# Paper 102: Programming & Problem solving through C

## Lecture-16:Unit-II Arrays and Strings

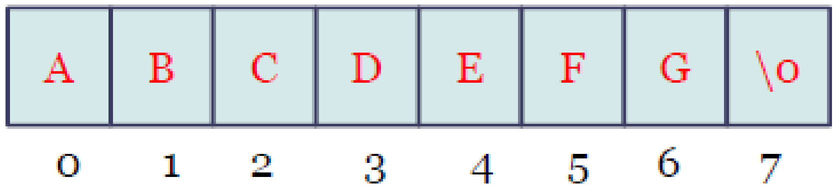
## Arrays & Strings

- This chapter will cover the following topics
  - Array declaration
  - Array manipulation
  - String manipulation
  - Passing arrays to functions
  - Passing strings to functions

## What are Arrays

- A collection of data elements of the same type that are referenced by a common name
- All elements of the array occupy a set of contiguous memory locations
- An index or subscript is used to reference its elements

```
char huruf[8];
```



## One-dimensional Arrays

- One-dimensional array declaration
  - data_type var_name[size];

int nombor[8];

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 11 | 32 | 44 | 21 | 11 | 21 | 33 | 53 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

float gaji[8];

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1.0 | 2.3 | 3.3 | 4.3 | 5.2 | 2.2 | 1.2 | 3.3 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

char huruf[8];

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| A | B | C | D | E | F | G | \0 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

## One-dimensional Arrays

- Array Initialization
  - data_type var_name[size] = {value_list} ;

```
int  nombor[8] = {11, 32, 44, 21, 11, 21, 33, 53 };
```

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 11 | 32 | 44 | 21 | 11 | 21 | 33 | 53 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

```
float  gaji[8]= {1.0, 2.3, 3.3, 4.3, 5.2, 2.2, 1.2, 3.3 };
```

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1.0 | 2.3 | 3.3 | 4.3 | 5.2 | 2.2 | 1.2 | 3.3 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

```
char  huruf[8] = {'A','B','C','D','E','F','G'};
```

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| A | B | C | D | E | F | G | \0 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

**Null character, automatically supply by the compiler**

## Passing One-dimensional Arrays to Functions

- A function can receive the address of an array by using a pointer, a sized array or an unsized array

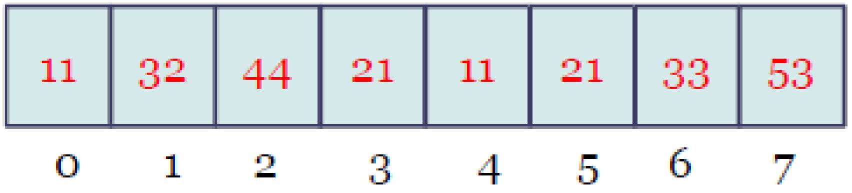
```
void main()
{		double x[5];
		void fn(double *);

		fn(x);
		printf(“%lf”, x[3]);
}
void fn(double *ptr)
{		*ptr[3]=4;
}
```

## Working with One-dimensional Arrays

- Accessing elements of an array
  - Use an index(subscript) to access to a specific element in the array

int nombor[8];



Access one data

```
Printf("Content array withn index =4  %d",number(4));
```

Access all data

```
for (index=0; index<8; index++)
{               printf("Content of the array with index = %d is %d ",index, number[index]);
}
```

## Working with One-dimensional Arrays

- Searching for a value

int nombor[8];

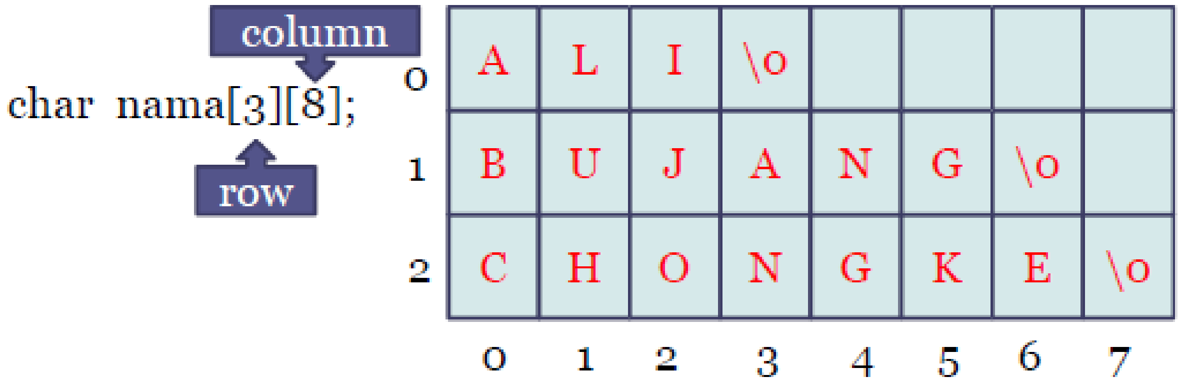
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 11 | 32 | 44 | 21 | 11 | 21 | 33 | 53 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

```
for (index=0; index <= 7; index++)
{
        if (nombor[index] > 30)
                printf("Content of the array more then 30 are %d ", nomber[index]);
}
```

