INFORMATIKA
UMN

UMN
UNIVERSITAS
MULTIMEDIA
NUSANTARA

IF232

ALGORITHMS & DATA STRUCTURES

01
ARRAYS & POINTERS

DENNIS GUNAWAN

OUTLINE

Arrays

Pointers

Strings

ARRAYS

- Array: a group of memory locations → **same name & same type**
- To refer to a particular location or element in the array, we specify the **name** of the array and the **position number** of the particular element in the array
- The first element in every array is the **zeroth (0th) element**
- The position number contained within square brackets is more formally called a **subscript** or **index** → must be an integer or integer expression

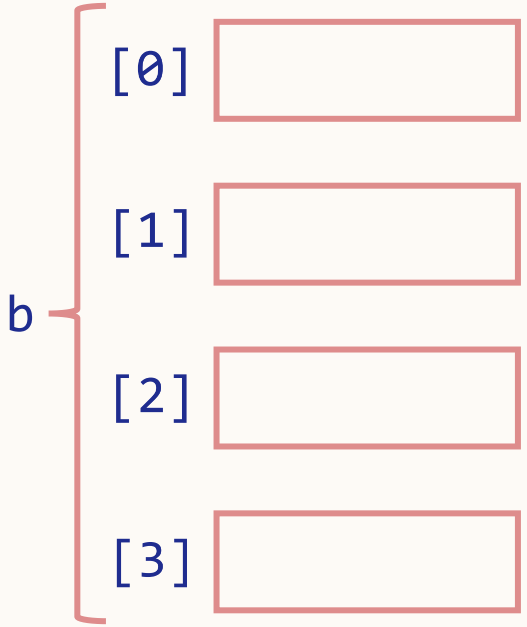
ARRAY DECLARATION

- Syntax

```
element_data_type array_name[size];
```

- Example

```
int a[100];
int b[4];
```



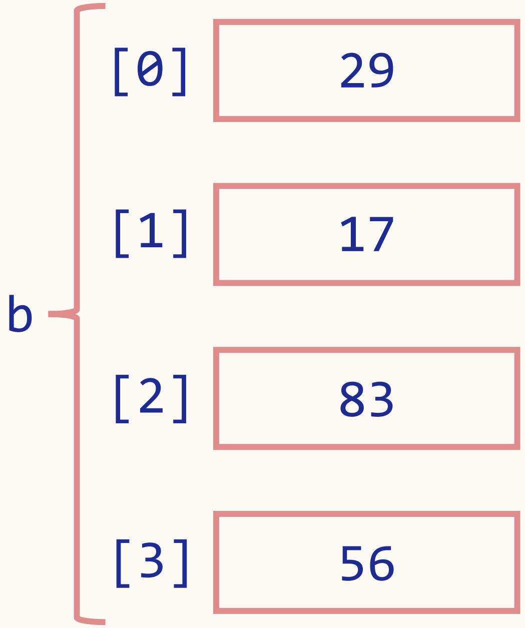
ARRAY INITIALIZATION

- Initialization using for statements

```
int a[100], i;

for(i = 0;i < 100;i++)
{
    a[i] = 0;
}
```

- Initialization using an initializer list

```
int b[4] = {29,17,83,56};
```



ARRAY INITIALIZATION

- If there are fewer initializers than elements in the array, the remaining elements are initialized to zero

```
int a[100] = {0};
```

  - This explicitly initializes the first element to zero and initializes the remaining 99 elements to zero

- If the array size is omitted from a definition with an initializer list, the number of elements in the array will be the number of elements in the initializer list

```
int b[] = {29,17,83,56};
```

  - This would create a four-element array

ARRAY ACCESS

- Syntax

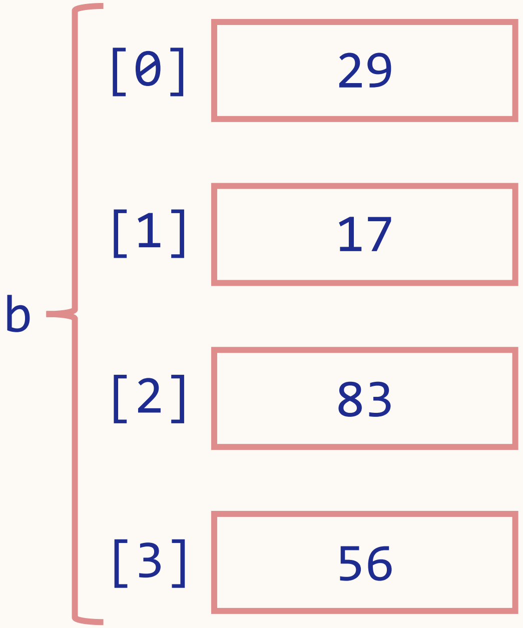
```
array_name[index]
```

- Example
  - How to print 83 ?

```
printf(“%d”,b[2]);
```

  - How to print 29 ?

```
printf(“%d”,b[0]);
```



b

[0] 29

[1] 17

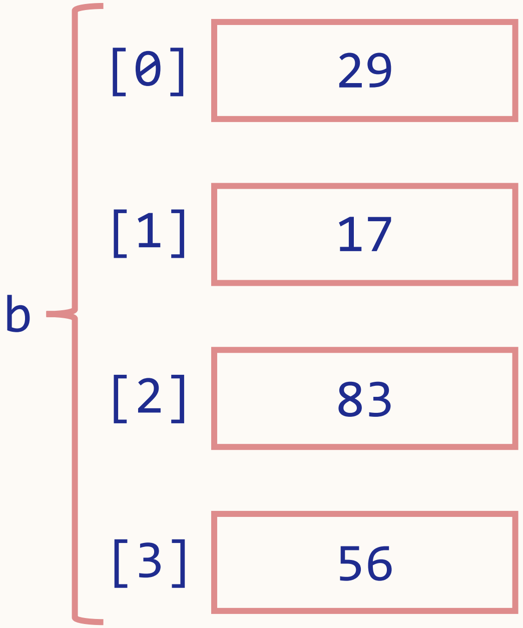
[2] 83

[3] 56

ARRAY ACCESS

- Using for loops for sequential access

```
int i;

for(i = 0;i < 4;i++)
{
   printf("%d\n",b[i]);
}
```



b

[0] 29

[1] 17

[2] 83

[3] 56

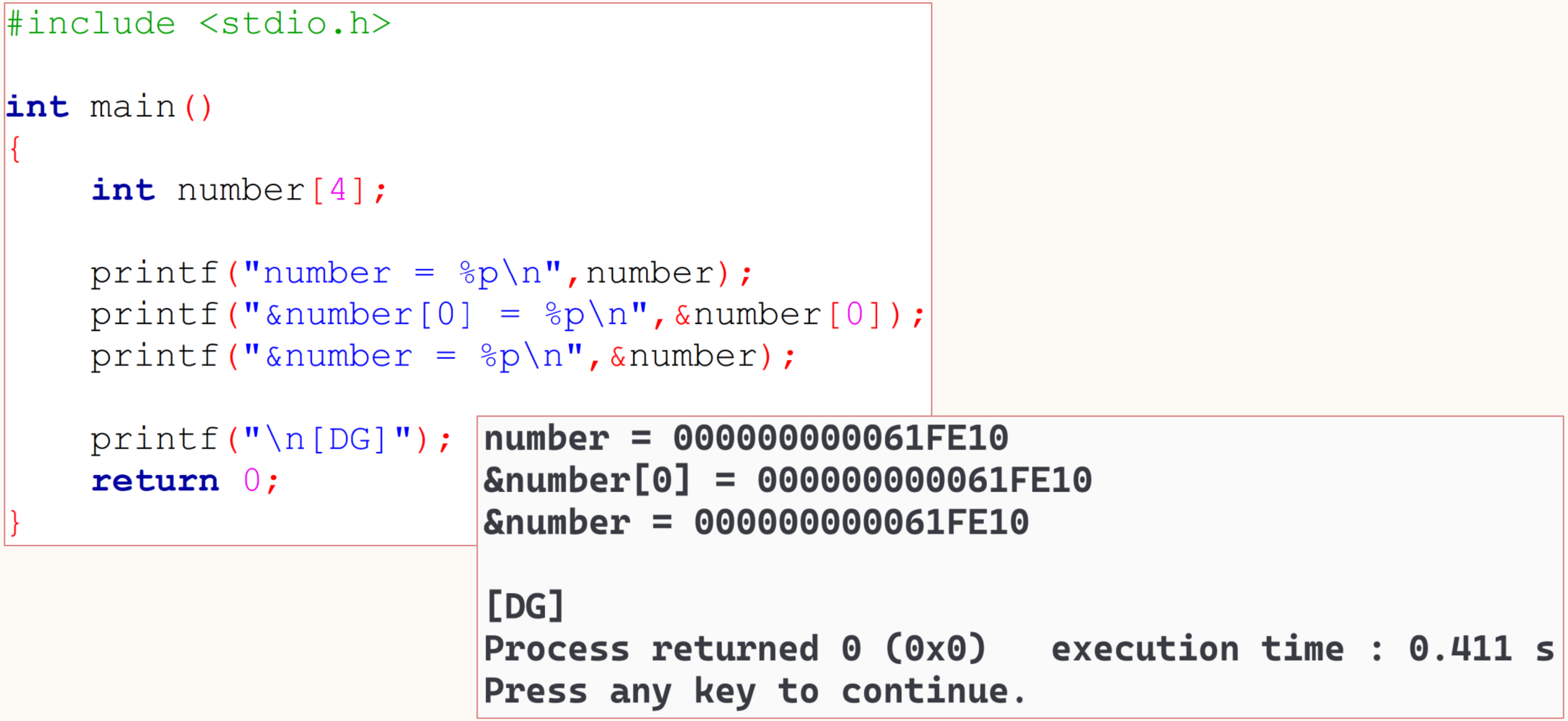
PASSING ARRAYS TO FUNCTIONS

- C automatically passes arrays to functions by reference
  - The called functions can modify the element values in the callers' original arrays

- The name of the array evaluates to the address of the first element of the array
  - **Array name is the same as the address of the array's first element**

```
array_name = &array_name[0]
```

