

LECTURE NOTES

COMP6599 – Algorithm and Programming

Minggu 5

Sesi 6

Pointer dan Array

LEARNING OUTCOMES

Learning Outcomes (Hasil Pembelajaran) :

Setelah menyelesaikan pembelajaran ini mahasiswa akan mampu :

1. LO 2 : Menerapkan sintaks-sintaks dan fungsi-fungsi bahasa pemrograman C dalam pemecahan masalah.
2. LO 3 : Membuat program dengan menggunakan bahasa C dalam pemecahan masalah.

OUTLINE MATERI (Sub-Topic):Pointers dan Arrays

1. Pointer
2. Array
3. Pointer Constant & Pointer Variable
4. String

Pointers dan Arrays

1. Array

Array adalah suatu tipe data terstruktur yang bertipe data sama dan berjumlah tetap (berdasarkan apa yang ditentukan) dan diberi suatu nama tertentu (sesuai variabel). Elemen-elemen *array* tersusun secara berurutan. Susunan tersebut membuat *array* memiliki alamat yang bersebelahan / berdampingan dalam memori sesuai dengan besar pemakaian memori tipe data yang digunakan. Namun, perlu diingat, walaupun elemen-elemen nya tersusun secara berurutan, *array* tetap dapat diakses secara acak di dalam memori. *Array* juga dapat berupa *array* 1 dimensi, 2 dimensi, bahkan n-dimensi.

Berikut contoh penggambaran *array* pada bahasa C.

0	1	2	3	4	5	6	7	indeks
10	44	2	7	25	56	32	40	value
1d2	1d4	1d6	1d8	1da	2dc	2de	1ed	alamat

Deklarasi Array :

```
data_type array_name[size];
```

contoh :

```
char nama[26];
```

Contoh diatas berarti anda memesan tempat di memori computer **sebanyak 26 tempat** dengan indeks yang dapat digunakan dari **index 0 sampai index 25**.

a. Inisialisasi *array* 1D

Untuk memasukkan suatu nilai kedalam sebuah *array*, dapat dilakukan dengan beberapa cara, salah satu caranya adalah :

```
int list[5] = {2,1,3,7,8};
```

Contoh di atas berarti program akan memasukkan angka 2, 1, 3, 7 dan 8 kedalam sebuah *array* dengan nama *list*.

b. Initializing Multidimensional Arrays

Untuk membuat *array* yang multi dimensi (contohnya pada penggunaan *array* dalam konsep matrix). Dapat dilakukan inisialisasi dengan cara:

```
intvariabel[x][y] = {
    {1, 2, ..., n},
    {1, 2, ..., n},
    ... dst
};
```

Dalam hal ini jumlah array yang dapat digunakan adalah mulai dari index [0][0] sampai dengan index [x-1][y-1]. Contoh:

```
int matrix[3][3] = {
    {11, 12, 13},
    {21, 22, 23},
    {32, 31, 33},
};
```

Contoh di atas berarti program akan memasukkan kumpulan angka yang berjumlah 9 tersebut kedalam *array* matrix yang akan menghasilkan: Matrix [0][0] = 11 Matrix [0][1] = 12 Matrix [0][2] = 13 Matrix [1][0] = 21 Matrix [2][0] = 32 Matrix [2][2] = 33 Dan seterusnya. Matrix yang dapat digunakan adalah mulai dari index [0][0] sampai dengan [2][2] yang bila dihitung akan berjumlah 9 buah.

2. Pointer

Pointer adalah suatu variabel yang berisi alamat memori dari suatu variabel lain. *Pointer* dilambangkan dengan operator “ * ” dan biasanya digunakan untuk memanipulasi isi variabel di dalam memori. Contoh penggunaannya adalah:

```
int a = 5;
int b = 3;
int *p;
p = &a; //1
b = *p; //2
```

Pada potongan program di atas dapat dilihat bahwa terdapat 3 buah variable yaitu a, b dan *pointer* p. Variabel a bernilai 5, variabel b bernilai 3. Pada //1 p menerima alamat dari a, dalam hal ini operator "&" digunakan untuk mengambil alamat dari variabel a, jadi setelah statement //1 dijalankan maka *pointer* p akan memiliki alamat dari a. Untuk mengambil nilai dari alamat yang dipegang oleh *pointer* p dapat dilakukan dengan cara seperti statement //2. Dengan memanggil *p maka nilai dari variabel a (alamat yang dipegang oleh p) akan diambil, dari contoh di atas nilai tersebut diberikan ke variabel b. Seperti penggunaan *array* dalam sebuah variabel, sebuah *pointer* juga dapat menggunakan *array*. Seperti contoh berikut ini:

```
int *ap[10];
```

Dari contoh di atas dapat dilihat bahwa variable ap adalah sebuah kumpulan variabel yang dapat menyimpan 10 alamat dari variabel lain. Index dari *pointer* tersebut dimulai dari *ap[0] sampai dengan *ap[9].

