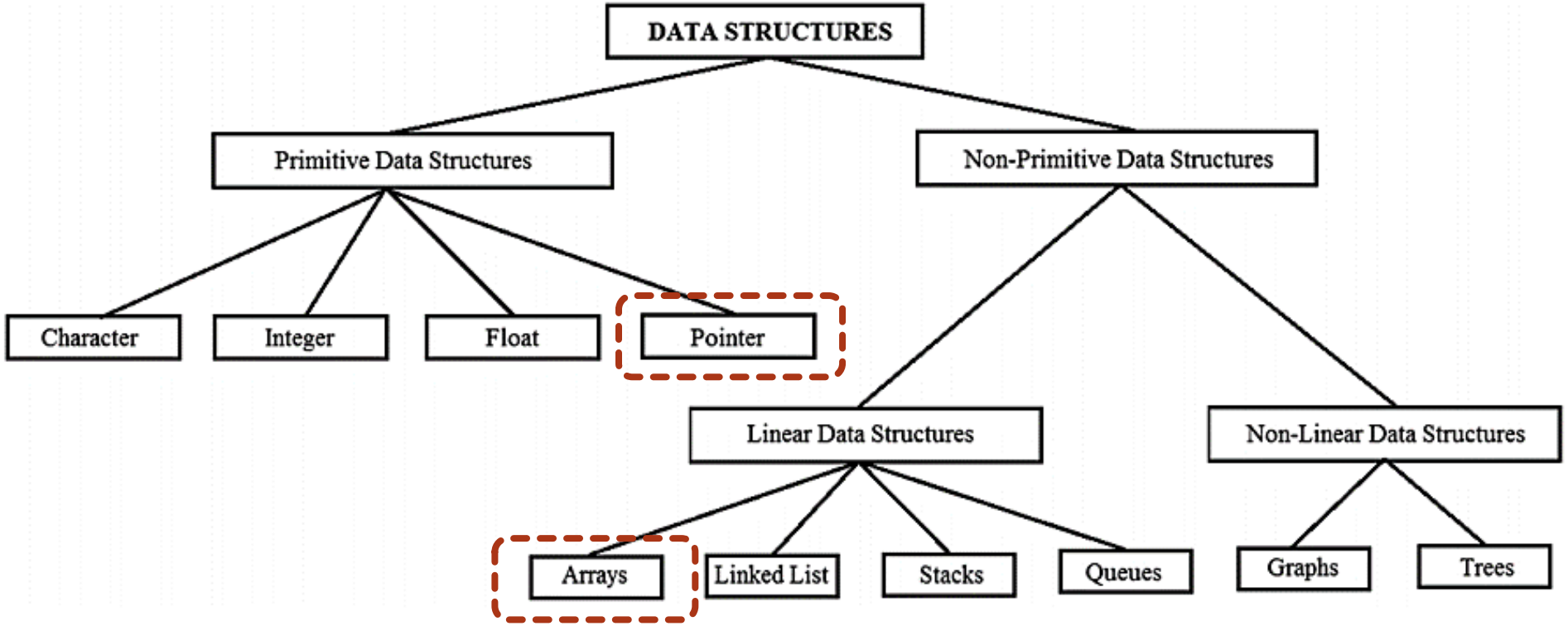
STRUKTUR DATA

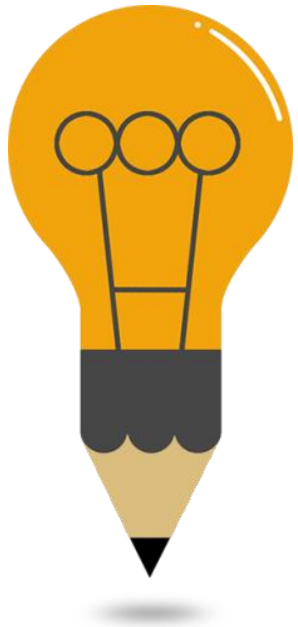
Pertemuan 3

Ratih Ngestrini, Nori Wilantika

Jenis-Jenis Struktur Data



Agenda Pertemuan



1. Pointer
2. Array
3. Alokasi Memori Dinamis

POINTER

