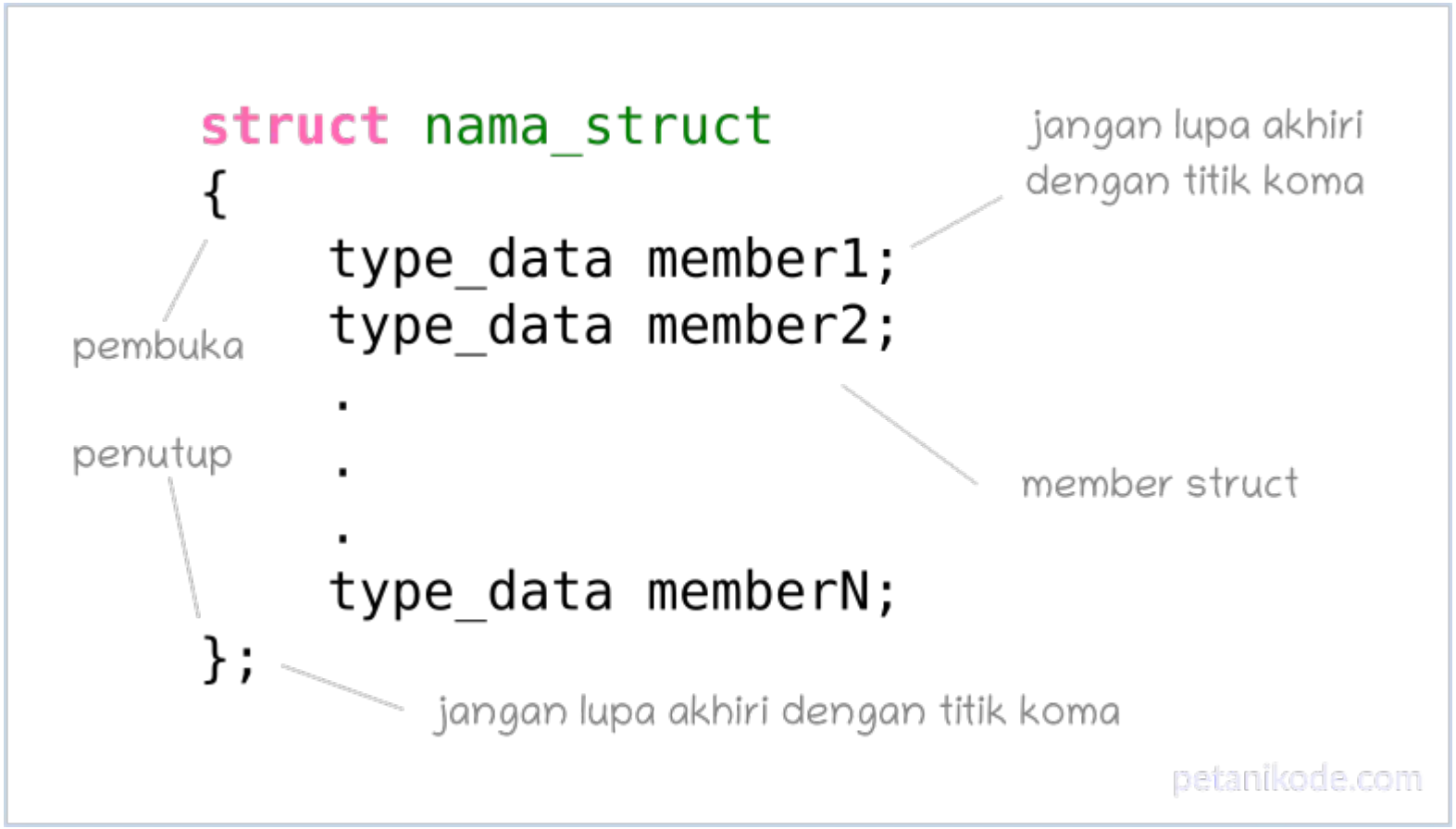
Tipe data yang didefinisikan user *(user-defined)*

- **Array**: Tipe data yang terdiri dari kumpulan tipe data dasar. Tipe data tersebut harus 1 jenis.
- **Pointer**: Tipe data untuk mengakses alamat memory suatu variabel secara langsung.
- **Structure**: Tipe data yang terdiri dari kumpulan tipe data dasar. Tipe data tersebut bisa lebih dari 1 jenis.
- **Union**: Tipe data yang menyimpan beberapa tipe data berbeda dalam lokasi memory yang sama.