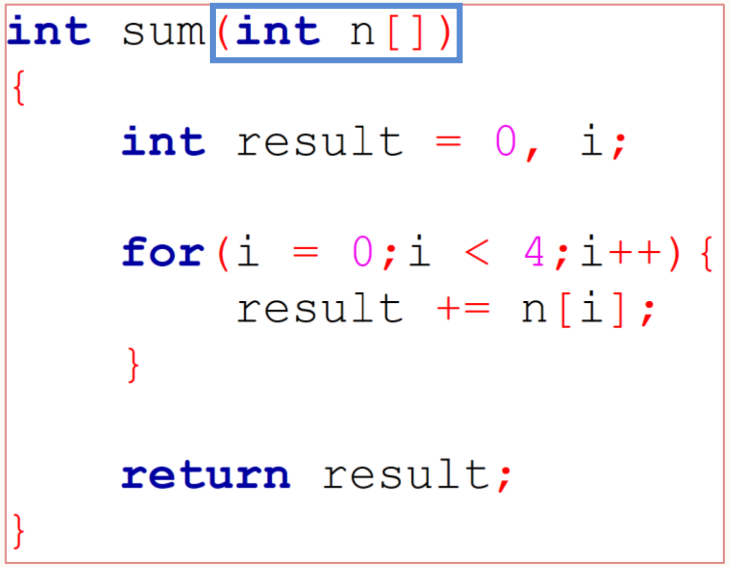
PASSING ARRAYS TO FUNCTIONS



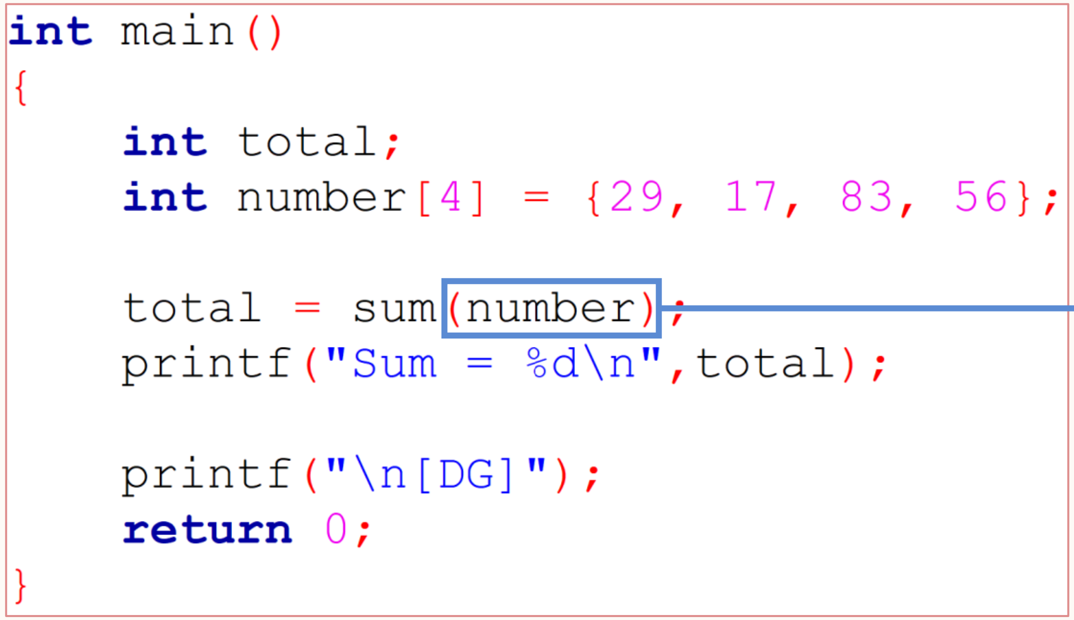
```c
#include <stdio.h>

int main()
{
    int number[4];

    printf("number = %p\n",number);
    printf("&number[0] = %p\n",&number[0]);
    printf("&number = %p\n",&number);

    printf("\n[DG]");
    return 0;
}
```

```
number = 000000000061FE10
&number[0] = 000000000061FE10
&number = 000000000061FE10

[DG]
Process returned 0 (0x0)   execution time : 0.411 s
Press any key to continue.
```

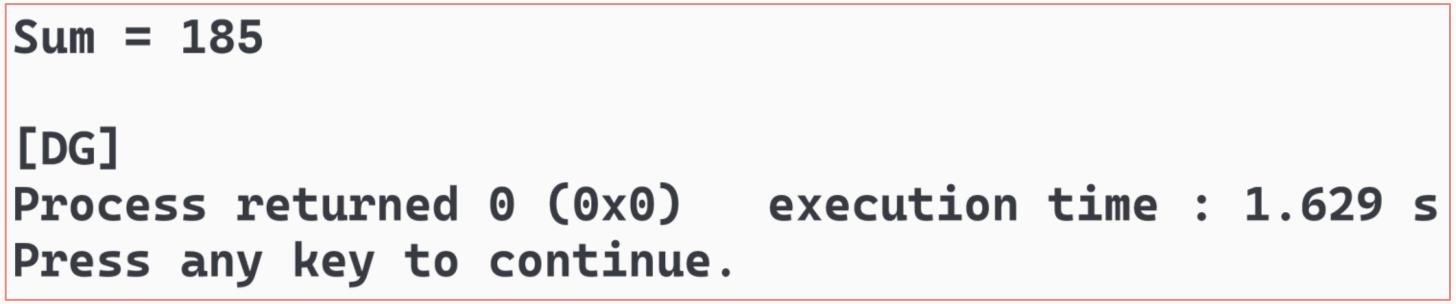
PASSING ARRAYS TO FUNCTIONS



```
int sum(int n[])
{
    int result = 0, i;

    for(i = 0;i < 4;i++){
        result += n[i];
    }

    return result;
}
```



```
int main()
{
    int total;
    int number[4] = {29, 17, 83, 56};

    total = sum(number);
    printf("Sum = %d\n",total);

    printf("\n[DG]");
    return 0;
}
```



```
Sum = 185

[DG]
Process returned 0 (0x0)   execution time : 1.629 s
Press any key to continue.
```

?

```
total = sum(&number[0]);
```

TWO-DIMENSIONAL ARRAYS

- Two-dimensional array is used to represent tables of data (tables of values consisting of information arranged in rows and columns), matrices, and other two-dimensional objects

	Column 0	Column 1	Column 2	Column 3
Row 0	arr[0][0]	arr[0][1]	arr[0][2]	arr[0][3]
Row 1	arr[1][0]	arr[1][1]	arr[1][2]	arr[1][3]
Row 2	arr[2][0]	arr[2][1]	arr[2][2]	arr[2][3]

MULTIDIMENSIONAL ARRAYS

- Declaration

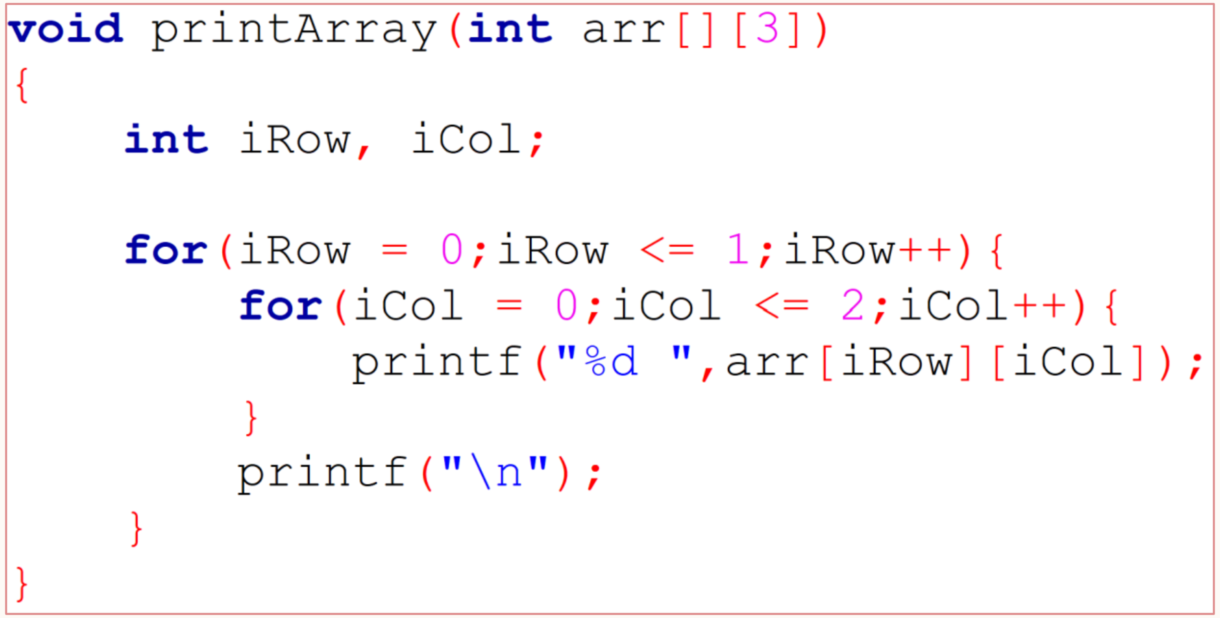
element_data_type array_name[size_1][size_2]...[size_n];

- As a function parameter

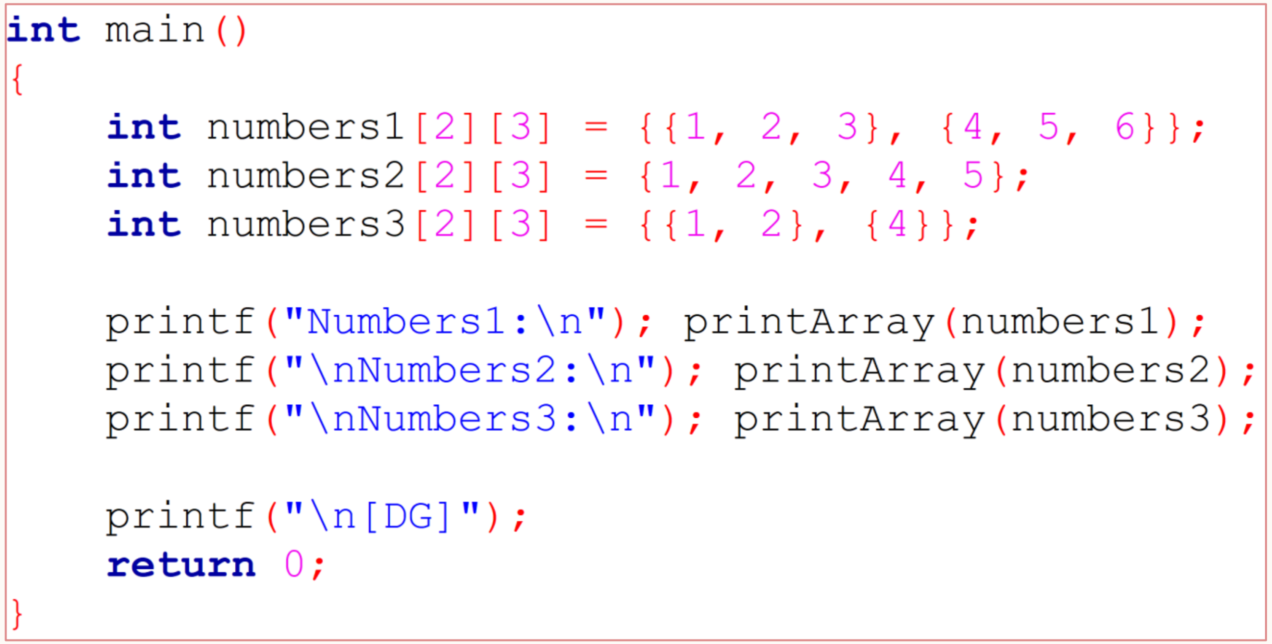
element_data_type array_name[size_1][size_2]...[size_n];

element_data_type array_name[][size_2]...[size_n];

