Homework Hustlers: https://discord.gg/aJ55rZBV

- Wizard.

Write a program to store elements in an array and print them using pointer

```
~

#include <stdio.h>
int main(){
    int a[] = {1,2,3,4,5};
    for (int i = 0; i < sizeof(a)/sizeof(a[0]); i++)printf("%d",*(a+i));
}
```

```
~

[wizard@archlinux w4]$ gcc main.c
[wizard@archlinux w4]$ ./a.out
12345[wizard@archlinux w4]$
```

WRite a program to sum all elements in an array

```
~

#include <stdio.h>
#include <stdlib.h>

int main() {
    int arr[] = {1,2,3,4,5,6}, sum=0;
    for(int i = 0; i<sizeof(arr)/sizeof(arr[0]);i++) sum += *(arr+i);
    printf("%d",sum);

}
```

```
~

[wizard@archlinux tutorial]$ gcc main.c
[wizard@archlinux tutorial]$ ./a.out
21
```

Write a c program to search for an element in an array using pointers;

```
~

#include <stdio.h>

int main() {
    int arr[] = {1, 2, 3, 4, 5, 6}, toSearch = 6, i = 0;
    for (; i < 6 && arr[i] != toSearch; i++);
    printf("%d\n", i < 6 ? i : -1);
    return 0;
}
```

```
~

[wizard@archlinux tutorial]$ gcc main.c
[wizard@archlinux tutorial]$ ./a.out
5
[wizard@archlinux tutorial]$
```

Write a C program to declare a pointer to an integer, assign the address of an integer variable int a = 25; to the pointer, and use the pointer to print both the address and the value of the variable.

```c
~

#include <stdio.h>
int main(){
    int a = 25;
    int *p = &a;
    printf("Address: %p\n",p);
    printf("Value: %d",*p);

}
```

```
~

[wizard@archlinux w4]$ ./a.out
Address: 0x7fffeaa2bcbc
Value: 25[wizard@archlinux w4]$
```

Write a C program that declares an integer variable int x = 50;, a pointer to an integer, and uses the pointer to reference and dereference the variable. Print the address and the value of x using both the variable and the pointer.

```c
~

#include <stdio.h>
int main(){
    int a = 50;
    int *p = &a;
    printf("Address: %p\n",p);
    printf("Value: %d",*p);

}
```

```
~

[wizard@archlinux w4]$ ./a.out
Address: 0x7ffd34f482fc
Value: 50[wizard@archlinux w4]$
```

Write a C program to demonstrate a pointer to a pointer (double pointer) by declaring an integer variable int num = 100;, a pointer to num, and a double pointer to the pointer. Use these to print the value of num through the double pointer

```
~

#include <stdio.h>
int main(){
    int num = 100;
    int *p = &a;
    int **z = &p;
    printf("Value at the pointer: %p\n",p);
    printf("Value through the pointer: %p\n",*p);
    printf("Value at the double pointer: %p\n",z);
    printf("Value through the double pointer: %p\n",**z);

}
```

```
~

[wizard@archlinux w4]$ ./a.out
Value at the pointer: 0x7ffe4a18cbc4
Value through the pointer: 100
Value at the double pointer: 0x7ffe4a18cbc8
Value through the double pointer: 100
[wizard@archlinux w4]$
```

Write a C program to create an array of integers int arr[] = {10, 20, 30, 40, 50};, and use pointer arithmetic to print each element of the array. Demonstrate the scale factor by printing the addresses of the array elements

```
~

#include <stdio.h>
int main(){
    int arr []= {10, 20, 30, 40, 50};
    for (int i = 0; i < sizeof(arr)/sizeof(arr[0]); i++) printf("%p\n", arr + i);
}
```

```
~

[wizard@archlinux w4]$ gcc main.c
[wizard@archlinux w4]$ ./a.out
0x7ffded7b3370
0x7ffded7b3374
0x7ffded7b3378
0x7ffded7b337c
0x7ffded7b3380
[wizard@archlinux w4]$
```