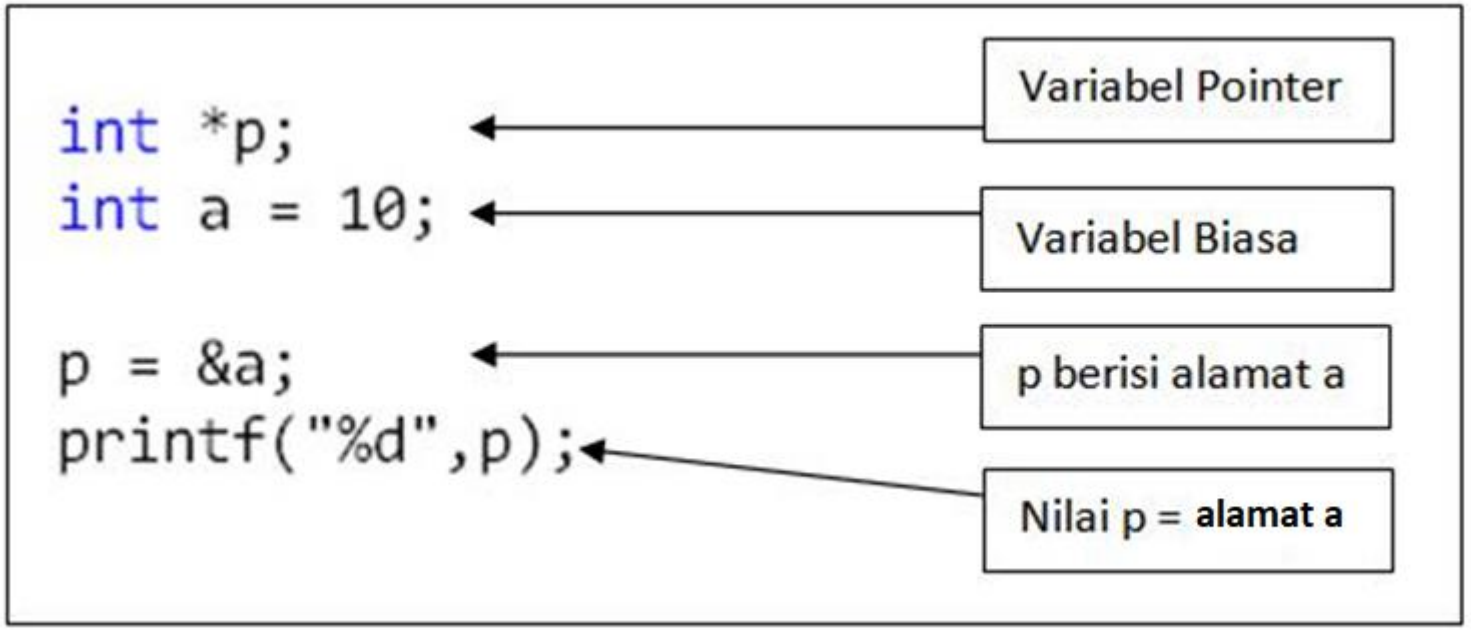
Pointer

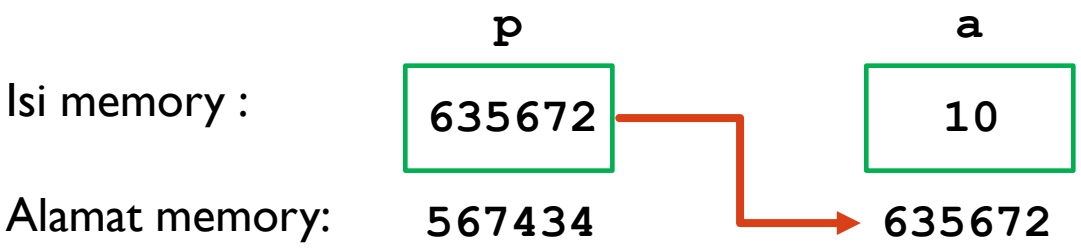
- Variabel yang menunjuk ke suatu variabel lain dengan menyimpan **alamat memory variabel** tersebut
- Contoh:

```
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;
}
```