MULTIDIMENSIONAL ARRAYS

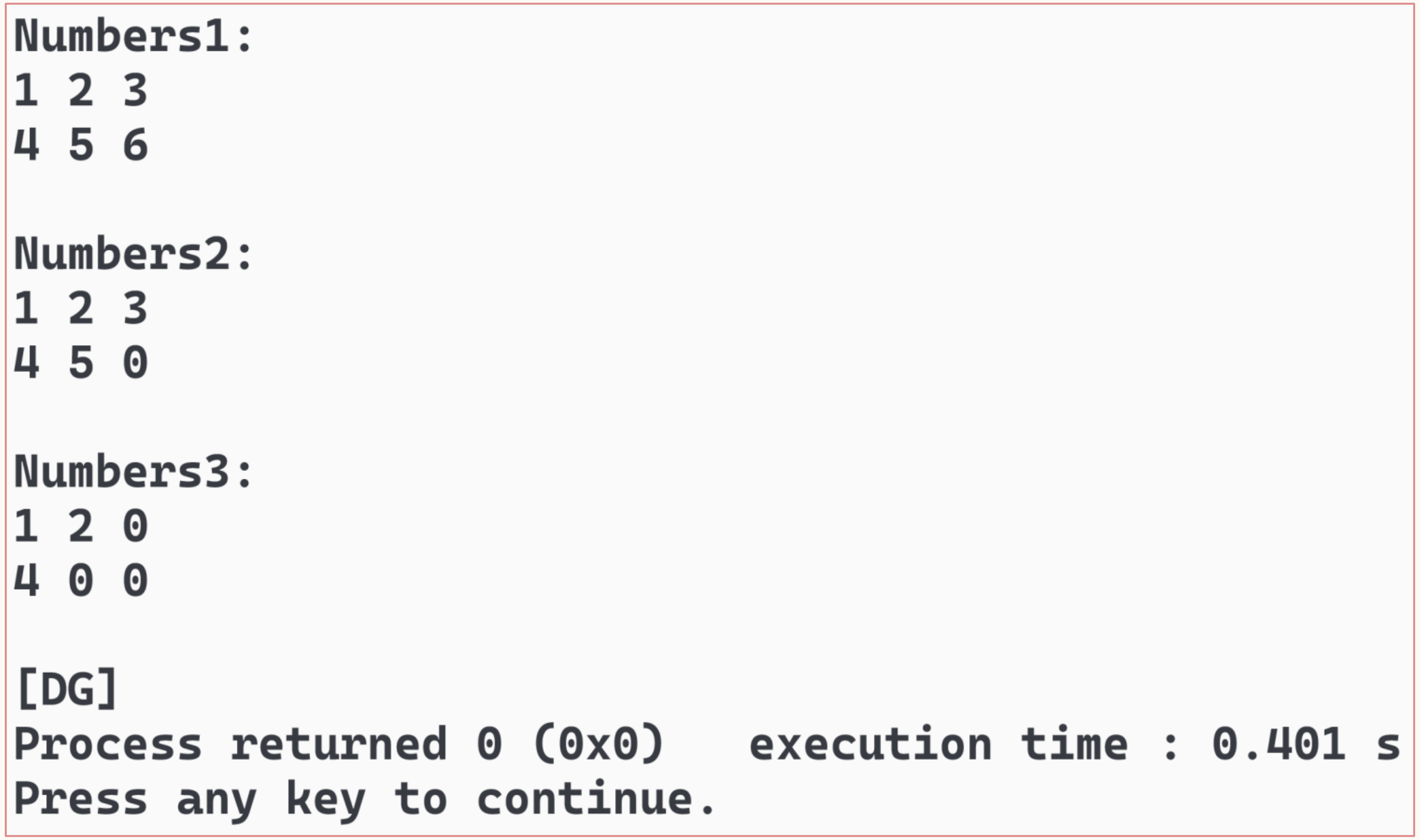


```c
void printArray(int arr[][3])
{
    int iRow, iCol;

    for(iRow = 0;iRow <= 1;iRow++){
        for(iCol = 0;iCol <= 2;iCol++){
            printf("%d ",arr[iRow][iCol]);
        }
        printf("\n");
    }
}
```



```c
int main()
{
    int numbers1[2][3] = {{1, 2, 3}, {4, 5, 6}};
    int numbers2[2][3] = {1, 2, 3, 4, 5};
    int numbers3[2][3] = {{1, 2}, {4}};

    printf("Numbers1:\n"); printArray(numbers1);
    printf("\nNumbers2:\n"); printArray(numbers2);
    printf("\nNumbers3:\n"); printArray(numbers3);

    printf("\n[DG]");
    return 0;
}
```

MULTIDIMENSIONAL ARRAYS



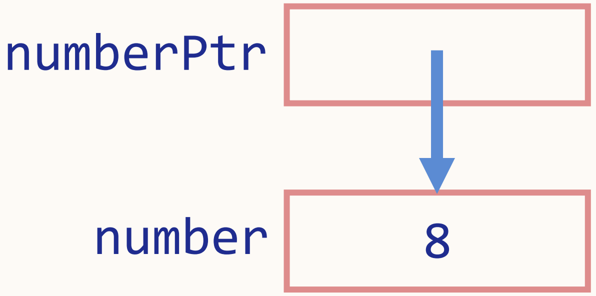
```
Numbers1:
1 2 3
4 5 6

Numbers2:
1 2 3
4 5 0

Numbers3:
1 2 0
4 0 0

[DG]
Process returned 0 (0x0)   execution time : 0.401 s
Press any key to continue.
```

POINTER VARIABLE DEFINITIONS & INITIALIZATION

- Pointers are variables whose values are memory addresses
- A pointer contains an address of a variable that contains a specific value
- A variable name directly references a value, but a pointer indirectly references a value

Pointer **numberPtr** indirectly references the value 8

Variable **number** directly contains the value 8



POINTER VARIABLE DEFINITIONS & INITIALIZATION

- Pointers must be defined before they can be used

```
data_type *pointer_name;
```

- Example

```
int *iPtr;
```

  - What is the data type of iPtr ?

```
int *iPtr;
```

  - What is the data type of *iPtr ?

```
int *iPtr;
```

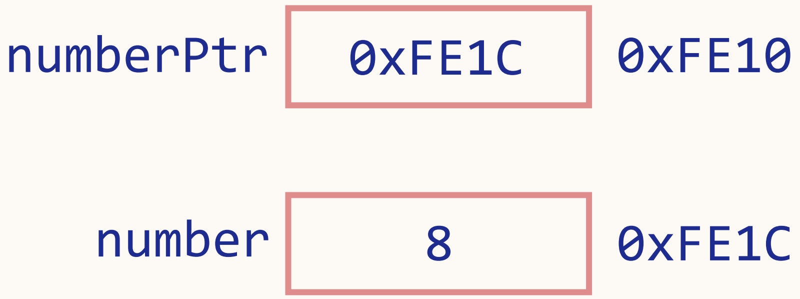
POINTER VARIABLE DEFINITIONS & INITIALIZATION

- Pointers should be initialized either when they are defined or in an assignment statement
- A pointer may be initialized to NULL or an address
- A pointer with the value NULL points to nothing

POINTER OPERATORS

- The address operator (&) returns the address of its operand (variable)
- The indirection operator / dereferencing operator (*) returns the value of the object to which its operand (pointer) points

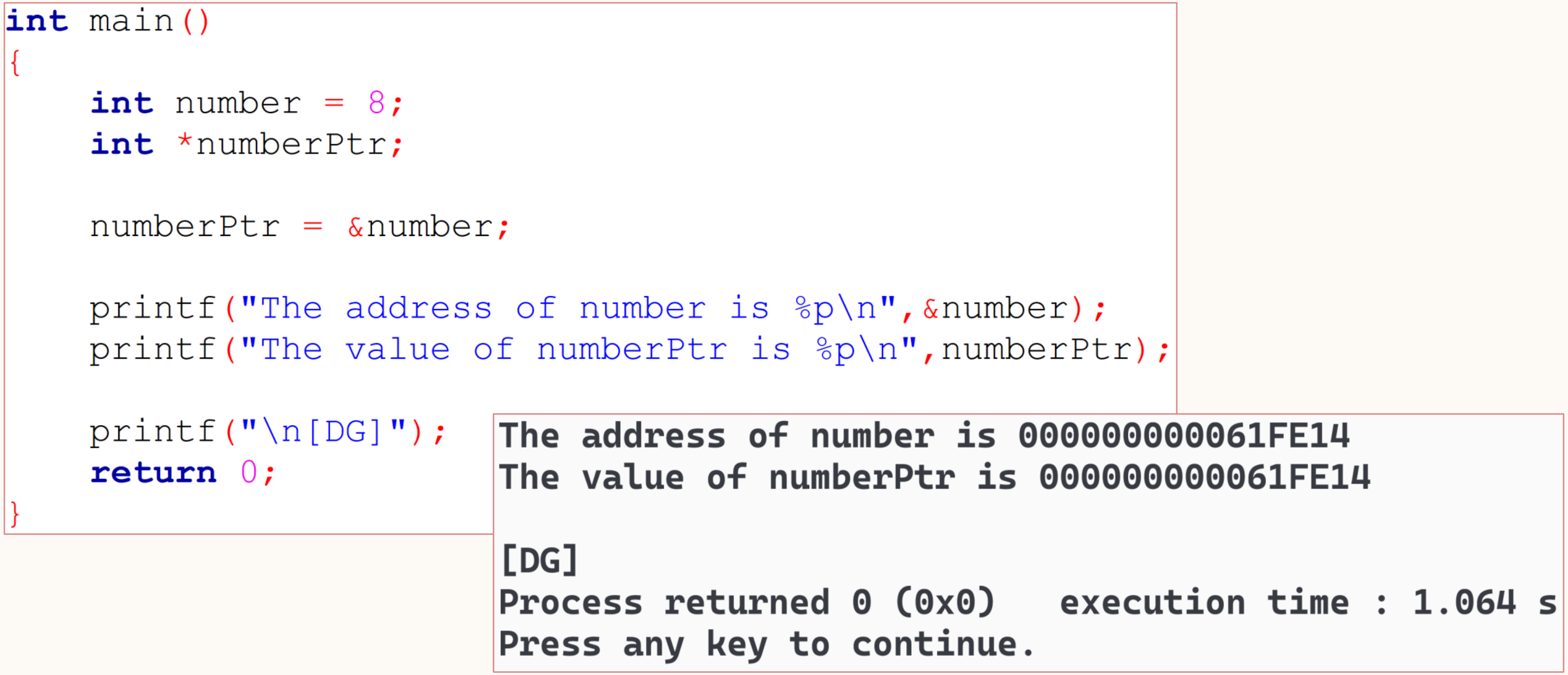
```
int number = 8;
int *numberPtr;

numberPtr = &number;

printf(“%d”,*numberPtr);
```



