

IT 112: Introduction to Programming

Dr. Manish Khare
Dr. Bakul Gohel

Lecture 26,27,28





DATA TYPE REVISITED



There are four storage classes in C

- Automatic storage class
- Static storage Class
- External storage class
- Register storage class

Automatic storage class

- Scope: local to the block in which the variable is defined
- Default value is garbage value if variable is not initialized
- Life: till the control remains within the block in which the variable is defined
- Local variable stored at stack segment in memory layout
- It is a default storage class for all local variable

Static storage Class

- Default value is Zero
- Stored at data segment in memory layout
- Scope: Local to the block in which the variable is defined
- Life: value of the variable persists between the different function calls

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

```
int fun_0();  
int fun_1();
```

```
int main()  
{  
    for(int i=0;i<5;i++)  
    {  
        printf("fun_0:%d , fun_1:%d\n", fun_0() , fun_1());  
    }  
    return 0;  
}
```

```
int fun_0()  
{  
    int count = 0; // auto class  
    count++;  
    return count;  
}  
  
int fun_1()  
{  
    static int count = 0; //static class  
    count++;  
    return count;  
}
```

OUTPUT:

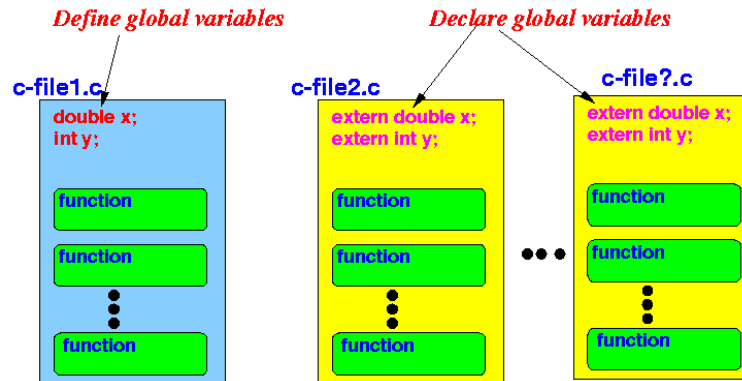
```
fun_0:1 , fun_1:1  
fun_0:1 , fun_1:2  
fun_0:1 , fun_1:3  
fun_0:1 , fun_1:4  
fun_0:1 , fun_1:5
```

Extern Storage class

- Default value is Zero
- Scope: global
- Life: as long as program's execution does not come to an end
- Stored at data segment in memory layout
- extern storage class simply tells us that the variable is defined elsewhere and not within the same block where it is used.

Extern Storage class

- The main purpose of using extern variables is that they can be accessed between two different files which are part of a large program.



```
Program_file.c
#include <functon_file.h>

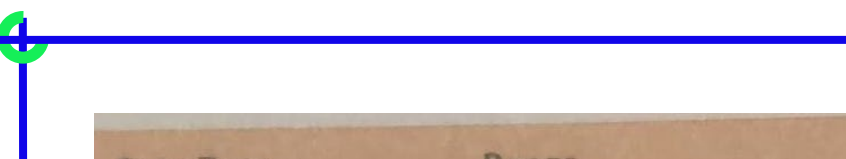
int global_var = 3;
int main()
{
    fun(); // WORKING: it shows '4'
}
```

```
funciton_file.h
void fun(){
    extern int global_var; //Note: 'int global_var' without 'extern' would
                          // simply create a separate different variable
    ++global_var;         // and '++global_var' wouldn't work since it'll
                          // complain that the variable was not initiazed.
    printf("%d", global_var);
}
```

Register Class

- Storage at CPU register (if available)
- Default value: Garbage
- Scope: local to block in which the variable is defined
- Life : Till control remain to block
- CPU registers are very limited in size
- Value stored in the CPU register can be accessed faster than the one that is stored in memory

```
register int i;  
for(i=1;i<=100000;i++)  
    // do some computation
```

Data Type	Range	Bytes	Forma
signed char	-128 to +127	1	%c
unsigned char	0 to 255	1	%c
short signed int	-32768 to +32767	2	%d
short unsigned int	0 to 65535	2	%u
signed int	-2147483648 to +2147483647	4	%d
unsigned int	0 to 4294967295	4	%u
long signed int	-2147483648 to +2147483647	4	%ld
long unsigned int	0 to 4294967295	4	%lu
float	-3.4e38 to +3.4e38	4	%f
double	-1.7e308 to +1.7e308	8	%lf
long double	-1.7e4932 to +1.7e4932	10	%Lf

Note: The sizes and ranges of int, short and long are compiler dependent.
Sizes in this figure are for 32-bit compiler.



