

Modul 2 PENGENALAN BAHASA C++ (BAGIAN KEDUA)

TUJUAN PRAKTIKUM

1. Memahami penggunaan *pointer* dan alamat memori
2. Mengimplementasikan fungsi dan prosedur dalam program

2.1 Array

Array merupakan kumpulan data dengan nama yang sama dan setiap elemen bertipe data sama. Untuk mengakses setiap komponen / elemen *array* berdasarkan indeks dari setiap elemen.

2.1.1 Array Satu Dimensi

Adalah *array* yang hanya terdiri dari satu larik data saja. Cara pendeklarasian *array* satu dimensi:

```
tipe_data nama_var[ukuran]
```

Keterangan:

Tipe_data → menyatakan jenis elemen *array* (int, char, float, dll).

Ukuran → menyatakan jumlah maksimum *array*.

Contoh:

```
int nilai[10];
```

Menyatakan bahwa *array* nilai mengandung 10 elemen dan bertipe *integer*.

Dalam C++ data *array* disimpan dalam memori pada lokasi yang berurutan. Elemen pertama memiliki indeks 0 dan elemen selanjutnya memiliki indeks 1 dan seterusnya. Jadi jika terdapat *array* dengan 5 elemen maka elemen pertama memiliki indeks 0 dan elemen terakhir memiliki indeks 4.

```
nama_var[indeks]
```

nilai[5] → elemen ke-5 dari *array* nilai. Contoh memasukkan data ke dalam *array*:

```
nilai[4] = 90;          /*memasukkan 90 ke dalam array nilai indeks ke-4*/  
cin << nilai[4]        /*membaca input-an dari keyboard*/
```

2.1.2 Array Dua Dimensi

Bentuk *array* dua dimensi ini mirip seperti tabel. Jadi *array* dua dimensi bisa digunakan untuk menyimpan data dalam bentuk tabel. Terbagi menjadi dua bagian, dimensi pertama dan dimensi kedua. Cara akses, deklarasi, inisialisasi, dan menampilkan data sama dengan *array* satu dimensi, hanya saja indeks yang digunakan ada dua.

Contoh:

```
int data_nilai[4][3];  
nilai[2][0] = 10;
```

	0	1	2
0			
1			
2	10		
3			

Gambar 2-1 Ilustrasi Array Dua Dimensi

2.1.3 Array Berdimensi Banyak

Merupakan *array* yang mempunyai indeks banyak, lebih dari dua. Indeks inilah yang menyatakan dimensi *array*. *Array* berdimensi banyak lebih susah dibayangkan, sejalan dengan jumlah dimensi dalam *array*.

Cara deklarasi:

```
tipe_data nama_var[ukuran1][ukuran2]...[ukuran-N];
```

Contoh:

```
int data_rumit[4][6][6];
```

Array sebenarnya masih banyak pengembangannya untuk penyimpanan berbagai betuk data, pengembangan *array* misalnya untuk *array* tak berukuran.

2.2 Pointer

2.2.1 Data dan Memori

Semua data yang ada digunakan oleh program komputer disimpan di dalam memori (RAM) komputer. Memori dapat digambarkan sebagai sebuah *array* 1 dimensi yang berukuran sangat besar. Seperti layaknya *array*, setiap *cell memory* memiliki “indeks” atau “alamat” unik yang berguna untuk identitas yang biasa kita sebut sebagai “**address**”

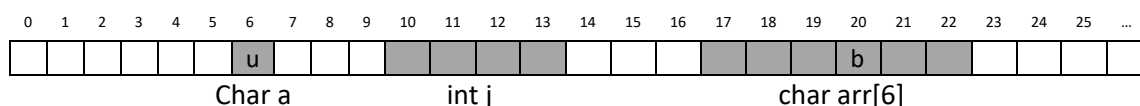
Saat program berjalan, Sistem Operasi (OS) akan mengalokasikan *space memory* untuk setiap variabel, objek, atau *array* yang kita buat. Lokasi pengalokasian memori bisa sangat teracak sesuai proses yang ada di dalam OS masing-masing. Perhatikan ilustrasi berikut



Gambar 2-2 Ilustrasi Memory

Digambarkan sebuah *memory* diasumsikan setiap *cell* menyimpan 1 *byte* data. Pada saat komputer pertama kali berjalan keadaan memori adalah kosong. Saat variabel dideklarasikan, OS akan mencari *cell* kosong untuk dialokasikan sebagai memori variabel tersebut.

```
char a;  
int j;  
char arr[6];  
arr[3] = 'b';  
a = 'u';
```