numberPtr 0xFE1C 0xFE10

number 8 0xFE1C

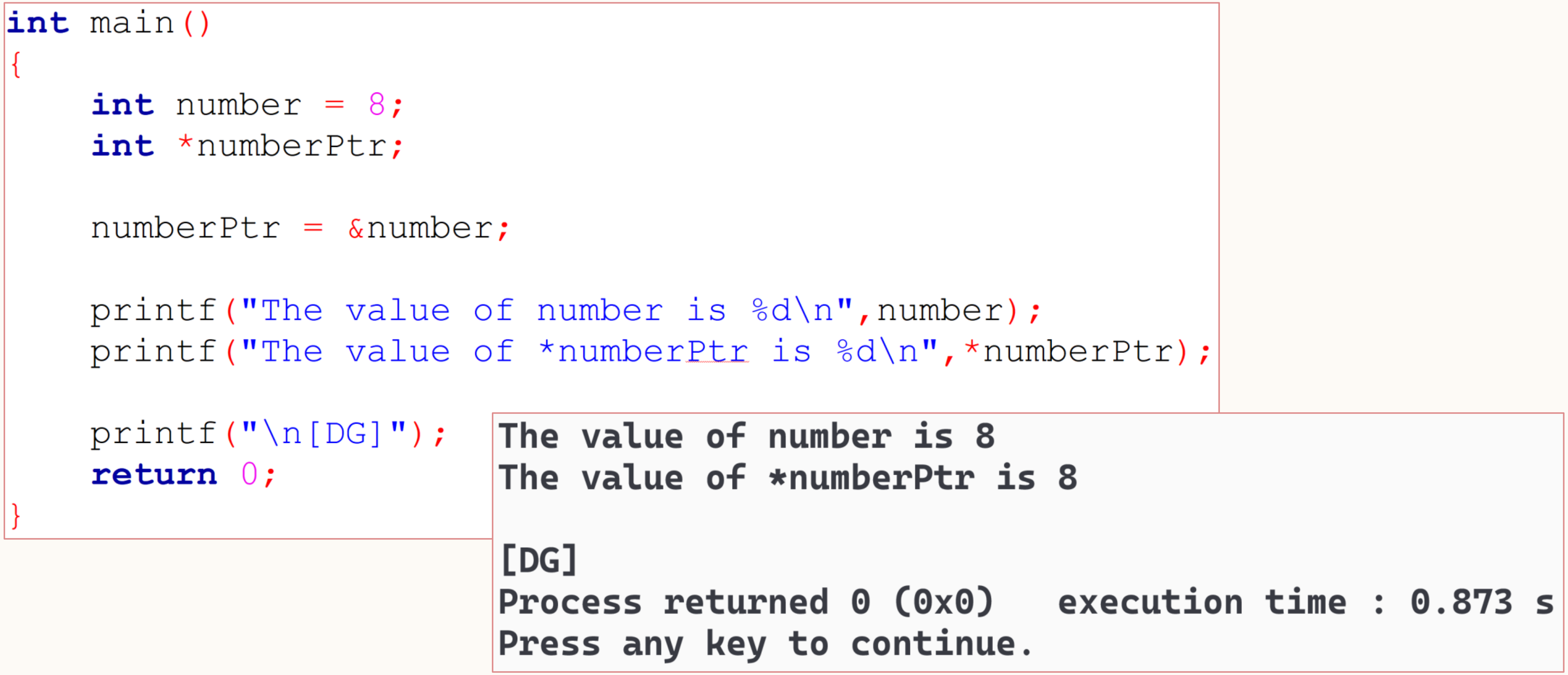
POINTER OPERATORS



```c
int main()
{
    int number = 8;
    int *numberPtr;

    numberPtr = &number;

    printf("The address of number is %p\n",&number);
    printf("The value of numberPtr is %p\n",numberPtr);

    printf("\n[DG]");
    return 0;
}
```

```
The address of number is 000000000061FE14
The value of numberPtr is 000000000061FE14

[DG]
Process returned 0 (0x0)   execution time : 1.064 s
Press any key to continue.
```

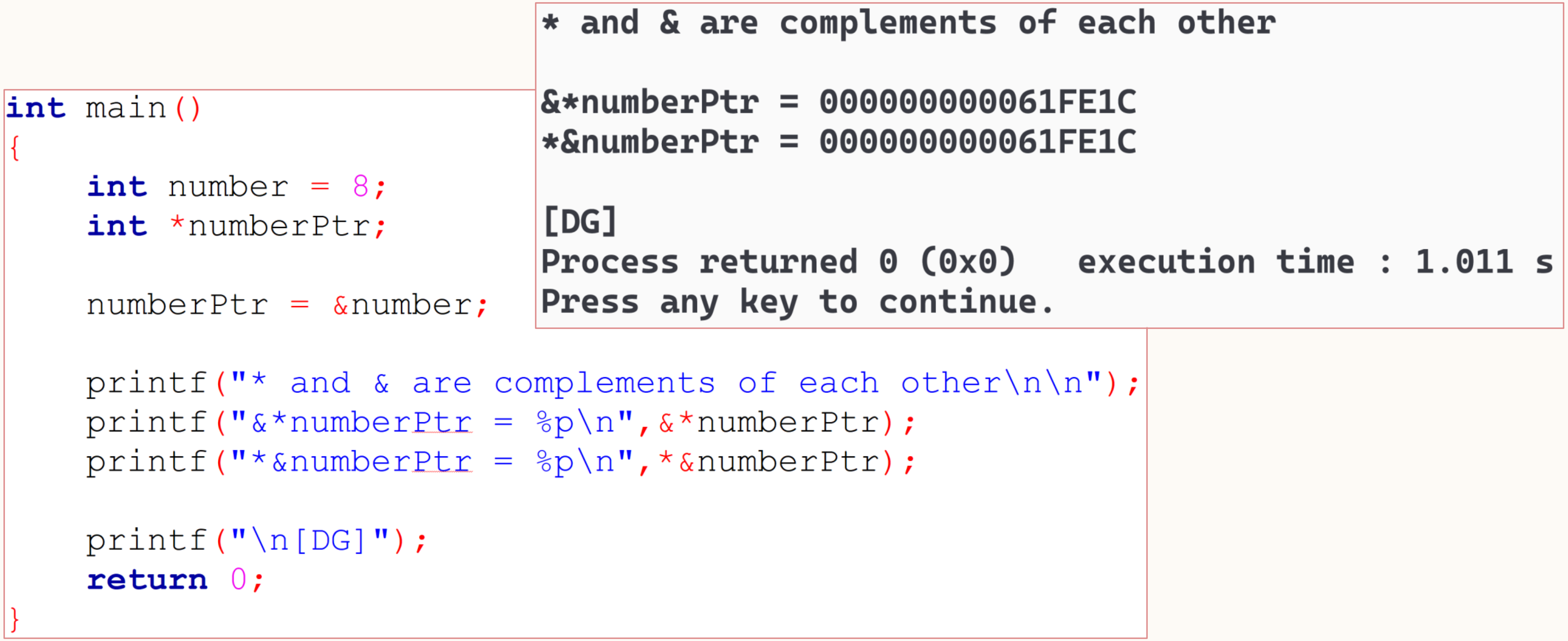
POINTER OPERATORS



```
int main()
{
    int number = 8;
    int *numberPtr;

    numberPtr = &number;

    printf("The value of number is %d\n",number);
    printf("The value of *numberPtr is %d\n",*numberPtr);

    printf("\n[DG]");
    return 0;
}
```

```
The value of number is 8
The value of *numberPtr is 8

[DG]
Process returned 0 (0x0)   execution time : 0.873 s
Press any key to continue.
```

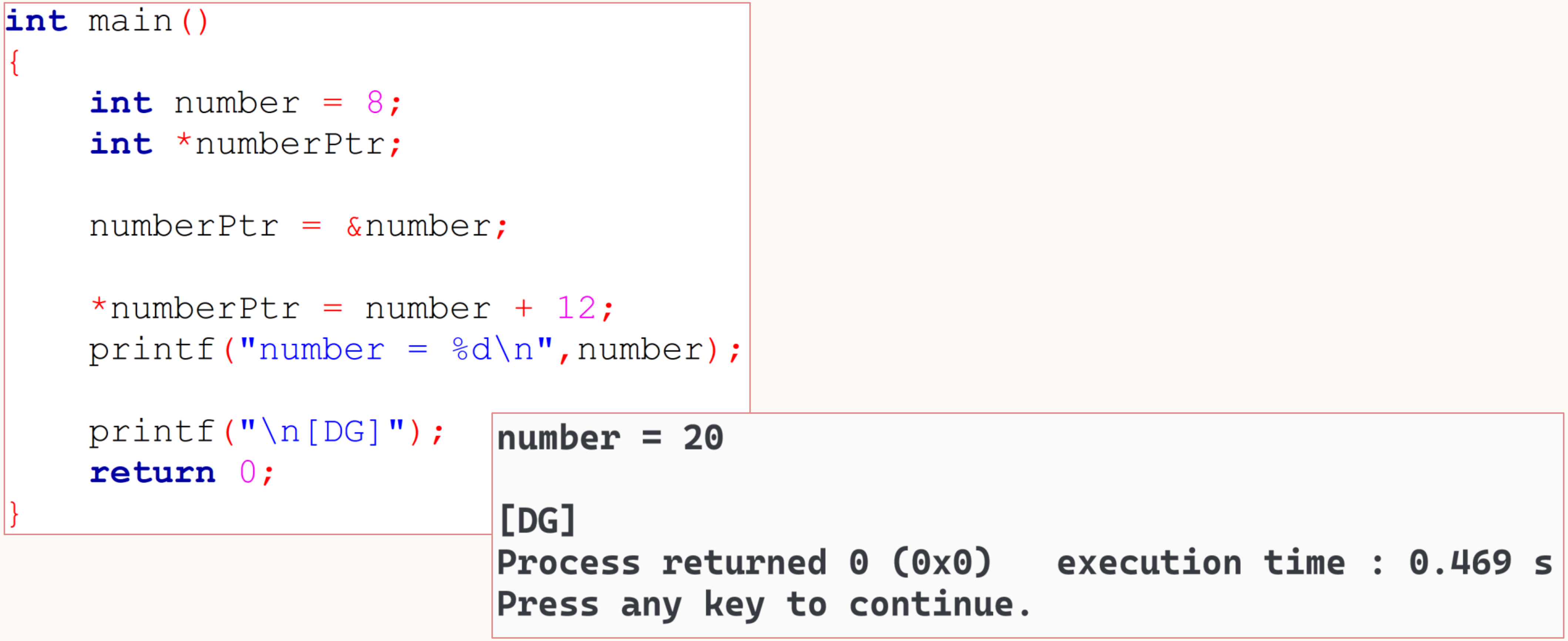
POINTER OPERATORS



```
int main()
{
    int number = 8;
    int *numberPtr;

    numberPtr = &number;

    printf("* and & are complements of each other\n\n");
    printf("&*numberPtr = %p\n",&*numberPtr);
    printf("*&numberPtr = %p\n",*&numberPtr);

    printf("\n[DG]");
    return 0;
}
```

```
* and & are complements of each other

&*numberPtr = 000000000061FE1C
*&numberPtr = 000000000061FE1C

[DG]
Process returned 0 (0x0)   execution time : 1.011 s
Press any key to continue.
```