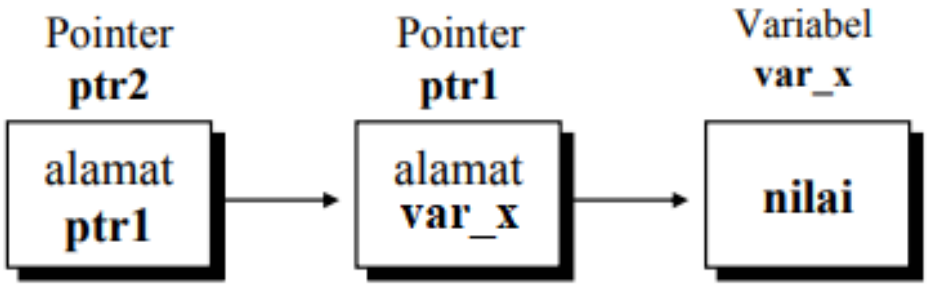
Hasil:

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

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

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

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

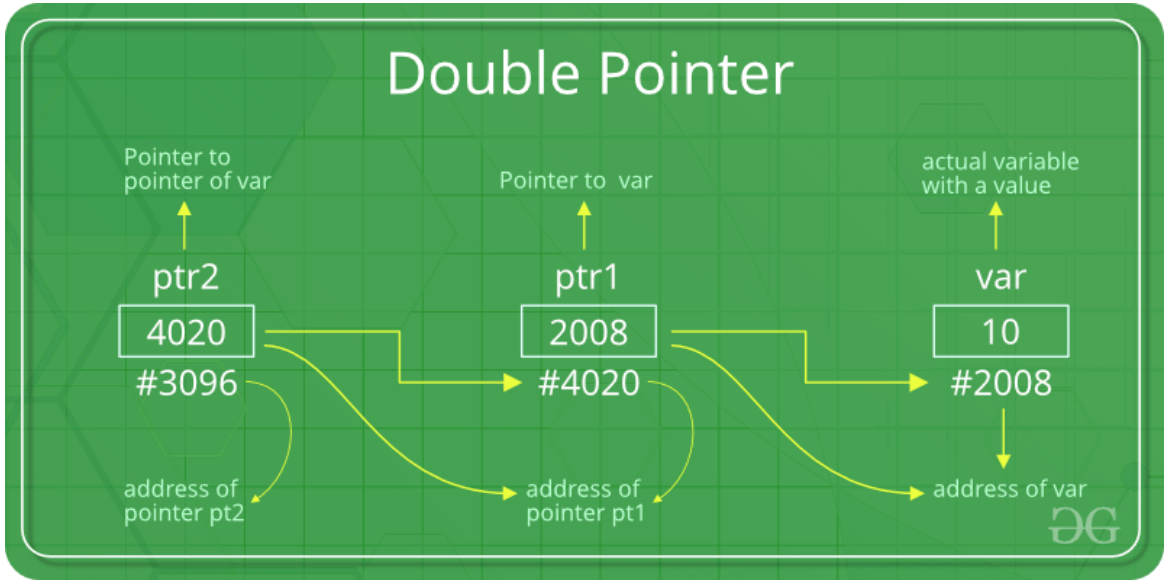


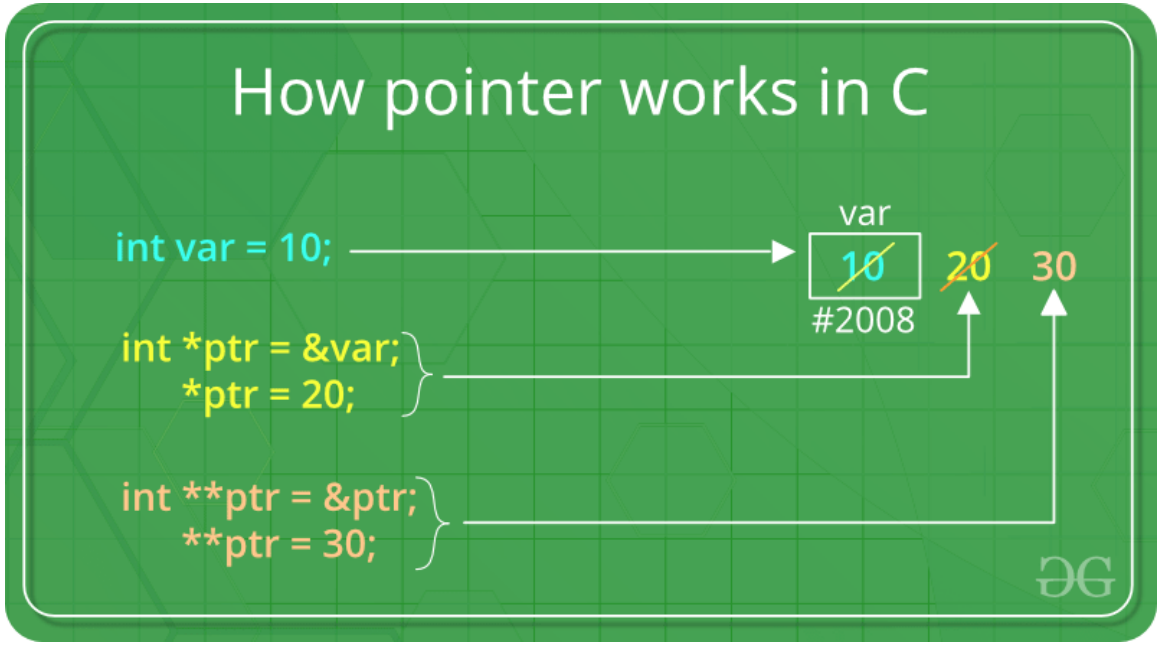
- Deklarasi:

```
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 = 10;
    int *ptr1, **ptr2;

    ptr1 = &var;
    ptr2 = &ptr1;

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

    printf("Alamat var = %d\n", ptr1);
    printf("Alamat ptr1 = %d\n", ptr2);