Gambar 2-3 Ilustrasi Alokasi Memory

Pada contoh di atas variabel *a* dialokasikan di *memory* alamat x6, variabel *j* dialokasikan di alamat x10-13, dan variabel *arr* dialokasikan di alamat x17-22. Nilai variabel yang ada di dalam memori dapat dipanggil menggunakan alamat dari *cell* yang menyimpannya. Untuk mengetahui alamat memori tempat di mana suatu variabel dialokasikan, kita bisa menggunakan keyword “&” yang ditempatkan di depan nama variabel yang ingin kita cari alamatnya.

C++	Output	Keterangan
<pre>Cout << a << endl; Cout << &a << endl; Cout << j << endl; Cout << &j << endl; Cout << &(arr[4]) << endl;</pre>	<pre>'u' x6 0 x10 x21</pre>	<pre>Nilai variabel a Alamat variabel a Nilai variabel j Alamat variabel j Alamat variabel arr[4]</pre>

2.2.2 Pointer dan Alamat

Variabel *pointer* merupakan dasar tipe variabel yang berisi *integer* dalam format heksadesimal. *Pointer* digunakan untuk menyimpan alamat memori variabel lain sehingga *pointer* dapat mengakses nilai dari variabel yang alamatnya ditunjuk.

Cara pendeklarasian variabel *pointer* adalah sebagai berikut:

```
type *nama_variabel;
```

Contoh:

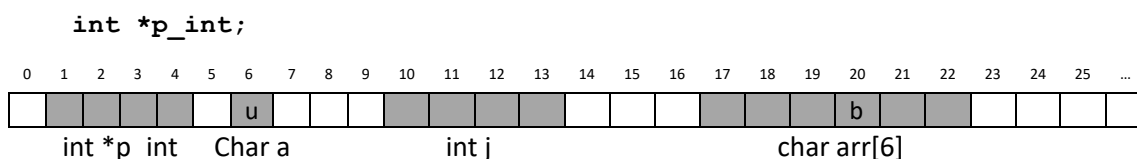
```
int *p_int;
/* p_int merupakan variabel pointer yang menunjuk ke data bertipe int */
```

Agar suatu *pointer* menunjuk ke variabel lain, mula-mula *pointer* harus diisi dengan alamat memori yang ditunjuk.

`p_int = &j;`
Pernyataan di atas berarti bahwa `p_int` diberi nilai berupa alamat dari variabel `j`. Setelah pernyataan tersebut di eksekusi maka dapat dikatakan bahwa `p_int` menunjuk ke variabel `j`. Jika suatu variabel sudah ditunjuk oleh *pointer*. Maka, variabel yang ditunjuk oleh *pointer* dapat diakses melalui variabel itu sendiri ataupun melalui *pointer*.

Untuk mendapatkan nilai dari variabel yang ditunjuk *pointer*, gunakan tanda `*` di depan nama variabel *pointer*

Pointer juga merupakan variabel, karena itu *pointer* juga akan menggunakan *space memory* dan memiliki alamat sendiri

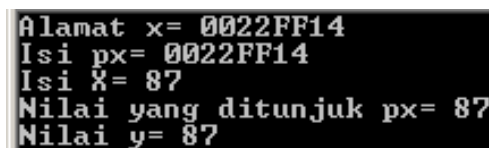


Gambar 2-4 Ilustrasi Alokasi Pointer

C++	Output	Keterangan
<pre>int j,k; j =10; int *p_int; p_int = &j; cout<< j << endl; cout<< &j << endl; cout<< p_int << endl; cout<< &p_int << endl; cout<< *p_int << endl; k = *p_int; cout << k << endl;</pre>	<pre>10 X6 X6 X1 10 10</pre>	<pre>Nilai variabel j Alamat variabel j Nilai variabel p_int Alamat variabel p_int Nilai variabel yang ditunjuk p_int Nilai variabel k</pre>