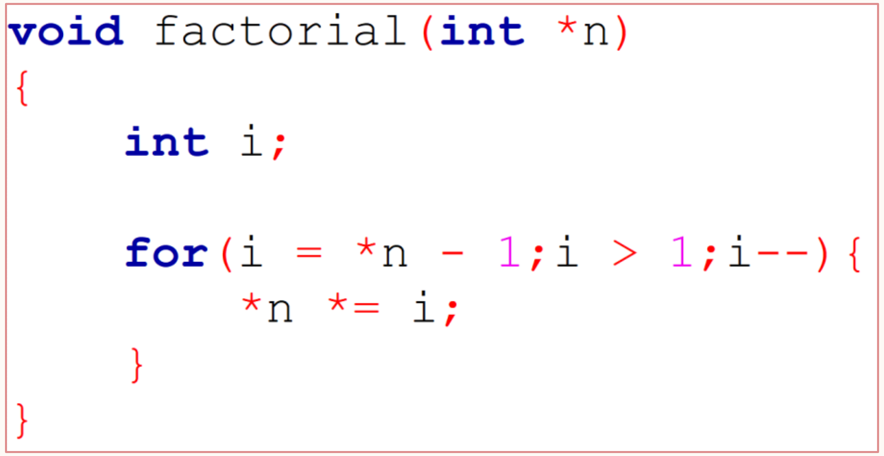
POINTER OPERATORS



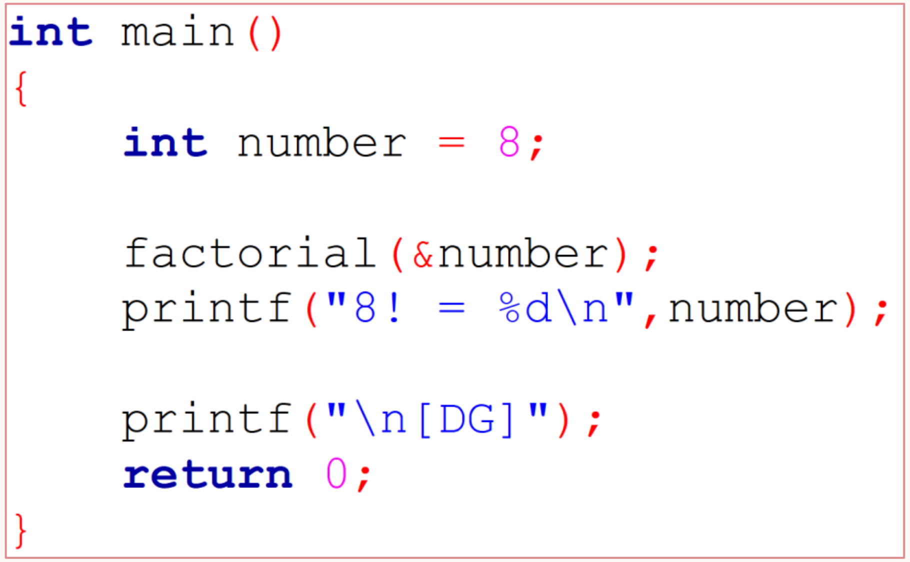
```
int main()
{
    int number = 8;
    int *numberPtr;

    numberPtr = &number;

    *numberPtr = number + 12;
    printf("number = %d\n",number);

    printf("\n[DG]");
    return 0;
}
```

```
number = 20

[DG]
Process returned 0 (0x0)   execution time : 0.469 s
Press any key to continue.
```

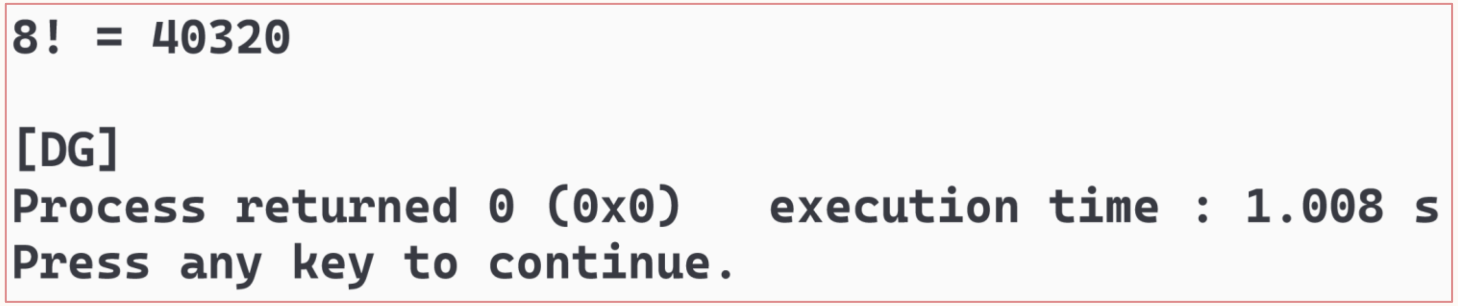
CALL BY POINTER



```c
void factorial(int *n)
{
    int i;

    for(i = *n - 1;i > 1;i--){
        *n *= i;
    }
}
```



```c
int main()
{
    int number = 8;

    factorial(&number);
    printf("8! = %d\n",number);

    printf("\n[DG]");
    return 0;
}
```



```
8! = 40320

[DG]
Process returned 0 (0x0)   execution time : 1.008 s
Press any key to continue.
```

POINTERS & ARRAYS

```
int arr[7];
int *arrPtr;
```

- Since the array name (without a subscript) is a pointer to the first element of the array, we can set **arrPtr** equal to the address of the first element in array **arr**

```
arrPtr = arr;
```

=

```
arrPtr = &arr[0];
```

POINTERS & ARRAYS

```
int arr[7];
int *arrPtr;
```

- The address **&arr[n]** can be written with the pointer expression

```
arrPtr + n
```

- Array element **arr[n]** can alternatively be referenced with the pointer expression

```
*(arrPtr + n)
```

POINTERS & ARRAYS

```
int arr[7];
int *arrPtr;
```

- The array itself can be treated as a pointer

```
*(arr + n)
```

- Pointers can be subscripted exactly as arrays can

```
arrPtr[n]
```

STRINGS

- A string is a series of characters treated as a single unit
- A string may include letters, digits, and various special characters
- String literals (string constants) in C are written in **double quotation marks**
- A string in C is an **array of characters ending in the null character (‘\0’)**
- A string is accessed via a pointer to the first character in the string
- The value of a string is the address of its first character

STRINGS

```
char color[] = “cyan”;
char color[] = {‘c’, ‘y’, ‘a’, ‘n’};
```

- Create a 5-element array color containing the characters ‘c’, ‘y’, ‘a’, ‘n’, and ‘\0’

color[0]	color[1]	color[2]	color[3]	color[4]
c	y	a	n	\0

```
char color[7] = “cyan”;
```

color[0]	color[1]	color[2]	color[3]	color[4]	color[5]	color[6]
c	y	a	n	\0		

STRINGS

```
char *color = “cyan”;
```

- Creates pointer variable color that points to the string “cyan” somewhere in memory

color[0]	color[1]	color[2]	color[3]	color[4]
c	y	a	n	\0

STRING LIBRARY FUNCTIONS

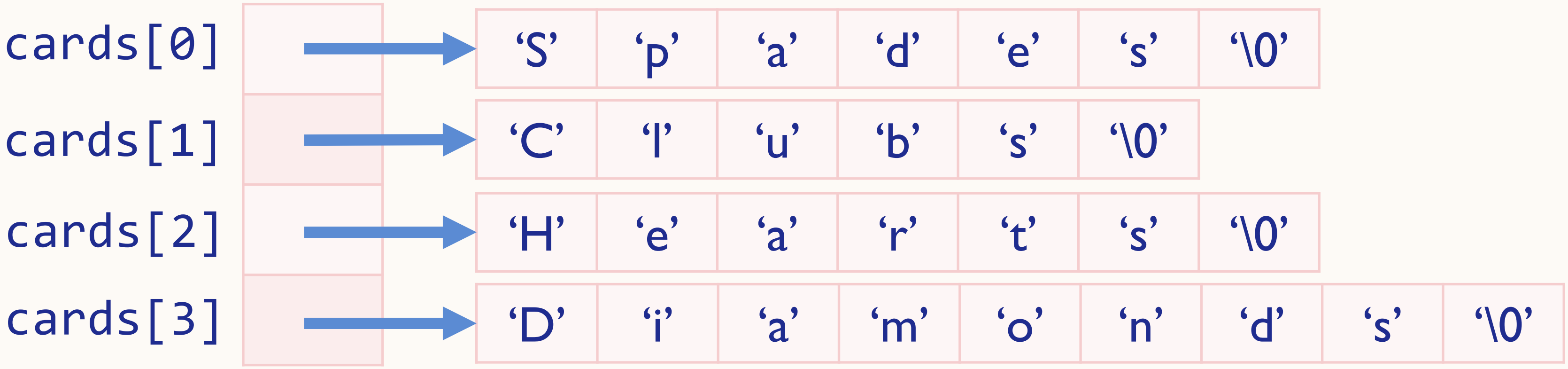
Function Prototype	Function Description
char *strcpy(char *s1, const char *s2);	**Copies string s2 into array s1** The value of s1 is returned
char *strncpy(char *s1, const char *s2, size_t n);	**Copies at most n characters of string s2 into array s1** The value of s1 is returned
char *strcat(char *s1, const char *s2);	**Appends string s2 to array s1** The first character of s2 overwrites the terminating null character of s1 The value of s1 is returned
size_t strlen(const char *s);	**Determines the length of string s** The number of characters preceding the terminating null character is returned

STRING LIBRARY FUNCTIONS

Function Prototype	Function Description
int strcmp(const char *s1, const char *s2);	**Compares the string s1 with the string s2** The function returns 0, less than 0, or greater than 0 if s1 is equal to, less than, or greater than s2, respectively
int strncmp(const char *s1, const char *s2, size_t n);	**Compares up to n characters of the string s1 with the string s2** The function returns 0, less than 0, or greater than 0 if s1 is equal to, less than, or greater than s2, respectively

ARRAYS OF STRINGS

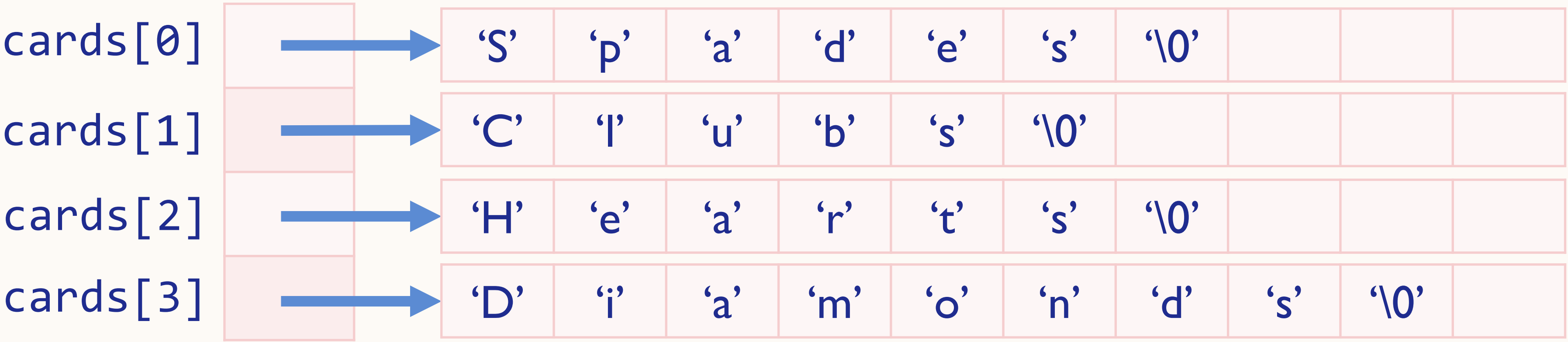
```
char *cards[4] = {“Spades”, “Clubs”, “Hearts”, “Diamonds”};
```



cards[0]	‘S’	‘p’	‘a’	‘d’	‘e’	‘s’	‘\0’		
cards[1]	‘C’	‘l’	‘u’	‘b’	‘s’	‘\0’			
cards[2]	‘H’	‘e’	‘a’	‘r’	‘t’	‘s’	‘\0’		
cards[3]	‘D’	‘i’	‘a’	‘m’	‘o’	‘n’	‘d’	‘s’	‘\0’