Write a C program to dynamically allocate memory for an array of 5 integers using malloc(). Ask the user to input 5 values, store them in the allocated memory using pointers, and print the values.

```
~

#include <stdio.h>
#include <stdlib.h>
int main(){
    int *ptr = malloc(5 * sizeof(int));
    for (int i = 0; i < 5; i++) scanf("%d",ptr+i);
    for (int i = 0; i < 5; i++) printf("%d",*(ptr+i));
    free(ptr);
}
```

```
~

[wizard@archlinux w4]$ gcc main.c
[wizard@archlinux w4]$ ./a.out
1
2
3
4
5
12345[wizard@archlinux w4]$
```

Write a C program to allocate memory for an integer array of size 5 using malloc(). Assign values to the array elements, print them, and then use free() to deallocate the memory

```
~


#include <stdio.h>
#include <stdlib.h>
int main(){
    int *ptr = malloc(5 * sizeof(int));
    for (int i = 0; i < 5; i++) scanf("%d",ptr+i);
    for (int i = 0; i < 5; i++) printf("%d",*(ptr+i));
    free(ptr);
}
```

```
~

[wizard@archlinux w4]$ gcc main.c
[wizard@archlinux w4]$ ./a.out
1
2
3
4
5
12345
[wizard@archlinux w4]$
```

Write a C program that declares a global variable int globalVar = 10; and a local variable int localVar = 20; inside a function. Use pointers to print the values and addresses of both variables

```
~


#include <stdio.h>
#include <stdlib.h>
int globalVar = 10;
int main(){
    int localVar = 20;
    printf("Local address: %p Value: %d",&localVar, localVar);
    printf("global address: %p Value: %d",&globalVar, globalVar);

}
```

```
~

[wizard@archlinux w4]$ ./a.out
Local address: 0x7fffa8f571b4 Value: 20global address: 0x5946e356d020 Value: 10
[wizard@archlinux w4]$
```

Write a C program that prints the addresses of different segments: text segment (using a function address), initialized data segment (using a global variable), uninitialized data segment (using a global uninitialized variable), heap (using malloc()), and stack (using a local

variable)

```
~

#include <stdio.h>
#include <stdlib.h>
int somdata = 69;
int un_data= 69;
void somfunc(){}
int main(){
    int x;
    printf("Address of a text segment %p\n", &somfunc);
    printf("Address of a initialized data segment %p\n", &somdata);
    printf("Address of a uninitialized data segment %p\n", &un_data);
    printf("Address of a heap segment: %p\n",malloc(1));
    printf("Address of a stack segment: %p\n",&x);
}
```

```
~

[wizard@archlinux w4]$ ./a.out
Address of a text segment 0x5ea3764d7159
Address of a initialized data segment 0x5ea3764da028
Address of a uninitialized data segment 0x5ea3764da02c
Address of a heap segment: 0x5ea392ca26b0
Address of a stack segment: 0x7ffebaf9a4b4
[wizard@archlinux w4]$
```

Write a C program to dynamically allocate memory for an integer array of size 3 using malloc(), then use realloc() to resize it to 6 integers. Ask the user to input values for all 6 elements, print them, and use free() to deallocate the memory.

```
~

#include <stdio.h>
#include <stdlib.h>

int main() {
    int *arr = malloc(3 * sizeof(int));
    arr = realloc(arr, 6 * sizeof(int));
    for (int i = 0; i < 6; i++) scanf("%d", &arr[i]);
    for (int i = 0; i < 6; i++) printf("%d\n", arr[i]);
    free(arr);
    return 0;
}
```

```
~

[wizard@archlinux w4]$ gcc main.c
[wizard@archlinux w4]$ ./a.out
1
2
3
4
5
6
1
2
3
4
5
6
[wizard@archlinux w4]$
```