## Two-dimensional Arrays

- Two dimensional arrays have two indexes (subscripts)
- The first subscript is used for rows while the second subscript is used for columns

  - data_type var_name[size1][size2] ;



|   | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| 0 | A | L | I | \0 |  |  |  |  |
| 1 | B | U | J | A | N | G | \0 |  |
| 2 | C | H | O | N | G | K | E | \0 |

## Two-dimensional Arrays

- Array Initialization
    - int nombor[2][3] = {1,2,3,4,5,6};
    - int nombor[2][3] = {{1,2,3},{4,5,6}};

## Two-dimensional Arrays

- Accessing element in the array

```
char  nama[3][8];
```

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| 0 | A | L | I | \o | | | | |
| 1 | B | U | J | A | N | G | \o | |
| 2 | C | H | O | N | G | K | E | \o |

printf(“%s”, nama[0][2]); ← Display “I”

printf(“%s”, nama[2][5]); ← Display “K”

## Two-dimensional Arrays

- Accessing element in the array

char nama[3][8];

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 0 | A | L | I | \0 | | | | |
| 1 | B | U | J | A | N | G | \0 | |
| 2 | C | H | O | N | G | K | E | \0 |
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

```
for (row=0; row <=2; row++)
{
        for (column=0; column <=7; column++)
        {
                printf("%s", nama[row][column]);
        }
cout << "\n";
}
```

## Multidimensional Arrays

- When an array has two or more dimensions, it is called a multi-dimensional array
- Multidimensional arrays declaration
  - data_type var_name[size1][size2] [size3];
  - Example
    - int paksi[4][5][6];

## Strings

- A string is any sequence of characters such as name, country or sentence
- C++ strings are stored in arrays of type char
- Standard library in C++, string.h and ctype.h

## String Constants

- A string constant is a sequence of character enclosed by a pair of double quotes (“)
- “C++ Programming”, “Hello World”
- A string end with the NULL character, ‘\o’ to tells the string function where the string ends

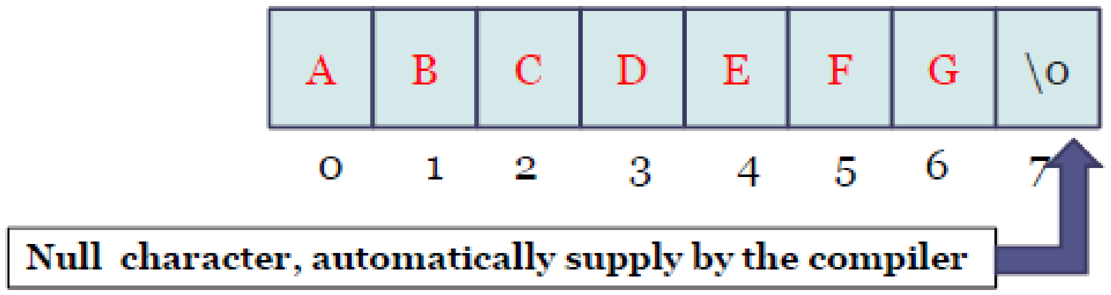
## String variables

- A string variable is C++ is really an array of type char
- The maximum length of the string is limited to the size of the array
  - char name[15];
  - printf(“%s”, name);

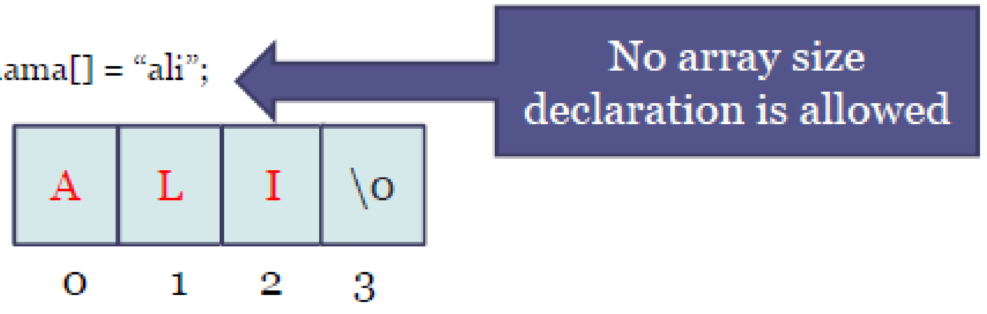
## Initializing Strings

- Strings can be initialized just like array of numbers

char huruf[8] = {"ABCDEFG"};



char nama[] = "ali";