Struktur

- kumpulan dari beberapa variabel dengan beragam tipe data yang dibungkus dalam satu varabel
- dikenal dengan *records* dalam bahasa pemrograman lain seperti Pascal

Deklarasi Struktur



Penggunaan struktur tanpa typedef

```
1  // membuat struct
2  struct Distance{
3      int feet;
4      float inch;
5  };
6
7  void main() {
8      // menggunakan struct
9      struct Distance d1, d2;
10 }
```

Contoh Struktur tanpa typedef

```c
#include <stdio.h>

// membuat struct
struct Mahasiswa {
    char *name;
    char *address;
    int age;
};

void main(){

    // menggunakan struct
    struct Mahasiswa mhs;

    // mengisi nilai ke struct
    mhs.name = "Dian";
    mhs.address = "Mataram";
    mhs.age = 22;

    // mencetak isi struct
    printf("## Mahasiswa ##\n");
    printf("Nama: %s\n", mhs.name);
    printf("Alamat: %s\n", mhs.address);
    printf("Umur: %d\n", mhs.age);
}
```

Menggunakan typedef pada struktur

```
1  // membuat struct dengan typedef
2  typedef struct Distance{
3      int feet;
4      float inch;
5  } distances;
6
7  void main() {
8      // menggunakan struct
9      distances dist1, dist2, sum;
10 }
```

Struktur Bersarang

```
1  struct complex
2  {
3   int imag;
4   float real;
5  };
6
7  struct number
8  {
9      struct complex comp;
10     int integers;
11 } num1, num2;
```

Pasing Struktur ke dalam fungsi

```c
#include <stdio.h>
struct student
{
    char name[50];
    int age;
};

void main() {
    struct student s1;

    printf("Enter name: ");
    scanf("%[^\n]%*c", s1.name);
//    gets(s1.name);

    printf("Enter age: ");
    scanf("%d", &s1.age);

    display(s1);   // passing structure as an argument
}
```

```c
    display(s1);   // passing structure as an argument
}

// membuat fungsi dengan struct sebagai parameter
void display(struct student s) {
  printf("\nDisplaying information\n");
  printf("Name: %s", s.name);
  printf("\nRoll: %d", s.age);
}
```