Berikut ini contoh program sederhana menggunakan *pointer*:

```
1  #include <iostream>
2  #include <conio.h>
3  using namespace std;
4
5  int main(){
6      int x,y; //x dan y bertipe int
7      int *px; //px merupakan variabel pointer menunjuk ke variabel int
8      x =87;
9      px=&x;
10     y=*px;
11     cout<<"Alamat x= "<<&x<<endl;
12     cout<<"Isi px= "<<px<<endl;
13     cout<<"Isi X= "<<x<<endl;
14     cout<<"Nilai yang ditunjuk px= "<<*px<<endl;
15     cout<<"Nilai y= "<<y<<endl;
16     getch();
17     return 0;
18 }
```



```
Alamat x= 0022FF14
Isi px= 0022FF14
Isi X= 87
Nilai yang ditunjuk px= 87
Nilai y= 87
```

Gambar 2-5 Output Pointer

2.2.3 Pointer dan Array

Ada keterhubungan yang kuat antara *array* dan *pointer*. Banyak operasi yang bisa dilakukan dengan *array* juga bisa dilakukan dengan *pointer*. Pendeklarasian *array*: `int a[10];`

Mendefinisikan *array* sebesar 10, kemudian blok dari objek *array* tersebut diberi nama `a[0],a[1],a[2],.....a[9]`.



Gambar 2-6 Array

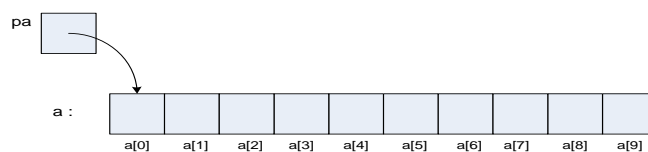
Notasi `a[i]` akan merujuk elemen ke-*i* dari *array*. Jika *pa* merupakan *pointer* yang menunjuk variabel bertipe *integer*, yang di deklarasi sebagai berikut :

```
int *pa;
```

maka pernyataan :

```
pa=&a[0];
```

akan membuat *pa* menunjuk ke alamat dari elemen ke-0 dari variabel *a*. Sehingga, *pa* akan mengandung alamat dari `a[0]`.



Gambar 2-7 Pointer dan Array

Sekarang, pernyataan :

```
x=*pa;
```

Akan menyalinkan isi dari $a[0]$ ke variabel x .

Jika pa akan menunjuk ke elemen tertentu dari *array*, maka pendefinisian $pa + 1$ akan menunjuk elemen berikutnya, $pa + i$ akan menunjuk elemen ke- i setelah pa , sedangkan $pa - i$ akan menunjuk elemen ke- i sebelum pa sehingga jika pa menunjuk ke $a[0]$ maka $*(pa + 1)$ akan mengandung isi elemen ke $a[1]$. $pa + i$ merupakan alamat dari $a[i]$, dan $*(pa + i)$ akan mengandung isi dari elemen $a[i]$.

```
1  #include <iostream>
2  #include <conio.h>
3  #define MAX 5
4  using namespace std;
5
6  int main(){
7      int i,j;
8      float nilai_total, rata_rata;
9      float nilai[MAX];
10     static int nilai_tahun[MAX][MAX]=
11     {
12         {0,2,2,0,0},
13         {0,1,1,1,0},
14         {0,3,3,3,0},
15         {4,4,0,0,4},
16         {5,0,0,0,5}
17     };
18     /*inisialisasi array dua dimensi */
19     for (i=0; i<MAX; i++){
20         cout<<"masukkan nilai ke-"<<i+1<<endl;
21         cin>>nilai[i];
22     }
23     cout<<"\ndata nilai siswa :\n";
24     /*menampilkan array satu dimensi */
25     for (i=0; i<MAX; i++){
26         cout<<"nilai k-"<<i+1<<"=" <<nilai[i]<<endl;
27     }
28     cout<<"\n nilai tahunan : \n";
29
30     /* menampilkan array dua dimensi */
31     for(i=0; i<MAX; i++){
32         for(j=0; j<MAX; j++)
33             cout<<nilai_tahun[i][j]<<endl;
34     }
35     getch();
36     return 0;
37 }
```

2.2.4 Pointer dan String

A. String

String merupakan bentuk data yang sering digunakan dalam bahasa pemrograman untuk mengolah data teks atau kalimat. Dalam bahasa C pada dasarnya *string* merupakan kumpulan dari karakter atau *array* dari karakter.

Deklarasi variabel *string*:

```
char nama[50];
```

50 → menyatakan jumlah maksimal karakter dalam *string*.