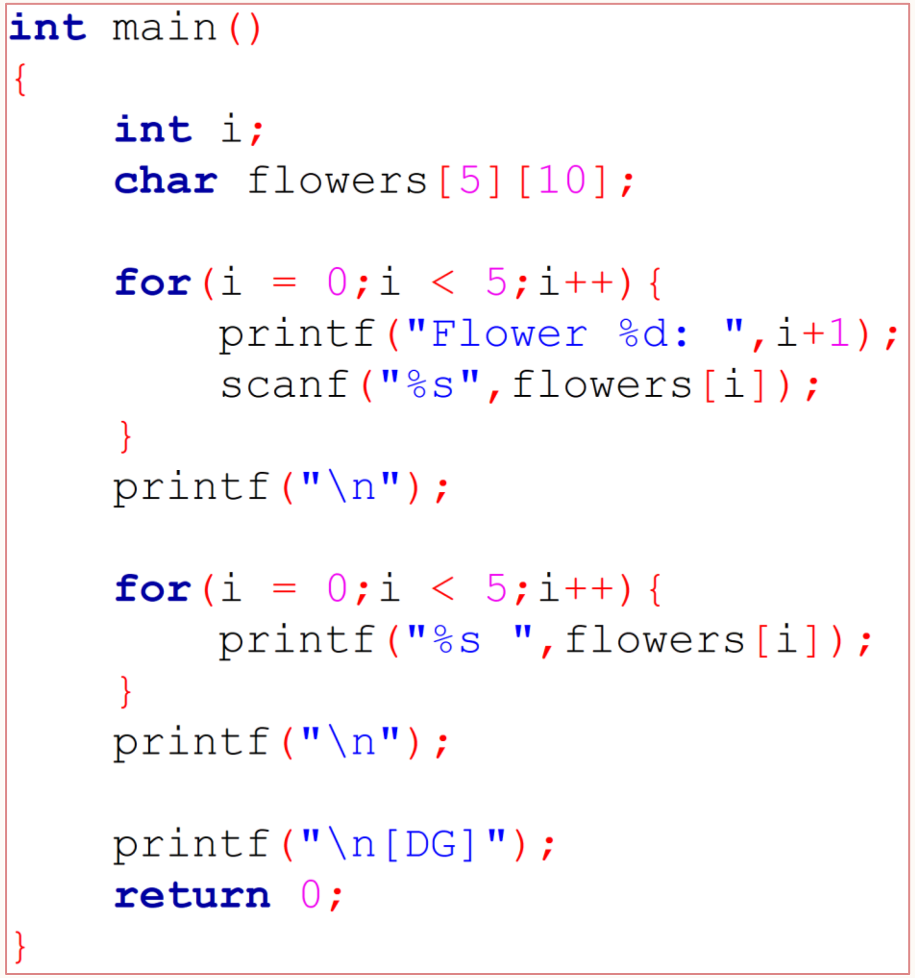
ARRAYS OF STRINGS

```
char cards[4][10] = {“Spades”, “Clubs”, “Hearts”, “Diamonds”};
```

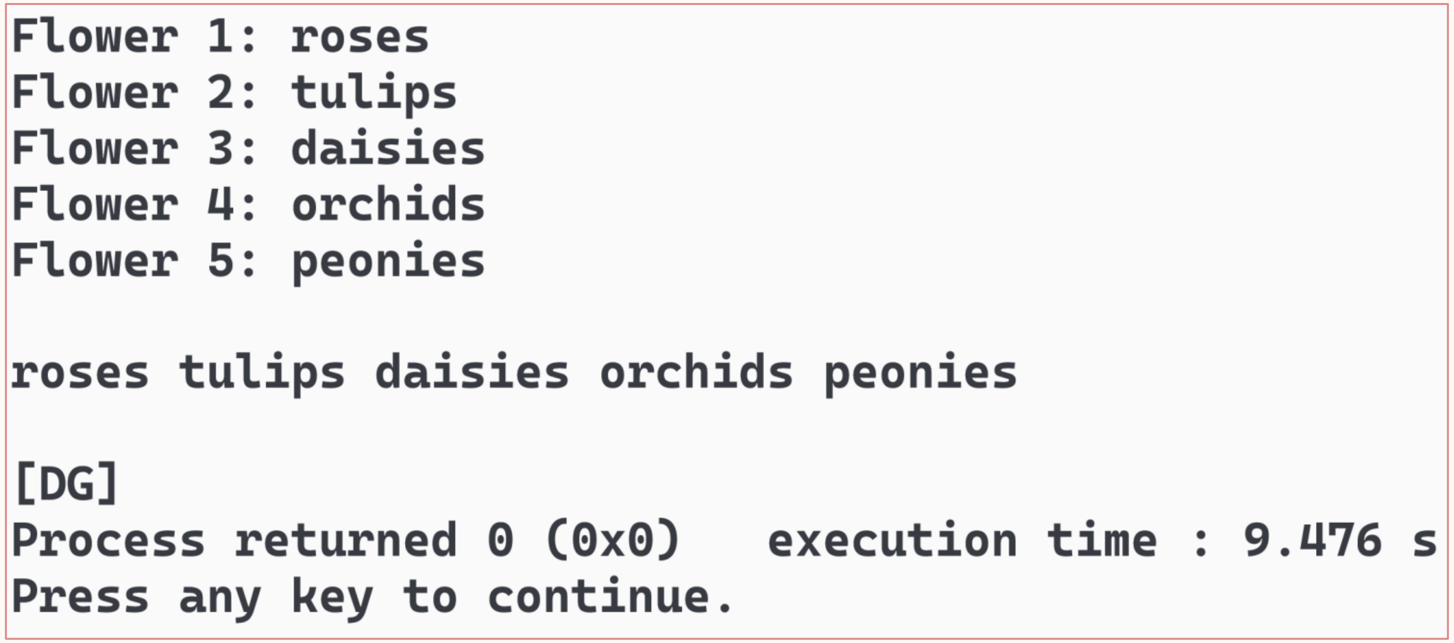


cards[0]	‘S’	‘p’	‘a’	‘d’	‘e’	‘s’	‘\0’			
cards[1]	‘C’	‘l’	‘u’	‘b’	‘s’	‘\0’				
cards[2]	‘H’	‘e’	‘a’	‘r’	‘t’	‘s’	‘\0’			
cards[3]	‘D’	‘i’	‘a’	‘m’	‘o’	‘n’	‘d’	‘s’	‘\0’	

ARRAYS OF STRINGS

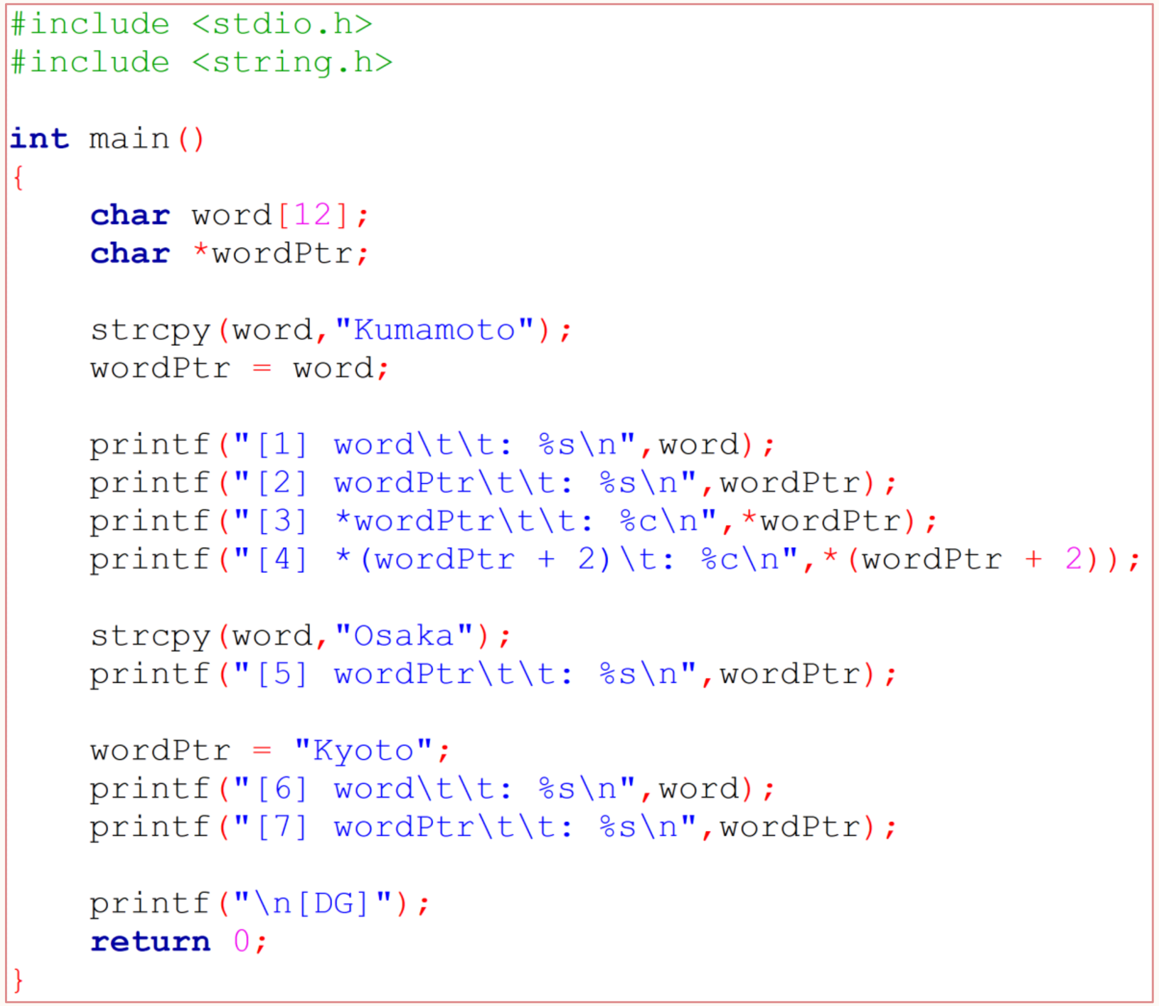


```
int main()
{
    int i;
    char flowers[5][10];

    for(i = 0;i < 5;i++){
        printf("Flower %d: ",i+1);
        scanf("%s",flowers[i]);
    }
    printf("\n");

    for(i = 0;i < 5;i++){
        printf("%s ",flowers[i]);
    }
    printf("\n");

    printf("\n[DG]");
    return 0;
}
```