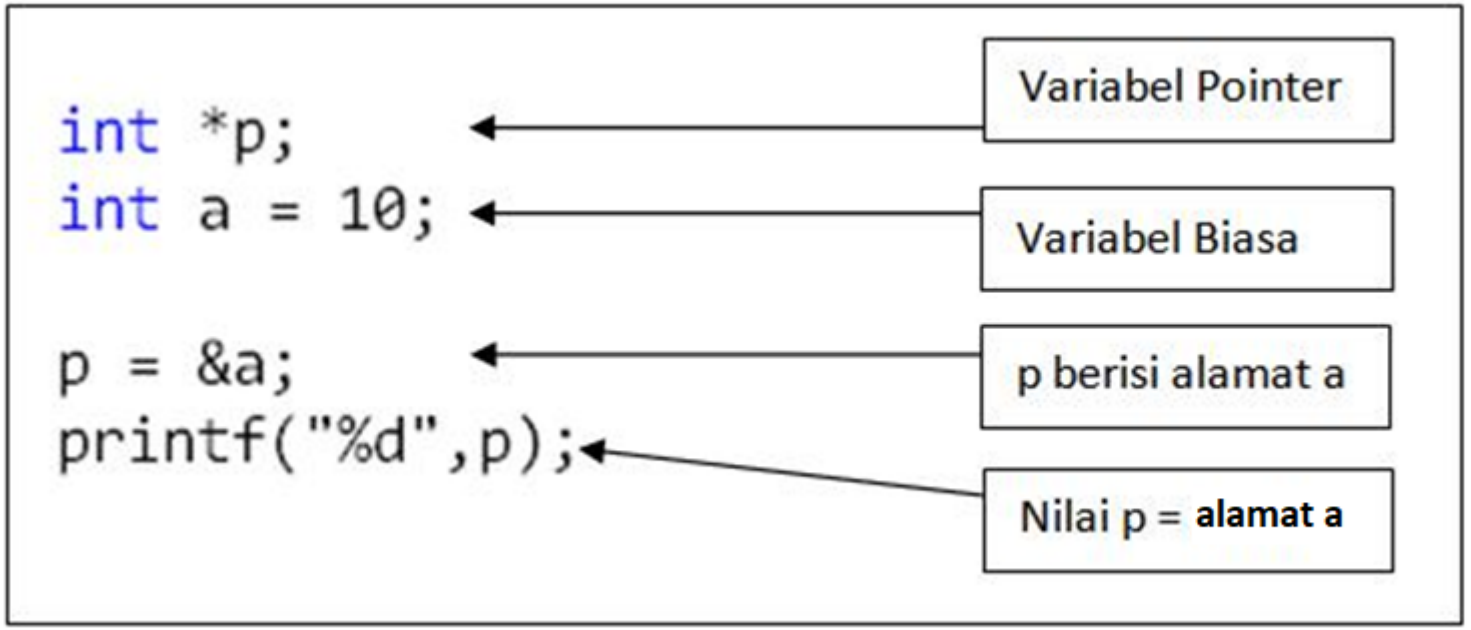
Pointer

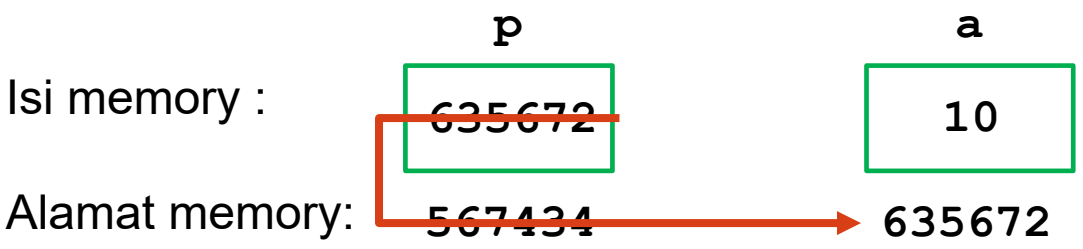
- Variabel yang menunjuk ke suatu variabel lain dengan menyimpan **alamat memory variabel** tersebut
- Contoh:

```
int *pi;
double *dp;
float *test;
```

- Tipe variabel pointer dan tipe data yang ditunjuk harus sejenis
- Operator:
  - * **(bintang)**: untuk mendapatkan nilai dari variabel yang ditunjuk oleh pointer
  - **&** : untuk mendapatkan alamat memory dari suatu variabel

Pointer





p adalah variabel pointer (menyimpan alamat memory), sedangkan ***a*** adalah variabel integer bernilai 10.

Ketika pernyataan ***p = &a*** dieksekusi maka variabel ***p*** akan menunjuk variabel ***a*** dengan menyimpan alamat memory dari ***a***.

(**ingat**: pointer dan variabel yang ditunjuk harus mempunyai tipe data yang sama, dalam contoh disamping, ***p*** dan ***a*** bertipe integer).

Ketika nilai ***p*** ditampilkan maka isinya adalah alamat memory dari ***a***.

Pointer (menampilkan nilai variabel dari pointer yang menunjuk)

```c
#include <stdio.h>

int main(){
    int *ptr, q;

    q = 50;

    ptr = &q;   /* mengambil alamat dari q */

    printf("Nilai : %d\n", *ptr); /* menampilkan nilai dari q */
    printf("Alamat : %d", ptr); /* menampilkan alamat dari q */

    return 0;
}
```

Hasil:

```
Nilai : 50
Alamat : 6356744
```

ptr menunjuk variabel ***q*** dengan menyimpan alamat memory dari ***q***.

Untuk menampilkan nilai dari ***q*** dari ***ptr***, gunakan statement ****ptr***

Latihan 1

```c
#include <stdio.h>
int main()
{
    int x, y;
    int *px;

    x = 150;
    px = &x;
    y = *px;

    printf("Alamat x = %p\n", &x);
    printf("Isi px = %p\n, px);
    printf("Nilai yang ditunjuk px = %d\n", *px);
    printf("Nilai y = %d\n", y);

    return 0;
}
```

1. Pernyataan `y = *px` bisa diganti dengan?

Latihan 2

```c
#include <stdio.h>
int main()
{
    float d, *pd;

    d = 54.5;
    pd = &d;

    printf("%g\n", d);

    *pd = *pd + 10;

    printf("%g\n", d);

    return 0;
}
```

2. Apa output dari program tersebut?

Pointer dan Array

- Misal sebuah Array **x** dan pointer **p**
- **Untuk menampilkan alamat setiap elemen**:

  Alamat elemen ke 1 : &x[0] atau x atau x+0 atau p atau p+0

  Alamat elemen ke 2 : &x[1] atau x+1 atau p+1

  Alamat elemen ke 3 : &x[2] atau x+2 atau p+2

  Alamat elemen ke n : &x[n-1] atau x+(n-1) atau p+(n-1)

X =	0	1	2	…	n-1
Alamat =	356	360	364		3xx

- **Untuk menampilkan nilai setiap elemen dalam array**:

  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))

Contoh: Pointer dari Array

Buat array dengan ukuran 3 berisi 10, 100, 1000. Tampilkan alamat memori untuk setiap isi dalam array tersebut menggunakan pointer dengan looping menggunakan **`for statement`**!

```c
#include <stdio.h>
int main()
{
    int x[] = {10, 100, 1000};
    int *px, i;

    px = &x;

    for(i = 0; i < 3; i++){
        printf("Nilai %d = %d, Alamat %d = %d\n", i, *(px + i), i, (px + i));
    }

    return 0;
}
```