    printf("Alamat ptr2 = %d\n", &ptr2);
}
```

```
Output:
10
10
10
Alamat var = 29359180
Alamat ptr1 = 29359168
Alamat ptr2 = 29359160
```

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

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

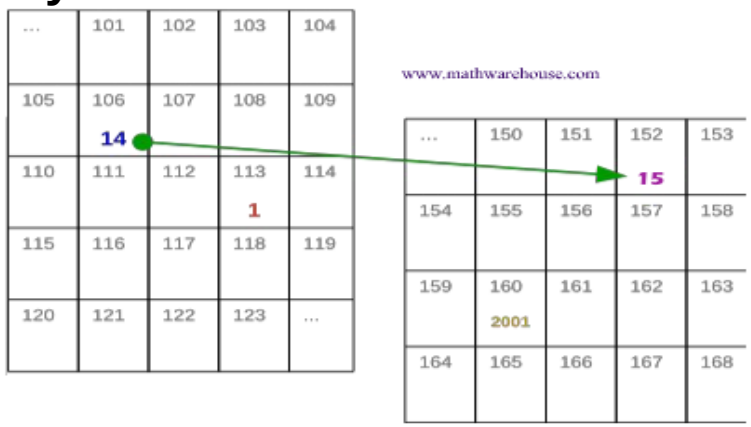
Pointer Sebagai Parameter Fungsi

```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

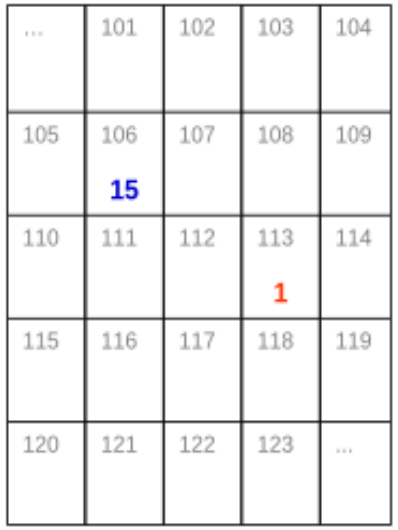



```c
#include <stdio.h>
int tambahsatu(int *a);
int main()
{
    int m = 14;
    printf("m = %d\n", m);

    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

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?