3. *Pointer Constant & Pointer Variable*

Pointer constant adalah *pointer* yang tidak dapat ditetapkan dengan nilai baru pada saat *run-time*. Dalam hal ini *array* merupakan salah satu contoh *pointer constant*.

Berikut contoh *pointer constant* :

```
int B[4]; // B is an Array → pointer constant
```

Pointer variabel adalah *pointer* yang diberikan dengan nilai baru pada saat *run-time*.

Dengan contoh sebagai berikut :

```
int *ptr; //ptr is pointer variable
```

4. *String*

String adalah kumpulan karakter ASCII. Dalam bahasa C terdapat 2 cara mendeklarasikan variable *string*, satu sebagai *pointer to character* dan yang lain sebagai *array of character*. Penanda akhir *string* dilakukan dengan *null character* ('\0').

Berikut contoh *array of character* pada bahasa C:

```
char name[40] = "Amir"; //ok
```

Berikut contoh pointer to character pada bahasa C :

```
char label[] = "Single";
char label2[10] = "Married";
char *labelPtr;

labelPtr = label;
```

String adalah array of character dimana sebuah string ini dapat dimanipulasi layaknya sebuah array. Pada bahasa C menyediakan beberapa *standart function* yang dapat digunakan untuk manipulasi *string*. *Function* ini tersimpan pada *header string.h*.

Berikut beberapa fungsi string manipulasi yang ada pada bahasa C :

a. strlen()

Fungsi ini mengembalikan angka yang menyatakan panjangnya suatu string tanpa menghitung adanya null karakter pada ujung *string*.


Berikut adalah penggambaran dari fungsi *strlen()* pada bahasa C:

```
char c[]={'P', 'r', 'o', 'g', 'r', 'a', 'm', '\0'};
temp=strlen(c);
```

Then, temp will be equal to 7 because, null character '\0' is not counted.

P	r	o	g	r	a	m	\0
---	---	---	---	---	---	---	----

Length of string



Berikut contoh penggunaan fungsi `strlen()` pada bahasa C :

```
#include <stdio.h>
#include <string.h>
int main(){
    char a[20]="Program";
    char b[20]={'P','r','o','g','r','a','m','\0'};
    char c[20];
    printf("Enter string: ");
    gets(c);
    printf("Length of string a=%d \n",strlen(a));
    //calculates the length of string before null character.
    printf("Length of string b=%d \n",strlen(b));
    printf("Length of string c=%d \n",strlen(c));
    return 0;
}
```

Pada program tersebut terlihat bahwa fungsi `strlen()` pada C menghitung banyaknya kata pada suatu variable. Jika dimasukkan kata '*String*' pada program tersebut maka program tersebut akan mengeluarkan panjang karakter dari variable a, b, dan c. Sehingga dari program tersebut akan menghasilkan *output* sebagai berikut :

```
Enter string: String
Length of string a=7
Length of string b=7
Length of string c=6
```

b. `strcpy(s1,s2)`

Fungsi ini menyalin string dari s2 ke s1, s2 dapat berupa variable atau *string constant*. Berikut adalah contoh penggunaan fungsi `strcpy()` pada bahasa C:

```
#include <stdio.h>
#include <string.h>
int main(){
    char a[10],b[10];
    printf("Enter string: ");
    gets(a);
    strcpy(b,a);    //Content of string a is copied to string b.
    printf("Copied string: ");
    puts(b);
    return 0;
}
```

Jika pada program tersebut dimasukkan kata “*Programming Tutorial*” yang kemudian akan disimpan pada variabel a. Kemudian digunakan fungsi *strcpy()* dimana variabel a disalin ke variabel b. Sehingga akan menghasilkan *output* sebagai berikut :

```
Enter string: Programming Tutorial    //variabel a
Copied string: Programming Tutorial  //variabel b
```

c. **strncpy(s1,s2,n)**

Fungsi ini menyalin string dari s2 ke s1 sebanyak n karakter, s2 dapat berupa variable atau *string constant*. Berikut adalah contoh penggunaan fungsi *strncpy()* pada bahasa C :

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

int main()
{
    char src[40];
    char dest[12];

    memset(dest, '\0', sizeof(dest));
    strcpy(src, "This is tutorialspoint.com");
    strncpy(dest, src, 10);

    printf("Final copied string : %s\n", dest);

    return(0);
}
```

Pada program tersebut variable **src** memiliki nilai ‘*This is tutorialspoint.com*’ dimana nilai dari variable **src** ini disalin ke variabel **dest**. Penyalinan dilakukan hanya sebanyak 10 karakter sehingga hasil output dari program tersebut adalah :

```
Final copied string : This is tu
```

d. **strcat(s1,s2)**

Fungsi ini berguna untuk menambahkan nilai string dari s2 keakhir string s1, s2 dapat berupa variable atau *string constant*. Berikut adalah contoh penggunaan fungsi *strcat()* pada bahasa C :


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

int main ()
{
    char src[50], dest[50];

    strcpy(src, "This is source");
    strcpy(dest, "This is destination");

    strcat(dest, src);

    printf("Final destination string : |%s|", dest);

    return(0);
}
```