Alamat elemen array juga bisa dipanggil dengan: **`(x+i)`**

```
Nilai 0 = 10,  Alamat 0 = 6356716
Nilai 1 = 100,  Alamat 1 = 6356720
Nilai 2 = 1000,  Alamat 2 = 6356724
```

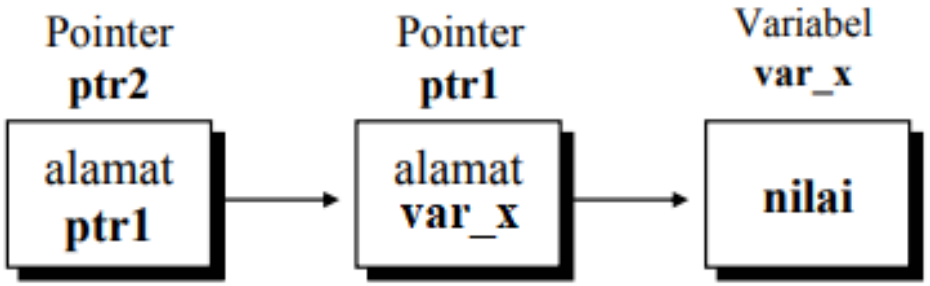
Array dari Pointer

```c
#include <stdio.h>

const int MAX = 3;

int main () {
   int  var[] = {10, 100, 200};
   int i, *ptr[MAX];

   for ( i = 0; i < MAX; i++) {
      ptr[i] = &var[i];
   }

   for ( i = 0; i < MAX; i++) {
      printf("Value of var[%d] = %d\n", i, *ptr[i] );
   }

   return 0;
}
```

Output:

```
Value of var[0] = 10
Value of var[1] = 100
Value of var[2] = 200
```

Pointer menunjuk ke Pointer (*pointer-to-pointer* atau double pointer)