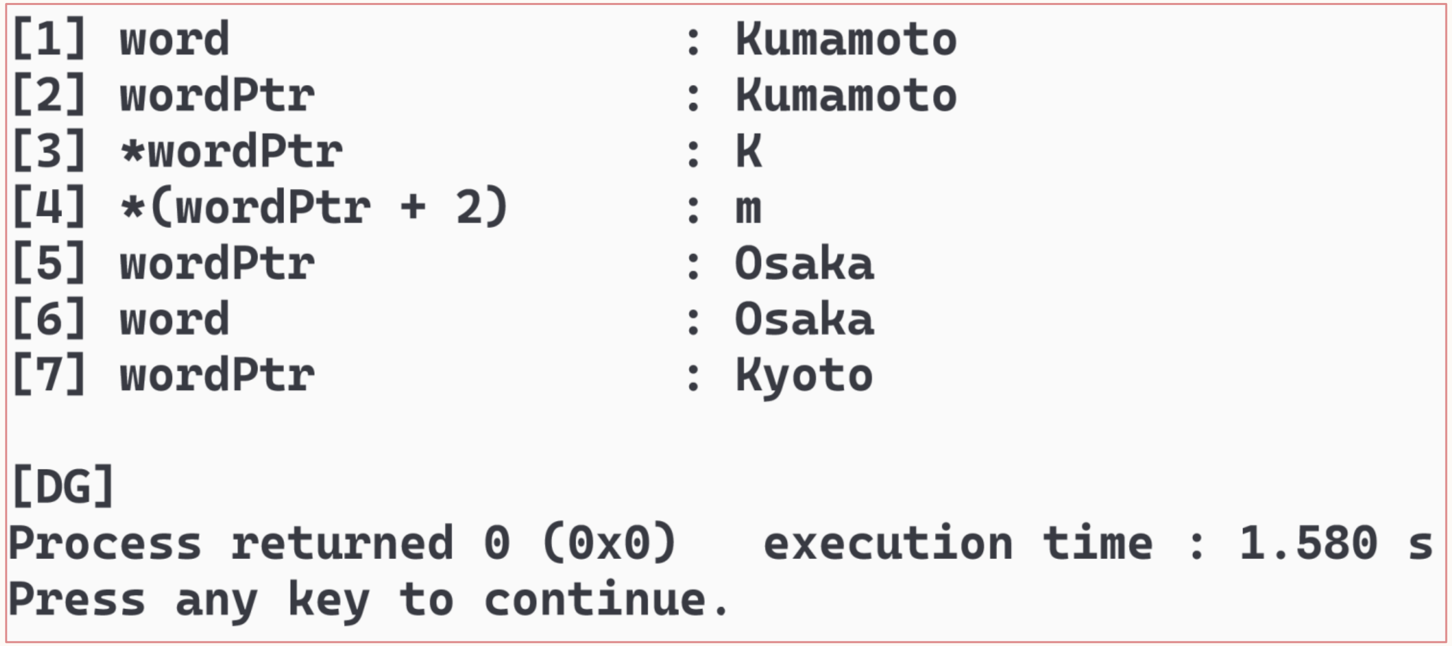
```
Flower 1: roses
Flower 2: tulips
Flower 3: daisies
Flower 4: orchids
Flower 5: peonies

roses tulips daisies orchids peonies

[DG]
Process returned 0 (0x0)   execution time : 9.476 s
Press any key to continue.
```

POINTER TO STRINGS



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

int main()
{
    char word[12];
    char *wordPtr;

    strcpy(word,"Kumamoto");
    wordPtr = word;

    printf("[1] word\t\t: %s\n",word);
    printf("[2] wordPtr\t\t: %s\n",wordPtr);
    printf("[3] *wordPtr\t\t: %c\n",*wordPtr);
    printf("[4] *(wordPtr + 2)\t: %c\n",*(wordPtr + 2));

    strcpy(word,"Osaka");
    printf("[5] wordPtr\t\t: %s\n",wordPtr);

    wordPtr = "Kyoto";
    printf("[6] word\t\t: %s\n",word);
    printf("[7] wordPtr\t\t: %s\n",wordPtr);

    printf("\n[DG]");
    return 0;
}
```

POINTER TO STRINGS



```
[1] word                    : Kumamoto
[2] wordPtr                 : Kumamoto
[3] *wordPtr                : K
[4] *(wordPtr + 2)          : m
[5] wordPtr                 : Osaka
[6] word                    : Osaka
[7] wordPtr                 : Kyoto

[DG]
Process returned 0 (0x0)   execution time : 1.580 s
Press any key to continue.
```

PRACTICE

EXERCISES

1. Find the error in each of the following program segments and correct the error.

   a.

   ```
   int p[5] = {0};
   int i;

   for(i = 0;i <= 5;++i){
       p[i] = 1;
   }
   ```

   b.

   ```
   int n[2][2] = {{8, 7}, {6, 5}};
   n[1, 1] = 3;
   ```

EXERCISES

1. Find the error in each of the following program segments and correct the error.

c.

```
char w[5] = “”;
scanf(“%s”, w); // user types hello
```

d.

```
double d[3] = {1.1, 10.01, 100.001, 1000.0001};
```

e.

```
float f = 87.56;
float fPtr = &f;
printf(“%f\n”, fPtr);
```

EXERCISES

1. Find the error in each of the following program segments and correct the error.

f.

```
int *a, b;
a = b;
```

g.

```
int *iPtr, j;
int i[5] = {1, 2, 3, 4, 5};
iPtr = i;

printf(“%d\n”,iPtr);

for(j = 0;j <= 5;j++)
   printf(“%d\n”,*iPtr[j]);
```

EXERCISES

1. Find the error in each of the following program segments and correct the error.

h.

```
char str[10] = “”;
strncpy(str, “hello”, 5);
printf(“%s\n”, str);
```

i.

```
printf(“%s”, ‘a’);
```

j.

```
char str[12] = “”;
strcpy(str, “Welcome Home”);
```

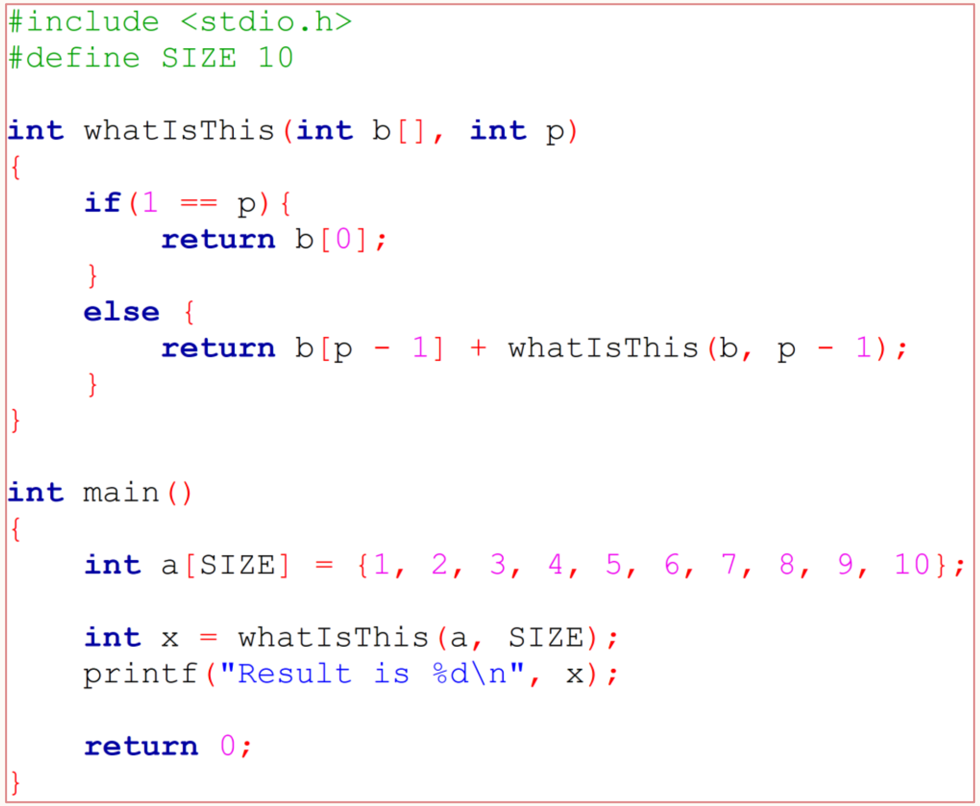
EXERCISES

1. Find the error in each of the following program segments and correct the error.
   k.

```
if(strcmp(string1, string2)){
    puts(“The strings are equal”);
}
```

EXERCISES

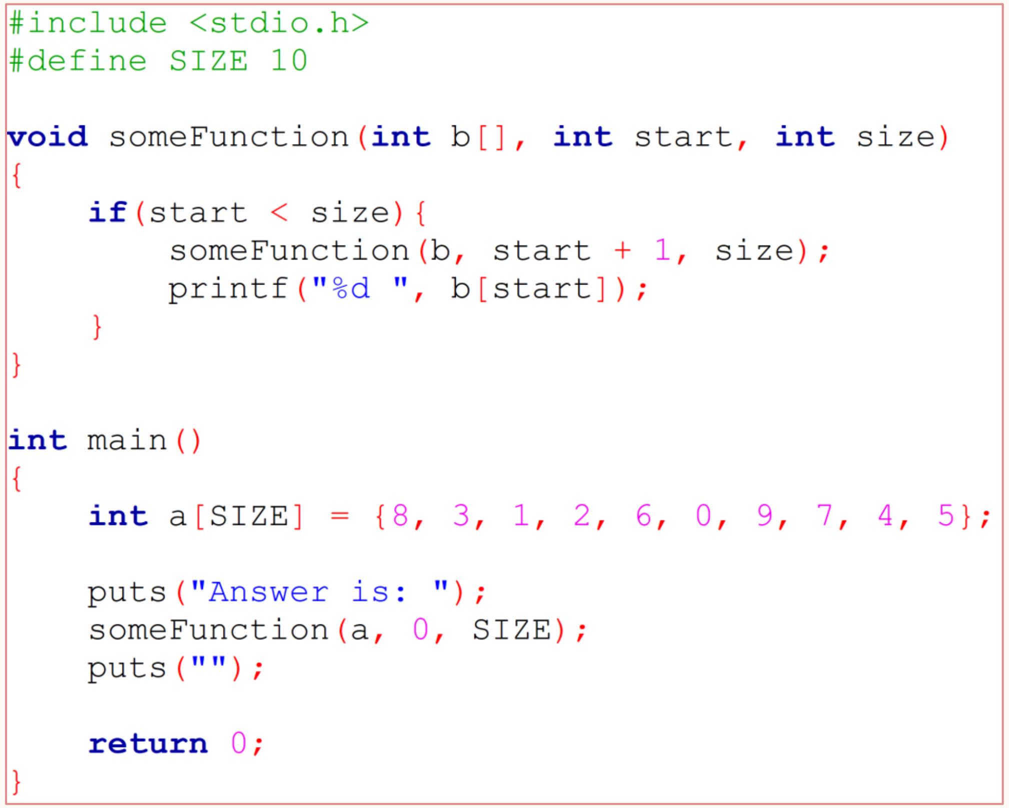
2. What does the following program do?