PRE-PROCESSOR

PROCESS BEFORE COMPILE

MACRO EXAPNASION

`#define NICK_NAME ACTUAL_ONE`

- Constant
- Faster and more compact processing
- Variable may inadvertently get altered

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

```
#define PI 3.1415
```

```
int main()
```

```
{
```

```
    float radius, area;
```

```
    printf("Enter the radius: ");
```

```
    scanf("%f", &radius);
```

```
    // Notice, the use of PI
```

```
    area = PI*radius*radius;
```

```
    printf("Area=%.2f",area);
```

```
    return 0;
```

```
}
```

`#define AND &&`

`#define OR ||`

FUNCTION like MACRO

- `#define circleArea(r) (PI*r*r)`
- `#define ISDIGIT(y) (y>=48 && y<=57)`

➤ *Do not give space between the macro template and its arguments*

➤ *Be careful*

- `#define SQUARE(n) n*n`
- **Value = 100/SQUARE(4)**
→ Value = 100/4*4 not 100/64

➤ *Make macro function simple and sweet*

```
#include <stdio.h>
#define PI 3.1415
#define circleArea(r) (PI*r*r)

int main() {
    float radius, area;

    printf("Enter the radius: ");
    scanf("%f", &radius);
    area = circleArea(radius);
    printf("Area = %.2f", area);

    return 0;
}
```

#include

one file to be included in another file

➤ #include “filename” // search in current directory and specified path

➤ #include <filename> // search in specified path only

```
// C program to illustrate file inclusion
// <> used to import system header file
#include <stdio.h>
```

```
// " " used to import user-defined file
#include "process.h"
```

```
// main function
```

```
int main()
```

```
{
```

```
    // add function defined in process.h
```

```
    add(10, 20);
```

```
    // mult function defined in process.h
```

```
    mult(10, 20);
```

```
    // printf defined in stdio.h
```

```
    printf("Process completed");
```

```
    return 0;
```

```
}
```

“process.h”

```
// It is not recommended to put function definitions
// in a header file. Ideally there should be only
// function declarations. Purpose of this code is
// to only demonstrate working of header files.
```

```
void add(int a, int b)
```

```
{
```

```
    printf("Added value=%d\n", a + b);
```

```
}
```

```
void multiply(int a, int b)
```

```
{
```

```
    printf("Multiplied value=%d\n", a * b);
```

```
}
```

#ifdef / #ifndef / #undef

```
#include <stdio.h>

#define UNIX 1.0

int main()
{
    #ifdef UNIX
        printf("UNIX specific function calls go here.\n");
        // code specific for UNIX system
    #else
        // code specific for INTEL system
    #endif

    // common code is here

    return 0;
}
```

```
#include <stdio.h>

#define UNIX 1.0
#ifndef UNIX
    #define UNIX 0.5
#endif

int main()
{
    // some code is here using UNIX
    return 0;
}
```

```
#include <stdio.h>

#ifndef __myfile_h
    #define __myfile_h
    #include "myfile.h"
#endif

int main()
{
    // some code is here
    return 0;
}
```



CONSOLE I/O

Formatted I/O

Self Study

Data type		Format specifier
Integer	short signed	%d or %i
	short unsigned	%u
	long signed	%ld
	long unsigned	%lu
	unsigned hexadecimal	%x
	unsigned octal	%o
Real	float	%f
	double	%lf
	long double	%Lf
Character	signed character	%c
	unsigned character	%c
String		%s

```
85      /* decimal */
0213    /* octal */
0x4b    /* hexadecimal */
30      /* int */
30u     /* unsigned int */
30l     /* long */
30ul    /* unsigned long */
```


FORMATED I/O

Escape Seq.	Purpose	Escape Seq.	Purpose
\n	New line	\t	Tab
\b	Backspace	\r	Carriage return
\f	Form feed	\a	Alert
\'	Single quote	\"	Double quote
\\	Backslash		