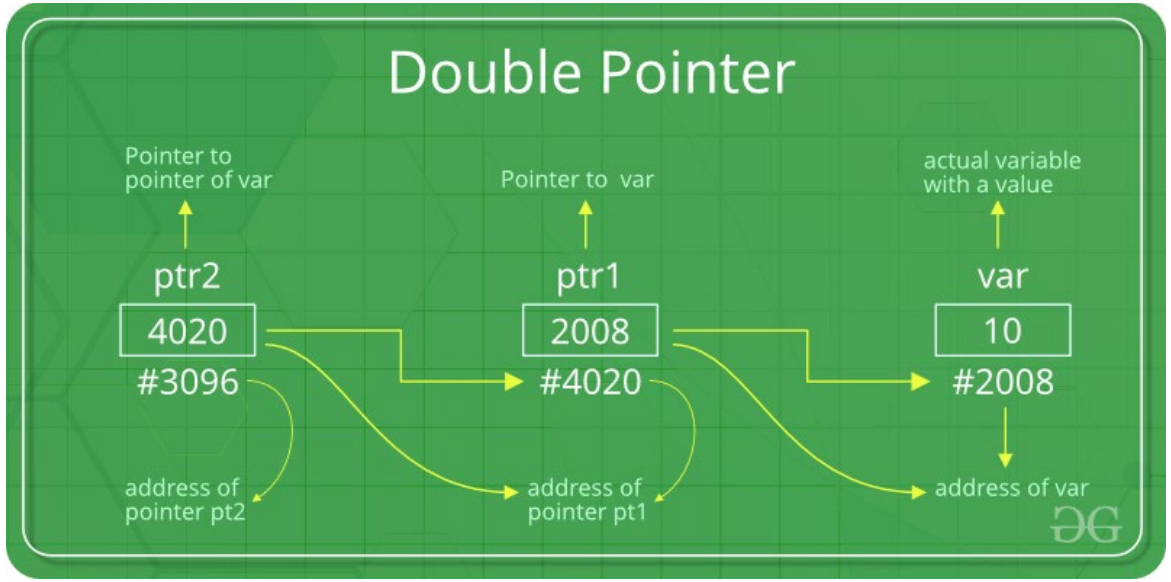


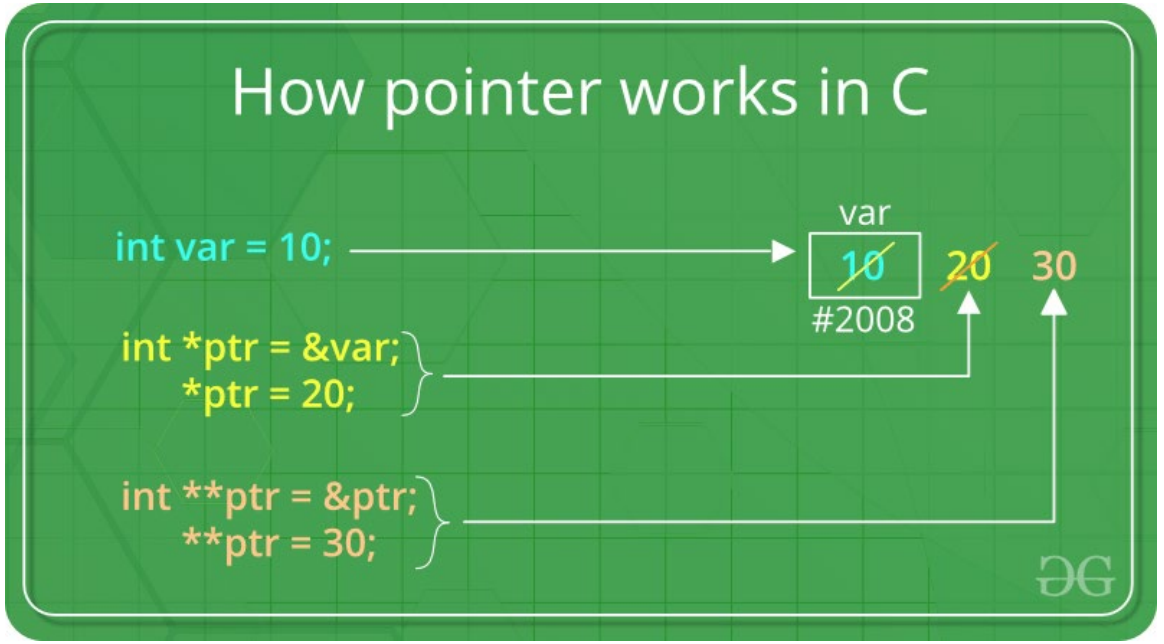
- Deklarasi:

```
int var_x;
int *ptr1;
int **ptr2;
```

- **`ptr1`** adalah variabel pointer yang menunjuk ke data bertipe int (`nilai`)
- **`ptr2`** adalah variabel pointer yang menunjuk ke pointer int (`ptr1`)

Pointer menunjuk ke Pointer (*pointer-to-pointer*)





- Pointer **`ptr1`** akan menunjuk variabel **`var`** dengan menyimpan alamat memory **`var`** yaitu **`2008`**
- Pointer **`ptr2`** akan menunjuk pointer **`ptr1`** dengan menyimpan alamat dari **`ptr1`** yaitu **`4020`**

- Kita bisa mengakses nilai dari variabel **`var`** dari pointer yang menunjuknya:
- **`*ptr1 = 20`** berarti nilai dari **`var`** diisi dengan nilai 20
- **`**ptr2 = 30`** berarti nilai dari **`var`** diisi dengan nilai 30

Contoh *pointer-to-pointer* atau Double Pointer

```c
#include <stdio.h>
int main()
{
    int var = 789;
    int *ptr2;
    int **ptr1;

    ptr2 = &var;
    ptr1 = &ptr2;

    printf("Nilai var = %d\n", var );
    printf("Nilai var menggunakan single pointer = %d\n", *ptr2 );
    printf("Nilai var menggunakan double pointer = %d\n", **ptr1);

    return 0;
}
```

- Berapa banyak level pointer yang dapat kita buat? Bisakah lebih dari **(double)?

Fungsi di Bahasa C

- Fungsi adalah suatu blok *statement/code* yang melakukan tugas tertentu

```c
#include <stdio.h>
int jumlah(int a, int b);
int main()
{
    int n1,n2,sum;
    printf("Masukan angka 1: ");
    scanf("%d",&n1);
    printf("Masukan angka 2: ");
    scanf("%d",&n2);
    sum = jumlah(n1, n2);
    printf("sum = %d",sum);
    return 0;
}
int jumlah(int a, int b)
{
    int hasil;
    hasil = a+b;
    return hasil;
}
```

Function prototype

Function call

Function definition

Return statement

Mengapa *function prototype* dibutuhkan?

- Fase 1 dari compiler (*lexical analysis*) akan membaca dari kiri ke kanan, atas ke bawah
- Menginformasikan ke compiler tentang nama fungsi, *return type*, dan parameter
- Jika fungsi didefinisikan sebelum `main()` maka *function prototype* tidak perlu

Fungsi di Bahasa C

Fungsi didefinisikan sebelum `main()` :

```c
#include<stdio.h>

void displayMessage() {
   printf("www.c4learn.com");
}

void main() {
   displayMessage();
}
```

Fungsi didefinisikan setelah `main()` :

```c
#include<stdio.h>

//Prototype Declaration
void displayMessage();

void main() {
   displayMessage();
}

void displayMessage() {
   printf("www.c4learn.com");
}
```

Void = Tidak *return* value apapun (seperti procedure dalam pascal)

Pointer Sebagai Parameter Fungsi

```c
#include <stdio.h>
void swap(int *a, int *b);
int main()
{
    int m = 10, n = 20;
    printf("m = %d\n", m);
    printf("n = %d\n\n", n);

    swap(&m, &n);
    printf("After Swapping:\n\n");
    printf("m = %d\n", m);
    printf("n = %d", n);
    return 0;
}

void swap(int *a, int *b)
{
    int temp;
    temp = *a;
    *a = *b;
    *b = temp;
}
```

- Hasil:

```
m = 10
n = 20

After Swapping:

m = 20
n = 10
```

- Bagaimana jika tidak memakai pointer?