ARRAY

Array

- Format deklarasi dalam bahasa C: **`tipe_data nama_variabel [jumlah_element]`**

- Deklarasi Array 1 dimensi
  - **`float bilangan[100]`**
  - **`char huruf[3]`**
- Mengisi nilai Array:

```
int bilangan[5];

bilangan[0] = 6;
bilangan[1] = 9;
bilangan[2] = -8;
bilangan[3] = 24;
bilangan[4] = -99;
```

atau

```
int bilangan[5] = {6, 9, -8, 24, -99};
```

Contoh Deklarasi dan Mengisi Nilai Array

```c
#include <stdio.h>

int main(void)
{
  int bilangan[5];

  bilangan[0] = 6;
  bilangan[1] = 9;
  bilangan[2] = -8;
  bilangan[3] = 24;
  bilangan[4] = -99;

  printf("Isi array bilangan pertama: %d \n",bilangan[0]);
  printf("Isi array bilangan kedua: %d \n",bilangan[1]);
  printf("Isi array bilangan ketiga: %d \n",bilangan[2]);
  printf("Isi array bilangan keempat: %d \n",bilangan[3]);
  printf("Isi array bilangan kelima: %d \n",bilangan[4]);

  return 0;
}
```

Atau

```c
#include <stdio.h>

int main(void)
{
  int bilangan[5] = {6, 9, -8, 24, -99};

  printf("Isi array bilangan pertama: %d \n",bilangan[0]);
  printf("Isi array bilangan kedua: %d \n",bilangan[1]);
  printf("Isi array bilangan ketika: %d \n",bilangan[2]);
  printf("Isi array bilangan keempat: %d \n",bilangan[3]);
  printf("Isi array bilangan kelima: %d \n",bilangan[4]);

  return 0;
}
```

Deklarasi Array tanpa mendefinisikan jumlah elemen

```c
#include <stdio.h>

int main(void)
{
  char kumpulan_huruf[] = {'a','C','x'};

  printf("Isi array kumpulan_huruf: ");
  printf("%c, %c, %c \n",kumpulan_huruf[0],kumpulan_huruf[1],kumpulan_huruf[2]);

  return 0;
}
```

Contoh Mengubah Nilai Elemen Array

```c
#include <stdio.h>

int main(void)
{
  float pecahan[] = {3.14,-99.01,0.002};

  printf("Isi array pecahan: ");
  printf("%.3f, %.3f, %.3f \n",pecahan[0],pecahan[1],pecahan[2]);
  printf(" \n");

  pecahan[1] = 9.123;
  pecahan[2] = 12.9925;

  printf("Isi array pecahan: ");
  printf("%.3f, %.3f, %.3f \n",pecahan[0],pecahan[1],pecahan[2]);

  return 0;
}
```

Output:
Isi array pecahan: 3.140, -99.010, 0.002

Isi array pecahan: 3.140, 9.123, 12.993

Array 2 Dimensi

- Array 2 dimensi: sebutan untuk array yang penomoran index-nya menggunakan 2 buah angka

```c
#include <stdio.h>

int main(void)
{
  int bilangan[2][2];

  bilangan[0][0] = 100;
  bilangan[0][1] = 101;
  bilangan[1][0] = 110;
  bilangan[1][1] = 111;

  printf("Isi array bilangan: \n");
  printf("%d, %d \n",bilangan[0][0],bilangan[0][1]);
  printf("%d, %d \n",bilangan[1][0],bilangan[1][1]);

  return 0;
}
```

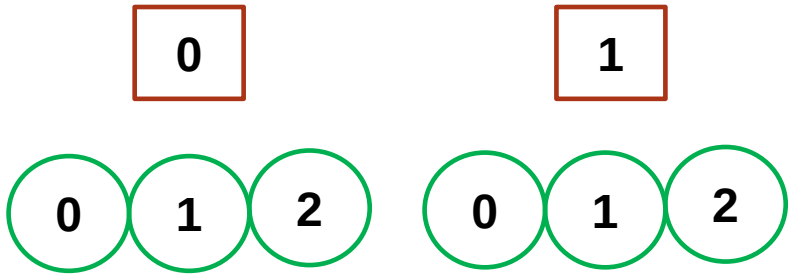
```c
#include <stdio.h>

int main(void)
{
  int matrix[2][3] = {{1,2,3},{7,8,9}};

  printf("Isi array matrix: \n");
  printf("%d %d %d \n", matrix[0][0], matrix[0][1], matrix[0][2]);
  printf("%d %d %d \n", matrix[1][0], matrix[1][1], matrix[1][2]);

  return 0;
}
```