Pada program tersebut variable **src** memiliki nilai *'This is source'* dan variable **dest** memiliki nilai *'This is destination'*. Jika menggunakan fungsi *strcat* maka variable **src** akan ditambahkan diakhir variable **dest**. Sehingga hasil *output* dari program tersebut adalah :

```
Final destination string : |This is destinationThis is source|
```

e. strncat(s1,s2,n)

Fungsi ini berguna untuk menambahkan n karakter pada string dari s2 keakhir string s1, s2 dapat berupa variable atau *string constant*. Berikut adalah contoh penggunaan fungsi *strcat()* pada bahasa C :

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

int main ()
{
    char src[50], dest[50];

    strcpy(src, "This is source");
    strcpy(dest, "This is destination");

    strncat(dest, src, 15);

    printf("Final destination string : |%s|", dest);

    return(0);
}
```

Pada program tersebut variable **src** memiliki nilai '*This is source*' dan variable **dest** memiliki nilai '*This is destination*'. Jika menggunakan fungsi *strncat* maka variable **src** akan ditambahkan sebanyak 15 karakter diakhir variable **dest**. Sehingga hasil *output* dari program tersebut adalah :

```
Final destination string : |This is destinationThis is source|
```

f. **strcmp(s1,s2)**

Fungsi ini berguna untuk membandingkan antara *string* s1 dan *string* s2, secara alfabetis. Berikut adalah contoh penggunaan fungsi *strcmp()* pada bahasa C :

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

int main ()
{
    char str1[15];
    char str2[15];
    int ret;

    strcpy(str1, "abcdef");
    strcpy(str2, "ABCDEF");

    ret = strcmp(str1, str2);

    if(ret < 0)
    {
        printf("str1 is less than str2");
    }
    else if(ret > 0)
    {
        printf("str2 is less than str1");
    }
    else
    {
        printf("str1 is equal to str2");
    }

    return(0);
}
```

Pada program tersebut dilakukan perbandingan antara 2 variabel yaitu str1 "abcdef" dengan variabel str2 "ABCDEF". Terdapat variable **ret** yang berguna untuk mengembalikan nilai dari fungsi *strcmp()* ini.

Dengan ketentuan nilai fungsi yang dikembalikan sebagai berikut :

- jika nilai kembali < 0 maka **str1** memiliki nilai kurang dari **str2**.
- Jika nilai kembali > 0 maka **str2** memiliki nilai kurang dari **str1**.
- Jika nilai kembali $= 0$ maka **str1** memiliki nilai yang sama dengan **str2**.

Sehingga hasil *output* dari program tersebut adalah :

```
str2 is less than str1
```

SIMPULAN

1. *Array*

Array adalah suatu tipe data terstruktur yang bertipe data sama dan berjumlah tetap (berdasarkan apa yang ditentukan) dan diberi suatu nama tertentu (sesuai variabel).

2. *Pointer*

Pointer adalah suatu variabel yang berisi alamat memori dari suatu variabel lain.

Pointer dilambangkan dengan operator “*” dan biasanya digunakan untuk memanipulasi isi variabel di dalam memori.

3. *Pointer Constant & Pointer Variable*

Pointer constant adalah pointer yang tidak dapat ditetapkan dengan nilai baru pada saat *run-time*.

Pointer variabel adalah pointer yang diberikan dengan nilai baru pada saat *run-time*.

4. *String*

String adalah kumpulan karakter ASCII. Dalam bahasa C terdapat 2 cara mendeklarasikan variabel *string*, satu sebagai *pointer to character* dan yang lain sebagai *array of character*. Terdapat fungsi string manipulasi yang dapat digunakan dalam bahasaC :

- **strlen()**
- **strcpy(s1,s2)**
- **strncpy(s1,s2,n)**
- **strcat(s1,s2)**
- **strncat(s1,s2,n)**
- **strcmp(s1,s2)**

DAFTAR PUSTAKA

1. <https://www.cs.bu.edu/teaching/cpp/string/array-vs-ptr/>
2. <http://www.programiz.com/c-programming/library-function/string.h>
3. http://www.tutorialspoint.com/c_standard_library/string_h.htm
4. Paul J. Dietel, Harvey M. Deitel, . 2010. C : how to program. PEAPH. New Jersey. ISBN:978-0-13-705966-9 Chapter 6 & 7
5. C Programming – Pointers: <http://www.exforsys.com/tutorials/c-language/c-pointers.html>
6. Storing Similar Data Items: <http://aelinik.free.fr/c/ch12.htm>
7. Thompson Susabda Ngoen, 2006. Pengantar Algoritma dengan Bahasa C. Salemba Teknika. ISBN : 979-9549-25-6. Bagian 6.