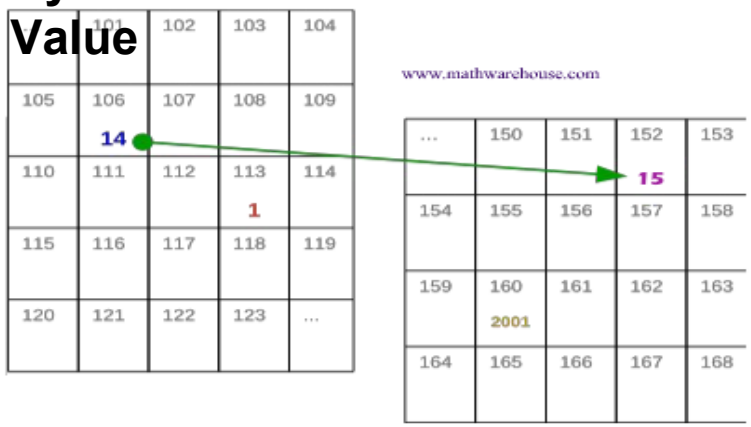
Fungsi di Bahasa C

- ***Pass by reference*** (Pointer Sebagai Parameter Fungsi) : alamat digunakan untuk mengakses parameter yang dipanggil fungsi, jika ada dilakukan perubahan dalam fungsi tsb maka akan mengubah nilai dalam alamat tsb
- ***Pass by value*** : mengakses nilai dari parameter yang dipanggil di fungsi

Pointer Sebagai Parameter Fungsi

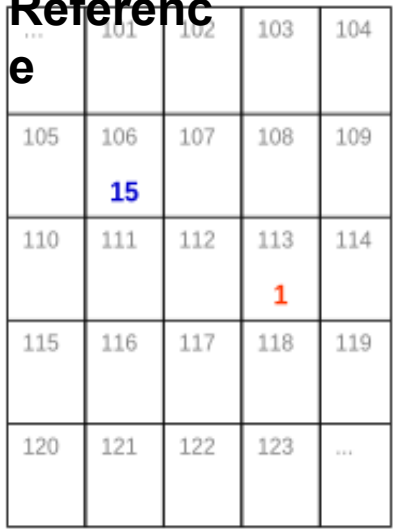
```c
#include <stdio.h>
int tambahsatu(int a);
int main()
{
    int m = 14;
    printf("m = %d\n", m);

    int n = tambahsatu(m);
    printf("m = %d\n", m);
    printf("n = %d\n", n);
    return 0;
}

int tambahsatu(int a)
{
    a = a + 1;
    return a;
}
```

By Value




```c
#include <stdio.h>
int tambahsatu(int *a);
int main()
{
    int m = 14;
    printf("m = %d\n", m);

    int n = tambahsatu(&m);
    printf("m = %d\n", m);
    printf("n = %d\n", n);
    return 0;
}

int tambahsatu(int *a)
{
    *a = *a + 1;
    return *a;
}
```

By Reference



Pointer Sebagai Return Value Suatu Fungsi

```c
#include <stdio.h>
int *getMax(int *, int *);

int main(void) {
  int x = 100;
  int y = 200;

  int *max;
  max = getMax(&x, &y);
  printf("Max value: %d\n", *max);

  return 0;
}

int *getMax(int *m, int *n) {
  if (*m > *n) {
    return m;
  }
  else {
    return n;
  }
}
```

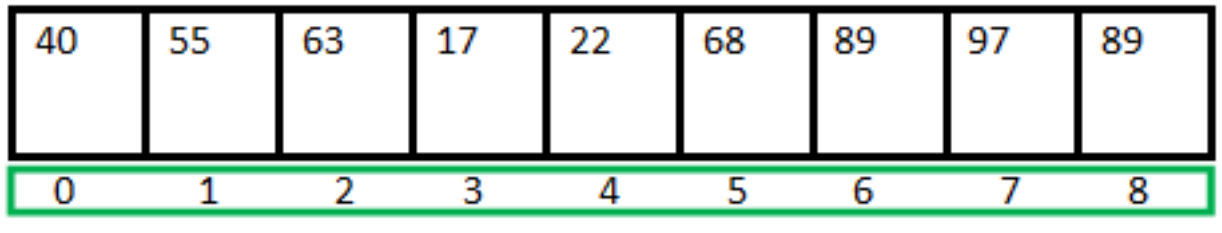
Jika nilai yang ditunjuk pointer m lebih besar dari nilai yang ditunjuk pointer n, maka return alamat dari m