Self Study

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

```
int main()
```

```
{
```

```
    printf("\n\n-----\\n---\n");
```

```
    printf("Hello World\nBakul");
```

```
    printf("\n\n-----\\t---\n");
```

```
    printf("\tHello World\n\tBakul");
```

```
    printf("\n\n-----\\r---\n");
```

```
    printf("Hello World\rBakul");
```

```
    printf("\n\n-----\\f---\n");
```

```
    printf("Hello World\nBakul");
```

```
    return 0;
```

```
}
```

OUTPUT:

```
-----\n---
```

```
Hello World
```

```
Bakul
```

```
-----\t---
```

```
    Hello World
```

```
    Bakul
```

```
-----\r---
```

```
Bakul World
```

```
-----\f---
```

```
Hello World
```

```
    Bakul
```

Formatted I/O

```
printf("%f",var)  
printf("%-w.df",var)
```

Self Study

Specifier	Description
w	Digits specifying field width
.	Decimal point separating field width from precision (precision means number of places after the decimal point)
d	Digits specifying precision
-	Minus sign for left justifying output in specified field width

Formatted I/O

Self Study

```
#include <stdio.h>
int main( )
{
    int weight = 63 ;
    printf ( "weight is %d kg\n", weight ) ;
    printf ( "weight is %2d kg\n", weight ) ;
    printf ( "weight is %4d kg\n", weight ) ;
    printf ( "weight is %6d kg\n", weight ) ;
    printf ( "weight is %-6d kg\n", weight ) ;
    printf ( "weight is %1d kg\n", weight ) ;
    return 0 ;
}
```

Columns	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0
weight																															
is																															
63																															
kg																															

`printf("%04d",var)` : padding with zero instead of space

Formatted I/O

```
#include <stdio.h>
int main( )
{
    printf ( "%10.1f %10.1f %10.1f\n", 5.0, 13.5, 133.9 );
    printf ( "%10.1f %10.1f %10.1f\n", 305.0, 1200.9, 3005.3 );
    return 0 ;
}
```

This results into a much better output...

```
01234567890123456789012345678901
      5.0      13.5      133.9
    305.0    1200.9    3005.3
```

Self Study

printf("%05.2f",3.14) : 03.14

Formatted I/O

- sscanf() Read from string
- sprintf() write to string

```
#include <stdio.h>
int main()
{
    int i=10;
    char ch='A';
    float a=3.14;
    char str[20];

    printf("%d %c %f\n",i,ch,a);
    sprintf(str, "%d %c %f\n",i,ch,a);
    printf("%s\n",str);

    return 0;
}
```

```
10 A 3.140000
10 A 3.140000
```

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

int main () {
    int day, year;
    char weekday[20], month[20], dtm[100];


    strcpy( dtm, "Saturday March 25 1989" );
    sscanf( dtm, "%s %s %d %d", weekday, month, &day, &year );

    printf("%s %d, %d = %s\n", month, day, year, weekday );

    return(0);
}
```

March 25, 1989 = Saturday

UNFORMATED I/O



```
#include <stdio.h>
int main()
{
    char ch;
    printf("\nType any alphabet: ");
    ch=getchar(); //must be followed by enter key
    printf("you typed: ");
    putchar(ch);
    return 0;
}
```