Memasukkan data *string* dari keyboard:

```
gets(nama_array);
```

contoh: `gets(nama);`

jika menggunakan `cin()`:

contoh: `cin>>nama;`

Inisialisasi *string*:

```
char nama[] = {'s','t','r','u','k','d','a','t','\0'};
```

Merupakan variabel nama dengan isi data *string* "strukdat".

Bentuk inisialisasi yang lebih singkat:

```
char nama[] = "strukdat";
```

Menampilkan *string* bisa menggunakan `puts()` atau `cout()`:

```
puts(nama);  
cout << nama;
```

Untuk mengakses data *string* sepertihalnya mengakses data pada *array*, pengaksesan dilakukan per karakter sesuai dengan indeks setiap karakter dalam *string*.

Contoh:

```
Cout<<nama[3]; /*menampilkan karakter ke-3 dari string*/
```

B. Pointer dan String

Sesuai dengan penjelasan di atas, misalkan ada *string*:

"I am string"

Merupakan *array* dari karakter. Dalam representasi internal, *array* diakhiri dengan karakter '\0' sehingga program dapat menemukan akhir dari program. Panjang dari *storage* merupakan panjang dari karakter yang ada dalam tanda petik dua ditambah satu. Ketika karakter *string* tampil dalam sebuah program maka untuk mengaksesnya digunakan *pointer* karakter. Standar *input/output* akan menerima *pointer* dari awal karakter *array* sehingga konstanta *string* akan diakses oleh *pointer* mulai dari elemen pertama.

Jika *pmessage* di deklarasikan:

```
char *pmessage;
```

Maka pernyataan berikut:

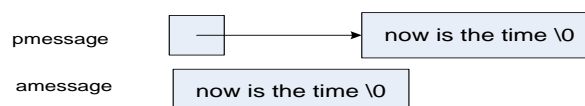
```
pmessage = "now is the time";
```

Akan membuat *pmessage* sebagai *pointer* pada karakter *array*. Ini bukan copy *string*, hanya *pointer* yang terlibat. C tidak menyediakan operator untuk memproses karakter *string* sebagai sebuah unit.

Ada perbedaan yang sangat penting diantara pernyataan berikut:

```
char amessage[] = "now is the time"; //merupakan array  
char *pmessage = "now is the time"; //merupakan pointer
```

Variabel *amessage* merupakan sebuah *array*, hanya cukup besar untuk menampung karakter-karakter sequence tersebut dan karakter null '\0' yang menginisialisasinya. Tiap-tiap karakter dalam *array* bisa saja berubah tapi variabel *amessage* akan selalu menunjuk kepada *storage* yang sama. Di sisi lain, *pmessage* merupakan *pointer*, diinisialisasikan menunjuk konstanta *string*, *pointer* bisa di modifikasi untuk menunjuk kemanapun, tapi hasilnya tidak akan terdefinisi jika kamu mencoba untuk mengubah isi *string*.



2.3 Fungsi

Fungsi merupakan blok dari kode yang dirancang untuk melaksanakan tugas khusus dengan tujuan:

1. Program menjadi terstruktur, sehingga mudah dipahami dan mudah dikembangkan. Program dibagi menjadi beberapa modul yang kecil.
2. Dapat mengurangi pengulangan kode (duplikasi kode) sehingga menghemat ukuran program.

Pada umumnya fungsi memerlukan masukan yang dinamakan sebagai parameter. Masukan ini selanjutnya diolah oleh fungsi. Hasil akhir fungsi berupa sebuah nilai (nilai balik fungsi).

Bentuk umum sebuah fungsi:

```
tipe_keluaran nama_fungsi(daftar_parameter) {
    blok_pernyataan_fungsi ;
}
```

Jika penentu_tipe fungsi merupakan tipe dari nilai balik fungsi, bila tidak disebutkan maka akan dianggap (default) sebagai int.