Lainnya,
maka return alamat dari n

Alokasi Memori

- Menyediakan fasilitas untuk membuat ukuran buffer dan array secara dinamik.
- **Dinamik** artinya bahwa ruang dalam memori akan dialokasikan ketika program dieksekusi (run time)
- Fasilitas ini memungkinkan user untuk membuat tipe data dan struktur dengan ukuran dan panjang berapapun yang disesuaikan dengan kebutuhan di dalam program.

Alokasi Memori Dinamik



Array length = 9
Index pertama = 0
Index terakhir = 8

Index
Array

- Jika hanya ada 5 elemen yang akan dimasukan ke array, 4 index yang tidak digunakan mengambil memori (perlu mengurangi ukuran/panjang array dari 9 ke 5)
- Jika ada 3 elemen lagi yang perlu dimasukan ke array, maka perlu menambah ukuran/panjang array dari 9 ke 12

- Prosedur ini disebut **Dynamic Memory Allocation :** suatu prosedur dimana struktur data (seperti array) berubah selama program dieksekusi (*run time)* => ada 4 fungsi dari **<stdlib.h>** untuk melakukan alokasi memori:
  - `malloc(), calloc(), free(), realloc()`

Fungsi `sizeof()`

- Untuk mendapatkan ukuran dari berbagai tipe data, variabel, ataupun struktur
- **Structure**:

```c
#include <stdio.h>

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

typedef struct {
 char namamk[25];
 int semester;
 int sks;
}mataKuliah;
```

```c
void main(){
 struct mahasiswa mhsl = {"2016823", "Budi Wahana",18};
 mataKuliah mkl = {"Struktur Data", 2, 3};

 //tampilkan data Mahasiswa
 printf("NIM : %s\n",mhsl.nim);
 printf("Nama : %s\n",mhsl.nama);
 printf("Usia : %d\n",mhsl.usia);

 //tampilkan data Mata Kuliah
 printf("Mata Kuliah : %s\n",mkl.namamk);
 printf("Semester : %d\n",mkl.semester);
 printf("SKS : %d\n",mkl.sks);

 return 0;
}
```

Penggunaan `sizeof()`

- Jika yang dipanggil adalah tipe data, maka output dari sizeof() adalah jumlah memori yang dialokasikan untuk tipe data tersebut dalam byte

```c
#include <stdio.h>
struct employee{
    char name[40];
    int id;
};
int main() {
   int myInt = 16;
   struct employee john;
   int arr[] = { 1, 2, 3, 4, 7 };
   printf("Size of variable myInt : %d\n",sizeof(myInt));
   printf("Size of variable john : %d\n",sizeof(john));
   printf("Size of variable arr : %d\n",sizeof(arr));
   printf("Size of int data type : %d\n",sizeof(int));
   printf("Size of char data type : %d\n",sizeof(char));
   printf("Size of float data type : %d\n",sizeof(float));
   printf("Size of double data type : %d\n",sizeof(double));
   return 0;
}
```

```
Size of variable myInt : 4
Size of variable john : 44
Size of variable arr : 20
Size of int data type : 4
Size of char data type : 1
Size of float data type : 4
Size of double data type : 8
```