```
#include <stdio.h>
int main()
{
    char footballer[40];
    puts("Enter name:");
    gets(footballer); // now deprecated
    puts("Congratess...");
    puts(footballer);
    return 0;
}
```



FILE I/O

FILE I/O

FILE *fp ;

file pointer

fp = fopen(file_name, Mode) ;

file name
in string

Operation
mode

FILE *fp;

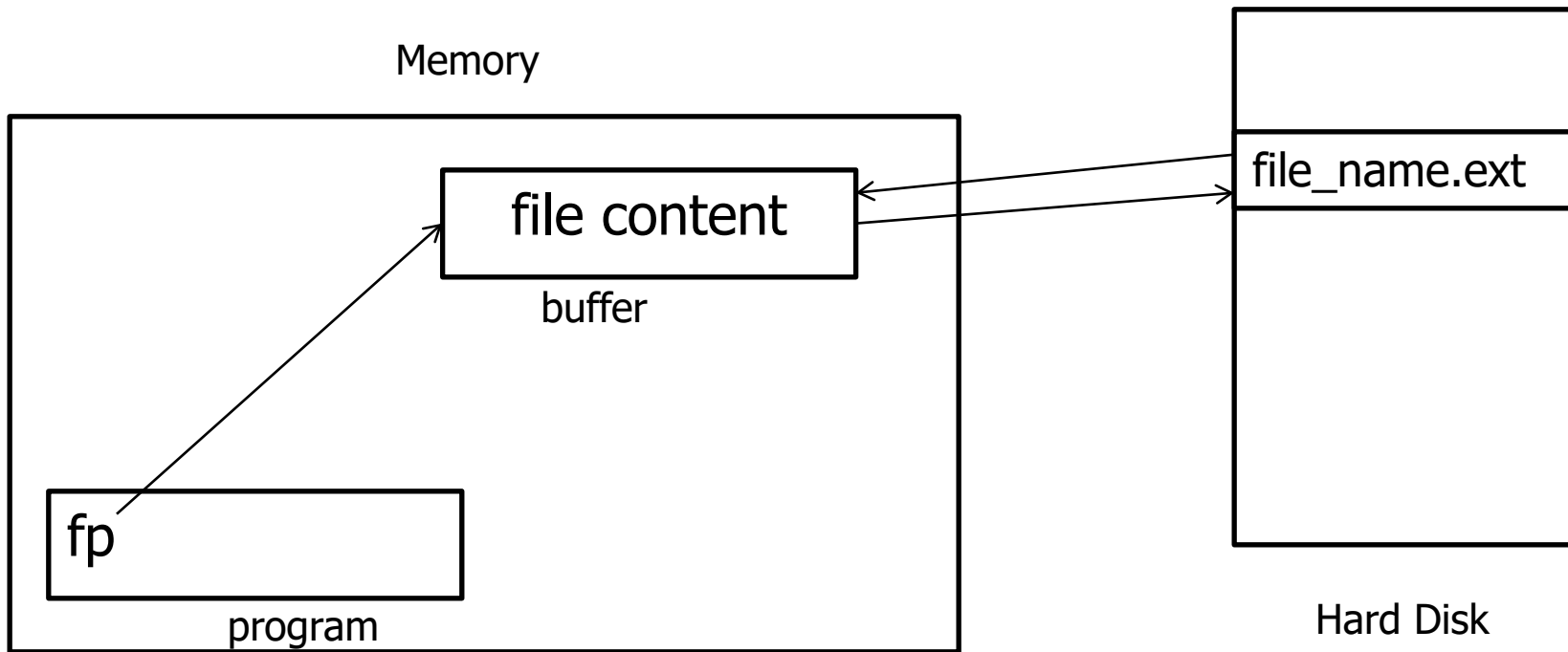
fp = fopen("newfile.txt", "r");

OR

fp = fopen("C:\\myfiles\\newfile.txt", "r");

Mode	Description
r	opens a text file in read mode
w	opens a text file in write mode
a	opens a text file in append mode
r+	opens a text file in read and write mode
w+	opens a text file in read and write mode
a+	opens a text file in read and write mode
rb	opens a binary file in read mode
wb	opens a binary file in write mode
ab	opens a binary file in append mode
rb+	opens a binary file in read and write mode
wb+	opens a binary file in read and write mode
ab+	opens a binary file in read and write mode

FILE I/O



Read and write single char

```
#include <stdio.h>
int main()
{
    char ch;

    /* Pointer for both the file*/
    FILE *fpr, *fpw;
    /* Opening file FILE1.C in "r" mode for reading */
    fpr = fopen("C:\\file1.txt", "r");

    /* Ensure FILE1.C opened successfully*/
    if (fpr == NULL)
    {
        puts("Input file cannot be opened");
    }

    /* Opening file FILE2.C in "w" mode for writing*/
    fpw = fopen("C:\\file2.txt", "w");

    /* Ensure FILE2.C opened successfully*/
    if (fpw == NULL)
    {
        puts("Output file cannot be opened");
    }
}
```