Algoritma	C++
Program coba_fungsi Kamus x,y,z : integer function max3(input: a,b,c : integer) : integer Algoritma input(x,y,z) output(max3(x,y,z)) function max3(input:a,b,c : integer) : integer kamus temp_max : integer algoritma temp_max ← a if (b>temp_max) then temp_max ← b if (c>temp_max) then temp_max ← c → temp_max	<pre>#include <conio.h> #include <iostream> #include <stdlib.h> using namespace std; int maks3(int a, int b, int c); /*mendeklarasikan prototype fungsi */ int main(){ system("cls"); int x,y,z; cout<<"masukkan nilai bilangan ke-1 ="; cin>>x; cout<<"masukkan nilai bilangan ke-2 ="; cin>>y; cout<<"masukkan nilai bilangan ke-3 ="; cin>>z; cout<<"nilai maksimumnya adalah =" <<maks3(x,y,z); getch(); return 0; } /*badan fungsi */ int maks3(int a, int b, int c){ /* deklarasi variabel lokal dalam fungsi */ Int temp_max =a; if(b>temp_max) temp_max=b; if(c>temp_max) temp_max=c; return (temp_max); }</pre>

2.4 Prosedur

Dalam C sebenarnya tidak ada prosedur, semua berupa fungsi, termasuk main() pun adalah sebuah fungsi. Jadi prosedur dalam C merupakan fungsi yang tidak mengembalikan nilai, biasa diawali dengan kata kunci void di depan nama prosedur.

Bentuk umum sebuah prosedur:

```
void nama_prosedur (daftar_parameter) {
    blok_pernyataan_prosedur ;
}
```

Algoritma	C++
Program coba_prosedur Kamus jum : integer procedure tulis(input: x: integer) Algoritma input(jum) tulis(jum) procedure tulis(input: x: integer) kamus i : integer algoritma i traversal [1..x] output("baris ke-",i+1)	<pre>#include <iostream> #include <conio.h> #include <stdlib.h> using namespace std; /*prototype fungsi */ void tulis(int x); int main() { System("cls"); int jum; cout << " jumlah baris kata="; cin >> jum; tulis(jum); getch(); return 0; } /*badan prosedur*/ void tulis(int x){ for (int i=0;i<x;i++) cout<<"baris ke-"<<i+1<<endl; }</pre>

2.5 Parameter Fungsi

2.5.1 Paramater Formal dan Parameter Aktual

Parameter formal adalah variabel yang ada pada daftar parameter ketika mendefinisikan fungsi. Pada fungsi maks3() contoh diatas, a, b dan merupakan parameter formal.

```
float perkalian (float x, float y) {
    return (x*y);
}
```

Pada contoh di atas x dan y adalah parameter formal.

Adapun parameter aktual adalah parameter (tidak selamanya menyatakan variabel) yang dipakai untuk memanggil fungsi.

```
X = perkalian(a, b);
Y = perkalian(10,30);
```

Dari pernyataan diatas a dan b merupakan parameter aktual, begitu pula 10 dan 30. parameter aktual tidak harus berupa variabel, melainkan bisa berupa konstanta atau ungkapan.

2.5.2 Cara melewati Parameter

A. Pemanggilan dengan Nilai (*call by value*)

Pada pemanggilan dengan nilai, nilai dari parameter aktual akan disalin kedalam parameter formal, jadi parameter aktual tidak akan berubah meskipun parameter formalnya berubah. Untuk lebih jelasnya perhatikan contoh berikut:

Algoritma	C++
<pre>Program coba_parameter_by_value Kamus a,b : integer procedure tukar(input: x,y : integer) Algoritma a ← 4 b ← 6 output(a,b) tukar(a,b) output(a,b) procedure tukar(input:x,y : integer) kamus temp : integer algoritma temp ← x x ← y y ← temp output(x,y)</pre>	<pre>#include <iostream> #include <conio.h> #include <stdlib.h> using namespace std; /*prototype fungsi */ void tukar(int x, int y); int main () { int a, b; system("cls"); a=4; b=6; cout << "kondisi sebelum ditukar \n"; cout << " a = "<<a<<" b = "<<b<<endl; tukar(a,b); printf("kondisi setelah ditukar \n"); cout << " a = "<<a<<" b = "<<b<<endl; getch(); return 0; } void tukar (int x, int y) { int temp; temp = x; x = y; y = temp; cout << "nilai akhir pada fungsi tukar \n"; cout << " x = "<<x<<" y = "<<y<<endl; }</pre>

Hasil eksekusi :

Kondisi sebelum tukar

a = 4, b = 6

Nilai akhir pada fungsi tukar

x = 6, y = 4

Kondisi setelah tukar

a = 4, b = 6

Jelas bahwa pada pemanggilan fungsi tukar, yang melewati variabel a dan b tidak merubah nilai dari variabel tersebut. Hal ini dikarenakan ketika pemanggilan fungsi tersebut nilai dari a dan b disalin ke variabel formal yaitu x dan y.