## Built-in String Functions

- C++ has several string processing functions
- These are contained in the header file string.h
  - Strcpy(), strcat(), strcmp(), strrev(), strlen()
  - Example
    - if (strcmp(s1, s2) == 0) printf"sama and serupa");

| Returned value | Meaning |
|---|---|
| Less than zero | s1 less than s2 |
| Zero | s1 identical to s2 |
| Greater than zero | s1 greater than s2 |

## The NULL Terminator

- We can use the NULL terminator at the end of each string
  - strcpy(str, "change case");
  - for (i=0; str[i]; i++)
      str[i] = toupper(str[i]);

Loop will execute until it encounters the NULL terminator which has the value 0

## Pointers & Strings

- An array name is really a pointer in disguise
- When an array name is used without an index, it is a pointer to the *first element* of that array

```
char huruf[8] = "ABCDEFG";
```

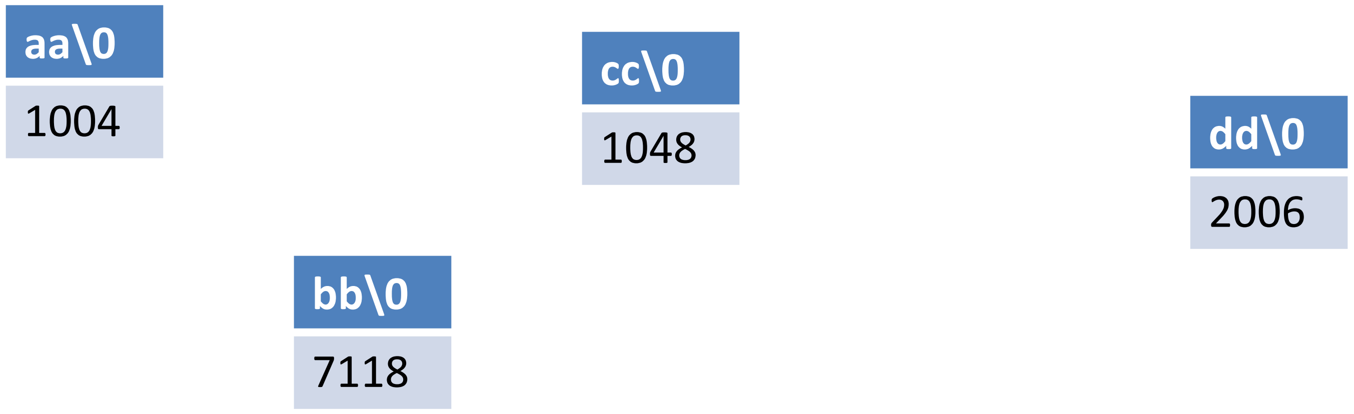
- If you print *huruf, you will see A

## Array of pointers to string

```
char *name[] = {
                "aa",
                "bb",
                "cc"
                };
```

In the above decleration name[] is an array of pointers.

So address of “aa” is stored in name[0], address “bb” is stored in name[1]



name[]

| 1004 | 7118 | 1048 | 2006 |
|---|---|---|---|
| 8112 | 8114 | 8116 | 8118 |

```
/*Exchange names using 2-D array of characters*/
void main()
{     char *name[] = {
                          “aa”,
                          “bb”,
                          “cc”,
                          “dd”
                         }
      char  *temp;
      printf(“Original %s %s”, name[2],name[3]);

      temp=name[2];
      name[2]=name[3];
      name[3]=name[2];

      printf(“New %s %s”, name[2],name[3]);
}
```

## • Limitation of array of pointers to string

```
void main()
{       char *name[2];
        int i;
        for (i=0; i<2; i++)
        {           printf("\nEnter name ");
                    scanf("%s", name[i]);
        }
}
```

The above program will not work, as when we created array it contains garbage value, so it is not valid to send the garbage values to scanf

So solution to this is to dynamically allocate the memory

```
#include <alloc.h>
void main()
{       char *name[2];          char n[50];   int len, I;       char *p;
        int i;
        for (i=0; i<2; i++)
        {           printf("\nEnter name ");
                    scanf("%s", n);
                    len=strlen(n);
                    p=malloc(len+1); //+1 for null character
                    strcpy(p,n);
                    name[i]=p;
        }
}
```

# malloc and free

malloc

Allocated a block of memory and returns a pointer to the allocated block of memory

usage: p = malloc(size);

where p is a pointer

size  is the size of the block to be allocate

free

Free the block of memory allocated using malloc

usage: free(p);

free the block pointed by p

SAC MCA 09