```
int fgetc( FILE * fp );
```

```
int fputc( int c, FILE *fp );
```



```
/*Read & Write Logic*/
while(1)
{
    ch = fgetc(fpr);
    if (ch==EOF)
        break;
    else
        fputc(ch, fpw);
}

/* Closing both the files */
fclose(fpr);
fclose(fpw);

return 0;
}
```

Read and write string

```
char *fgets( char *buf, int n, FILE *fp )
```

```
int fputs( const char *s, FILE *fp )
```

If fgets function encounters a newline character '\n' or the end of the file EOF before they have read the maximum number of characters then it returns only the characters read up to that point including the new line character.

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

```
main() {
```

```
    FILE *fp;  
    char buff[255];
```

```
    fp = fopen("/tmp/test.txt", "r");
```

```
    fgets(buff, 255, fp);  
    printf("2: %s\n", buff );
```

```
    fgets(buff, 255, (FILE*)fp);  
    printf("3: %s\n", buff );  
    fclose(fp);
```

```
}
```

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

```
main() {  
    FILE *fp;
```

```
    fp = fopen("/tmp/test.txt", "w+");  
    fputs("This is testing for fputs...\n", fp);  
    fclose(fp);
```

```
}
```

Read and write string

```
int fscanf(FILE *stream, const char *format [, argument, ...])
```

```
int fprintf(FILE *stream, const char *format [, argument, ...])
```

Similar to sscanf and sprintf as seen above

Only different is first argument that is file pointer instead of string

```
#include <stdio.h>

int main ()
{
    char str [80];
    float f;
    FILE * pFile;

    pFile = fopen ("myfile.txt","w+");
    fprintf (pFile, "%f %s", 3.1416, "PI");
    rewind (pFile);
    fscanf (pFile, "%f", &f);
    fscanf (pFile, "%s", str);
    fclose (pFile);
    printf ("I have read: %f and %s \n",f,str);
    return 0;
}
```



FILE I/O

Command line argument

```
int main(int argc, char *argv[] )
```

argc counts the number of arguments. It counts the file name as the first argument

argv[] contains the total number of arguments. The first argument is the file name always.

Command line argument

progame.c

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

int main(int argc, char *argv[])
{
    int i;
    if( argc >= 2 )
    {
        printf("The arguments supplied are:\n");
        for(i = 0; i < argc; i++)
        {
            printf("%d : %s\n",i, argv[i]);
        }
    }
    else
    {
        printf("argument list is empty.\n");
    }
    return 0;
}
```

- Compiling programme's file give executable file progame.exe
- Run executable file on command prompt

>>./progame.exe file.txt 100

The arguments supplied are:
0 : progame.exe
1 : file.txt
2 : 100



TYPDEF

typedef



```
typedef unsigned char BYTE;
```

```
BYTE b1, b2;
```

```
b1=255
```

```
b2=1;
```

typedef and structure

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

```
struct Point{  
    int x;  
    int y;  
};  
typedef struct Point Point;
```

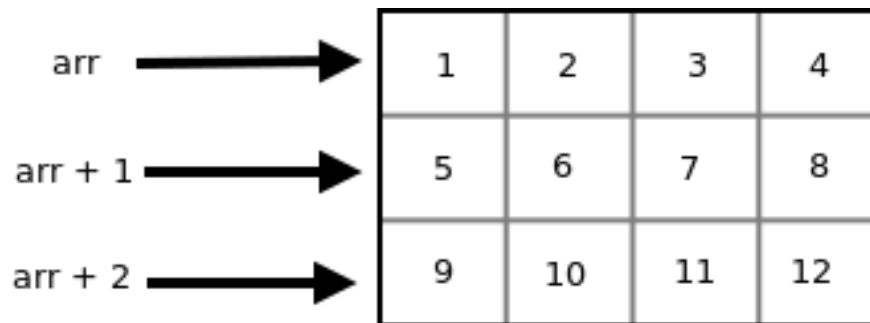
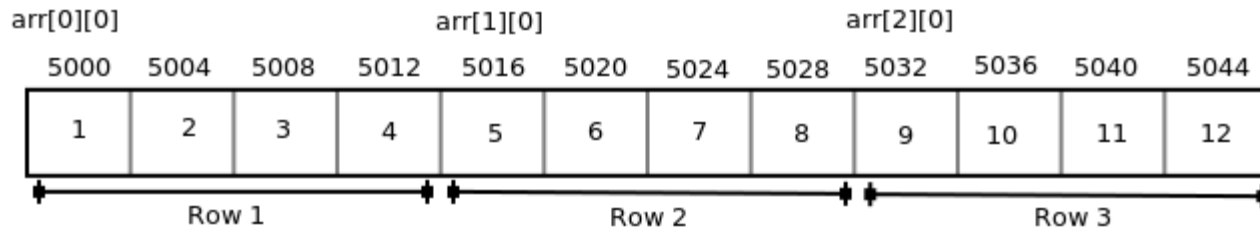
```
int main() {  
    Point p1;  
    p1.x = 1;  
    p1.y = 3;  
    printf("%d \n", p1.x);  
    printf("%d \n", p1.y);  
    return 0;  
}
```