B. Pemanggilan dengan Pointer (*call by pointer*)

Pemanggilan dengan *pointer* merupakan cara untuk melewati alamat suatu variabel ke dalam suatu fungsi. Dengan cara ini dapat merubah nilai dari variabel aktual yang dilewatkan ke dalam fungsi. Jadi cara ini dapat merubah variabel yang ada diluar fungsi.

Cara penulisan :

```
tukar(int *px, int *py) {
  int temp;
  temp = *px;
```

```

    *px = *py;
    *py = temp;
    ... ..
}

```

Cara pemanggilan:

```
tukar(&a, &b);
```

Pada ilustrasi tersebut, *px merupakan suatu variabel *pointer* yang menunjuk ke suatu variabel *integer*. Pada pemanggilan fungsi tukar(), &a dan &b menyatakan "alamat a" dan "alamat b". dengan cara diatas maka variabel yang diubah dalam fungsi tukar() adalah variabel yang dilewatkan dalam fungsi itu juga, karena yang dilewatkan dalam fungsi adalah alamat dari variabel tersebut, jadi bukan sekedar disalin.

C. Pemanggilan dengan Referensi (Call by Reference)

Pemanggilan dengan referensi merupakan cara untuk melewatkan alamat suatu variabel kedalam suatu fungsi. Dengan cara ini dapat merubah nilai dari variabel aktual yang dilewatkan ke dalam fungsi. Jadi cara ini dapat merubah variabel yang ada diluar fungsi. Cara penulisan :

```

tukar(int &px, int &py) {
    int temp;
    temp = px;
    px = py;
    py = temp;
    ... ..
}

```

Cara pemanggilan:

```
tukar(a, b);
```

untuk melewatkan nilai dengan referensi, argumen dilalui ke fungsi seperti nilai lain. Jadi pada akhirnya, harus mendeklarasikan di parameter awal, serta untuk pemanggilan tidak perlu menggunakan paramter tambahan seperti pada *call by pointer*.

Algoritma	C++
Program coba_parameter_by_reference Kamus a,b : integer procedure tukar(input/output: x,y : integer) Algoritma a ← 4 b ← 6 output(a,b) tukar(a,b) output(a,b) procedure tukar(input/output :x,y : integer) kamus temp : integer algoritma temp ← x x ← y y ← temp output(x,y)	<pre> #include <iostream> #include <conio.h> #include <stdlib.h> using namespace std; /*prototype fungsi */ void tukar(int &x, int &y); int main () { int a, b; system("cls"); a=4; b=6; cout << "kondisi sebelum ditukar \n"; cout << " a = "<<a<<" b = "<<b<<endl; tukar(a,b); printf("kondisi setelah ditukar \n"); cout << " a = "<<a<<" b = "<<b<<endl; getch(); return 0; } void tukar (int &x, int &y) { int temp; temp = x; x = y; y = temp; cout<< "nilai akhir pada fungsi tukar \n"; cout << " x = "<<x<<" y = "<<y<<endl; </pre>

Call By
Reference

	}
--	---

Algoritma	C++
Program coba_parameter_by_pointer Kamus a,b : integer procedure tukar(input/output: x,y : integer) Algoritma a ← 4 b ← 6 output(a,b) tukar(a,b) output(a,b) procedure tukar(input/output :x,y : integer) kamus temp : integer algoritma temp ← x x ← y y ← temp output(x,y)	<pre> #include <iostream> #include <conio.h> #include <stdlib.h> using namespace std; /*prototype fungsi */ void tukar(int *x, int *y); int main () { int a, b; system("cls"); a=4; b=6; cout << "kondisi sebelum ditukar \n"; cout << " a = "<<a<<" b = "<<b<<endl; tukar(&a,&b); printf("kondisi setelah ditukar \n"); cout << " a = "<<a<<" b = "<<b<<endl; getch(); return 0; } void tukar (int *x, int *y) { int temp; temp = *x; *x = *y; *y = temp; cout << "nilai akhir pada fungsi tukar \n"; cout << " x = "<<x<<" y = "<<y<<endl; } </pre> <div data-bbox="1225 539 1394 723" style="border: 1px solid blue; padding: 5px; position: absolute; top: 241px; left: 768px;"> <i>Call by Pointer</i> </div>