Index Array Multi Dimensi




```
int matrix[2][3] = {{1,2,3},{7,8,9}};
```

Berapakah nilai `matrix[1][0]` ?

Array 3 Dimensi dan String (Array of char)

- Array 3 dimensi

```c
#include <stdio.h>

int main(void)
{
  int matrix[2][3][4] =
  {
     { {7, 4, 12, 3}, {-9, 29, 3, 11}, {6, 34, 23, 20} },
     { {6, 15, 1, 5}, {17, 8, -3, 15}, {99, -1, 44, 9} }
  };

  printf("Isi matrix[0][0][0]: %d \n",matrix[0][0][0]);
  printf("Isi matrix[0][1][0]: %d \n",matrix[0][1][0]);
  printf("Isi matrix[1][1][3]: %d \n",matrix[1][1][3]);
  printf("Isi matrix[1][2][3]: %d \n",matrix[1][2][3]);

  return 0;
}
```

Output:

```
Isi matrix[0][0][0]: 7
Isi matrix[0][1][0]: -9
Isi matrix[1][1][3]: 15
Isi matrix[1][2][3]: 9
```

- String

```c
char c[] = "abcd";

char c[50] = "abcd";

char c[] = {'a', 'b', 'c', 'd', '\0'};

char c[5] = {'a', 'b', 'c', 'd', '\0'};
```

```c
#include <stdio.h>
int main()
{
      char name[20];
      printf("Enter name: ");
      scanf("%s", name);
      printf("Your name is %s.", name);
      return 0;
}
```

Format output untuk string

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:

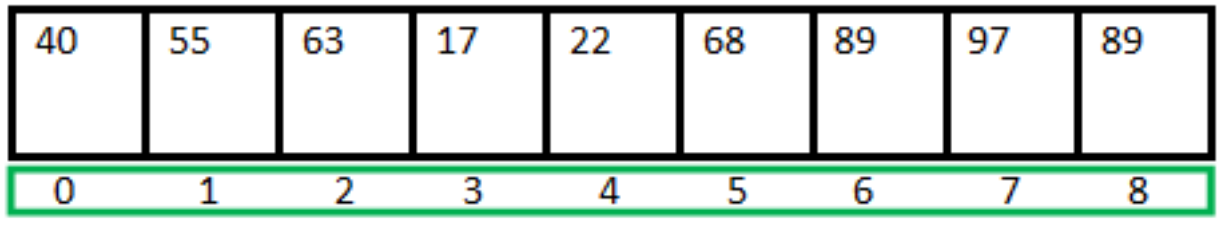
```
Value of var[0] = 10
Value of var[1] = 100
Value of var[2] = 200
```