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

```
typedef struct Point{  
    int x;  
    int y;  
} Point;  
int main() {  
    Point p1;  
    p1.x = 1;  
    p1.y = 3;  
    printf("%d \n", p1.x);  
    printf("%d \n", p1.y);  
    return 0;  
}
```

2D array

	Col 1	Col 2	Col 3	Col 4
Row 1	1	2	3	4
Row 2	5	6	7	8
Row 3	9	10	11	12



arr is the **pointer to array** with four elements

Passing 2D array in Function

```
#include <stdio.h>
#define M 3
#define N 3

void print_v1(int arr[M][N]);
void print_v2(int arr[ ][N]);
void print_v3(int *arr);
void print_v4(int (*arr)[N]);

int main()
{
    int arr[][3] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
    printf("\n**** with both dimension ****");
    print_v1(arr);
    printf("\n**** without first dimension ****");
    print_v2(arr);
    printf("\n**** using single pointer ****");
    print_v3((int *) arr);
    printf("\n**** using pointer to array ****");
    print_v4(arr);
    return 0;
}

// **** with both dimension ****
void print_v1(int arr[M][N])
{
    int i, j;
    for (i = 0; i < M; i++)
    {
        printf("\n");
        for (j = 0; j < N; j++)
            printf("%d ", arr[i][j]);
    }
}
```

```
// **** without first dimension ****
void print_v2(int arr[][N])
{
    int i, j;
    for (i = 0; i < M; i++)
    {
        printf("\n");
        for (j = 0; j < N; j++)
            printf("%d ", arr[i][j]);
    }
}

//**** using single pointer ****
void print_v3(int *arr)
{
    int i, j;
    for (i = 0; i < M; i++)
    {
        printf("\n");
        for (j = 0; j < N; j++)
            printf("%d ", *(arr+N*i+j));
    }
}

//**** using pointer to array ****
void print_v4(int (*arr)[N])
{
    int i, j;
    for (i = 0; i < M; i++)
    {
        printf("\n");
        for (j = 0; j < N; j++)
            //printf("%d ", *(*(arr+i)+j));
            printf("%d ", arr[i][j]);
    }
}
```

const

```
#include<stdio.h>
int main(){
    const float PI=3.14;
    printf("The value of PI is: %f",PI);
    return 0;
}
```

Output:
The value of PI is: 3.140000

```
#include<stdio.h>
int main(){
    const float PI=3.14;
    PI=4.5;
    printf("The value of PI is: %f",PI);
    return 0;
}
```

*Constant variable
can not be changed*

Output:
Compile Time Error: Cannot modify a const object

```
#include <stdio.h>
#define SIZE_ARRAY 5

void get_changed(const int arr[],int arr2[]);

int main()
{
    int arr[]= { 10,20,30,40,50};
    int arr2[]={ 0,0,0,0,0};
    get_changed(arr, arr2);
    for(int i=0;i<SIZE_ARRAY;++i)
    {
        printf("%d become %d\n", arr[i], arr2[i]);
    }
    return 0;
}
```

```
void get_changed(const int arr[], int arr2[])
{
    for(int i=0;i<SIZE_ARRAY;++i)
    {
        // arr[i]=arr[i]+2; // it will not allow any content changes through const
        // variable
        arr2[i]=arr[i]+2;
    }
}
```

10 become 12
20 become 22
30 become 32
40 become 42
50 become 52