```
#include <stdio.h>
#define SIZE 10

int whatIsThis(int b[], int p)
{
    if(1 == p){
        return b[0];
    }
    else {
        return b[p - 1] + whatIsThis(b, p - 1);
    }
}

int main()
{
    int a[SIZE] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

    int x = whatIsThis(a, SIZE);
    printf("Result is %d\n", x);

    return 0;
}
```

EXERCISES

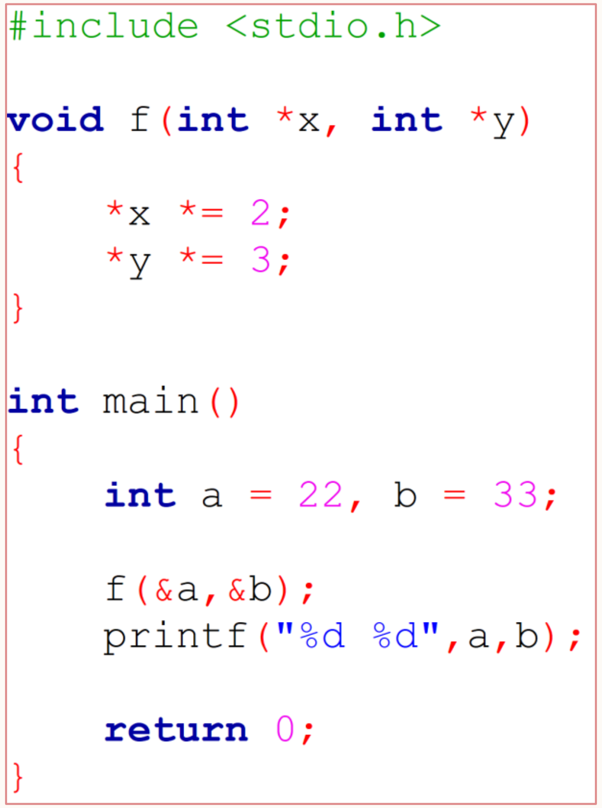
3. What does the following program do?



```c
#include <stdio.h>
#define SIZE 10

void someFunction(int b[], int start, int size)
{
    if(start < size){
        someFunction(b, start + 1, size);
        printf("%d ", b[start]);
    }
}

int main()
{
    int a[SIZE] = {8, 3, 1, 2, 6, 0, 9, 7, 4, 5};

    puts("Answer is: ");
    someFunction(a, 0, SIZE);
    puts("");

    return 0;
}
```

EXERCISES

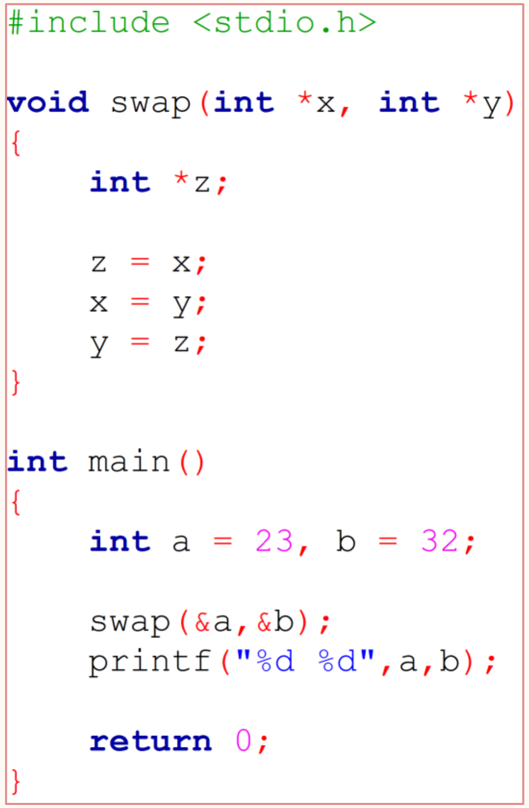
4. What is displayed by the following code?



```
#include <stdio.h>

void f(int *x, int *y)
{
    *x *= 2;
    *y *= 3;
}

int main()
{
    int a = 22, b = 33;

    f(&a,&b);
    printf("%d %d",a,b);

    return 0;
}
```

EXERCISES

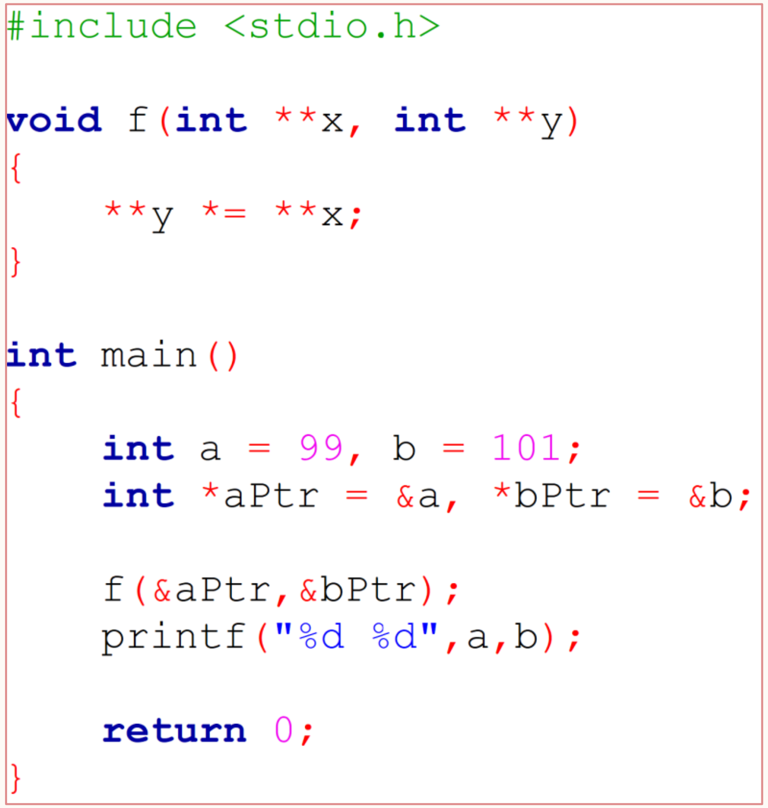
5. What is displayed by the following code?



```c
#include <stdio.h>

void swap(int *x, int *y)
{
    int *z;

    z = x;
    x = y;
    y = z;
}

int main()
{
    int a = 23, b = 32;

    swap(&a,&b);
    printf("%d %d",a,b);

    return 0;
}
```

EXERCISES

6. What is displayed by the following code?



```c
#include <stdio.h>

void f(int **x, int **y)
{
    **y *= **x;
}

int main()
{
    int a = 99, b = 101;
    int *aPtr = &a, *bPtr = &b;

    f(&aPtr,&bPtr);
    printf("%d %d",a,b);

    return 0;
}
```

EXERCISES

7. What, if anything, prints when each of the following C statements is performed? If the statement contains an error, describe the error and indicate how to correct it.

```
char s1[50] = “asterix”;
char s2[50] = “obelix”;
char s3[50] = “”;
```

a. `printf(“%c%s”, toupper(s1[0]), &s1[1]);`
b. `printf(“%s”, strcpy(s3,s2));`
c. `printf(“%s”, strcat(strcat(strcpy(s3,s1),“ and ”),s2));`
d. `printf(“%u”, strlen(s1) + strlen(s2));`

EXERCISES

8. Given these declarations.

```
char ssn[12] = “123-45-6789”;
char ssn1[4], ssn2[3], ssn3[5];
```

Write statements to accomplish the following.

a. Store in ssn1 the first three characters of ssn.
b. Store in ssn2 the middle two-digit portion of ssn.
c. Store in ssn3 the final four digits of ssn.

EXERCISES

9. Given the string name (value is “Adam, Bryan Zachary”) and the 40-character temporary variables tmp1 and tmp2, what string is displayed by the following code fragment?

```
strncpy(tmp1, &name[6], 5);
tmp1[5] = ‘\0’;
strcat(tmp1, “ ”);
strncpy(tmp2, name, 4);
tmp2[4] = ‘\0’;
printf(“%s\n”, strcat(tmp1, tmp2));
```

LAB

EXERCISES

1. Write a program that reads a line of text and prints the number of occurrences of each letter of the alphabet in the text.

```
Followers will never know how hard the leader tries to create path

a    4          n    2
b    0          o    5
c    1          p    1
d    2          q    0
e    9          r    6
f    1          s    2
g    0          t    5
h    4          u    0
i    2          v    1
j    0          w    4
k    1          x    0
l    5          y    0
m    0          z    0
```

EXERCISES

2. Write a program that reads a line of text and prints the number of one-letter words, two-letter words, three-letter words, and so on, appearing in the text.

```
Followers will never know how hard the leader tries to create path

Word Length        Occurrences
1                  0
2                  1
3                  2
4                  4
5                  2
6                  2
7                  0
8                  0
9                  1
```

EXERCISES

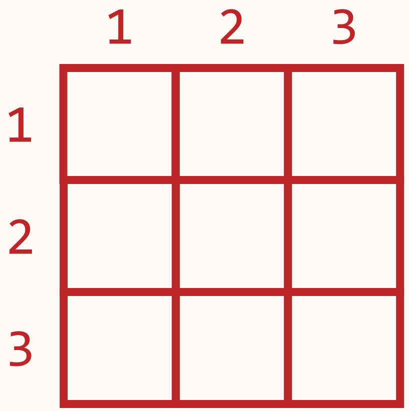
3. Write a program to play a game of tic-tac-toe.

   Input format: [row] [column] [X/O]

```
2 2 X
1 1 O
2 1 X
2 3 O
1 2 X
3 2 O
3 3 X
1 3 O
3 1 X

Draw
```

```
2 2 O
1 3 X
2 1 O
2 3 X
1 1 O
3 3 X

X wins
```

```
2 2 O
1 1 X
1 2 O
2 1 X
3 2 O

O wins
```



REFERENCES

- Deitel, P. and Harvey Deitel (2022), C How to Program (9th Edition), Pearson Education.

- Thareja, R. (2014), Data Structures Using C (2nd Edition), India: Oxford University Press.

NEXT

Structures, Unions, & Enumerations:

Structures

Unions

Enumerations

VISION

To become an **outstanding** undergraduate Computer Science program that produces **international-minded** graduates who are **competent** in software engineering and have **entrepreneurial spirit** and **noble character**.

MISSION

1. To conduct studies with the best technology and curriculum, supported by professional lecturer
2. To conduct research in Informatics to promote science and technology
3. To deliver science-and-technology-based society services to implement science and technology

Without hard work,
nothing grows but weeds.

UMN
UNIVERSITAS MULTIMEDIA NUSANTARA

INFORMATIKA
UMN

Have patience.
All things are difficult before they become easy.