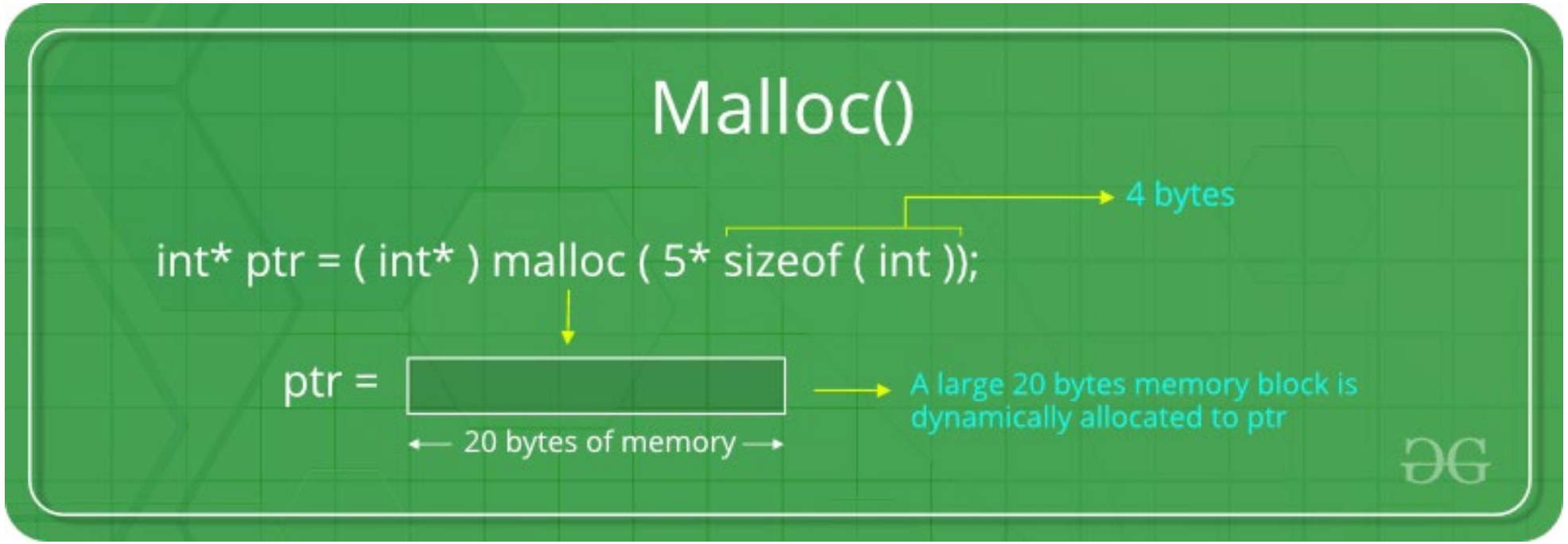
Fungsi `malloc()`

- “malloc” atau “memory allocation” digunakan untuk mengalokasikan satu blok memori dengan ukuran tertentu secara dinamis
- **Jika berhasil/sukses**, `malloc()` akan return sebuah pointer bertipe *void* yang dapat dikonversi ke pointer dengan tipe lain
- **Jika gagal**, fungsi akan return sebuah pointer NULL

- **`ptr = (tipe_data_konversi*) malloc(jumlah_byte)`**
- Contoh:
  - **`int *ptr = (int*) malloc(100 * sizeof(int));`**

  => (ukuran dari int adalah 4 byte, fungsi malloc di sini akan mengalokasikan memori 400 bytes, dan **pointer ptr akan menyimpan alamat byte pertama dari memori yang dialokasikan**)

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 `malloc()`

```
int *ptr;
ptr = (int*) malloc(sizeof(int));
```

sizeof(int) = ukuran byte sebuah integer

malloc mengalokasikan storage memori dengan ukuran 4 byte (int)

ptr = **void*** malloc(…)

Pointer tersebut haruslah dikonversi kepada tipe yang sesuai

ptr = **(int*)** malloc(…)

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,

Fungsi `free()`

- Jika bekerja dengan menggunakan memori yang dialokasikan secara dinamis, maka memori harus dibebaskan kembali setelah selesai digunakan untuk dikembalikan kepada sistem.
- Setelah suatu ruang memori dibebaskan, ruang tersebut bisa dipakai lagi untuk alokasi variabel dinamis lainnya.

Fungsi `free()`

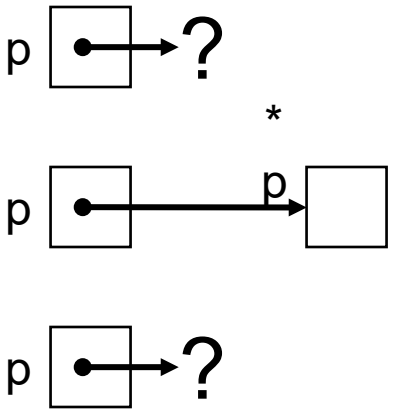
```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    char *pblok;
    pblok = (char *) malloc(500 * sizeof(char));
    if (pblok == NULL)
        printf("Error on malloc");
    else {
        printf("OK, alokasi memori sudah dilakukan\n");
        printf("------\n");
        free(pblok);
        printf("Blok memori telah dibebaskan kembali\n");
    }
}
```

Fungsi `free()`

```
#include <stdlib.h>
int main () {
  int *p;  int a=2;
  p = (int*) malloc(sizeof(int));
  *p = 4;
  *p += a;
       …

  p = NULL;
}
```

Menghapus sel yang ditunjuk p dengan **`p = NULL`** atau **`free(p)`**

Pada saat variabel dinamik tidak digunakan lagi kita perlu membebaskannya. Kompiler tidak mendealokasi storage space secara otomatis



Latihan

- Buat array dinamis 2 dimensi menggunakan fungsi malloc()!
- Buat program untuk menghitung jumlah dari ***N*** angka integer yang diinput oleh user!

Pertemuan Selanjutnya

- Fungsi `calloc()`
- Fungsi `realloc()`
- Linked List

TERIMA KASIH