ALOKASI MEMORI DINAMIS

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
```

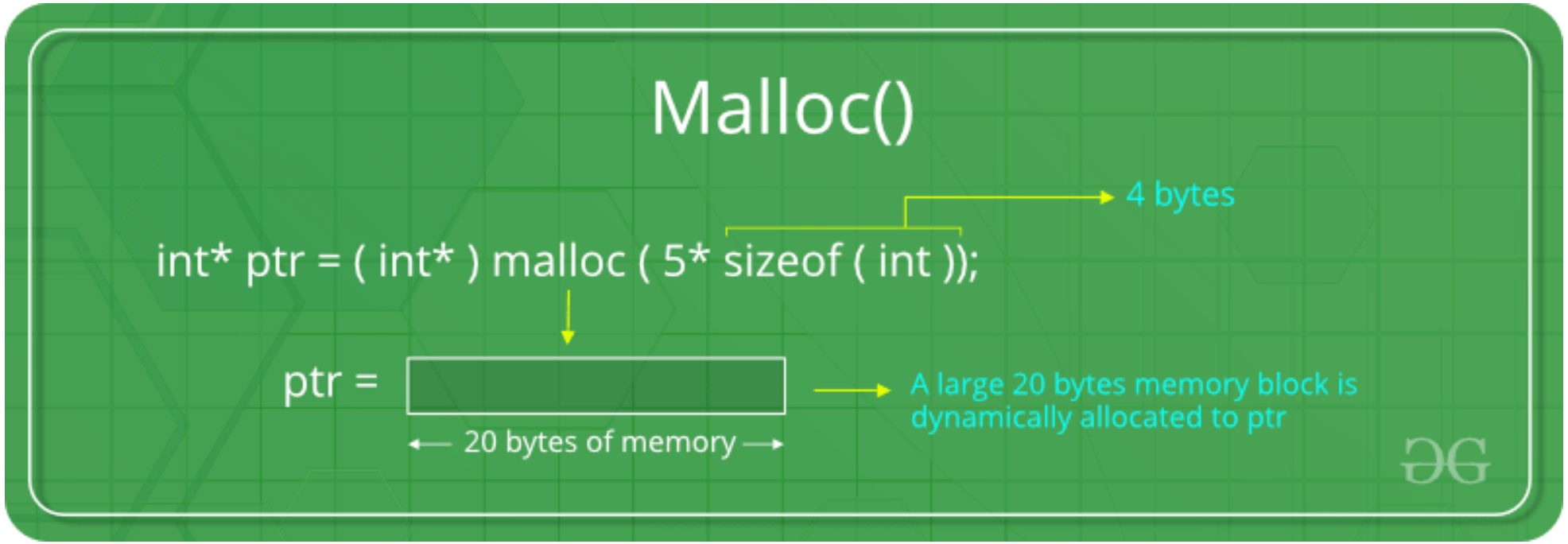
Fungsi `malloc()`

- “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,
```

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;
```

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()`

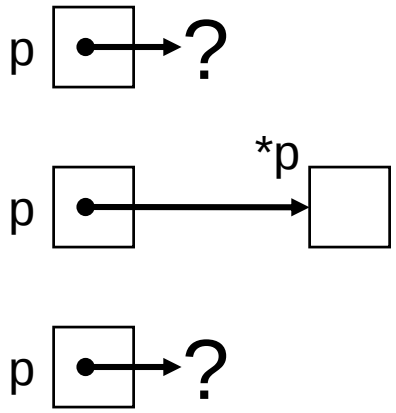
```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    char *pblok;
    pblok = (char *) malloc(500 * sizeof(char));
    if (pblok == NULL)
        printf("Error on malloc");
    else {
        printf("OK, alokasi memori sudah dilakukan\n");
        printf("------\n");
        free(pblok);
        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 3

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

int main()
{
    int *p, *q;
    p = (int*) malloc(sizeof(int));
    q = (int*) malloc(sizeof(int));

    *p = 3;

    free(q); // q di-free-kan dulu sebelum q = p
    q = p;

    printf("Nilai p = %d\n", *p);
    printf("Nilai q = %d\n", *q);

    printf("%d\n", p);
    printf("%d\n", q);

    free(p);
    return 0;
}
```

- Apakah output dari program tersebut?

Latihan 4

```c
#include <stdlib.h>

int main()
{
    int *ptr = (int *) malloc(sizeof(int));

    return 0;
}
```

```c
#include <stdlib.h>

int main()
{
    int *ptr = (int *) malloc(sizeof(int));
    free(ptr);
    return 0;
}
```

- Mana yang lebih tepat?

Latihan 5

```c
#include <stdio.h>
#include <stdlib.h>
int main () {
  int *p, *q;
  p = (int*) malloc(sizeof(int));
  q = p;
  *q = 3;

  printf("%d %d", *p, *q);

  free(p);

  printf("%d %d", *p, *q);

  return 0;

}
```

- Apa output dari program tersebut?

Latihan 6

- Apa kegunaan fungsi calloc() dan realloc? Apakah perbedaannya dengan fungsi malloc()?

TERIMA KASIH