آموزش - اشاره گر

سال ها پیش که قلی برای خودش بین برنامه نویس ها هنوز کسی بود و هنوز به آینده سفر نکرده بود (ریک و مورتی را هم حتی نمی شناخت)، اتفاقی افتاد که در واقع شروعی بود بر کاهش علاقه ی وی به برنامه نویسی و نهایتاً رها کردن آن.

البته آن ماجرا هیچ ارتباطی به اشاره گر ها ندارد، فقط لازم دانستیم که کمی با پیشینه ی حرفه ای قلی آشنایتان کنیم.

ترم اول دانشگاه یکی از همین روز های پاییزی بود که استاد وارد کلاس شد و اسلاید شو ای با نام "C pointers" را برای کلاس به نمایش قرار داد. قلی با یک ذهنیت منفی در رابطه با اشاره گر ها سر کلاس نشسته بود و فکر می کرد که اشاره گر ها چیز های به غایت خفنی هستند. که البته کاملاً درست فکر می کرد ولی آن هم داستانی برای یک روز دیگر است. در ادامه تمام مطالبی که استاد قلی آن روز برایشان تدریس کرده است را برایتان می آوریم. شاید بتوانید روزی از آن ها استفاده ی مفیدی بکنید، شاید هم نه. اما چیزی که حائز اهمیت است، این است که قسمت بسیار بزرگی از امتحان پایان ترم تان را تشکیل می دهد و از این جهت شباهت عجیبی به "آش کشک" معروف خاله جان دارد.

پوینتر ها، یا همان اشاره گر ها، در حقیقت آدرس متغیر و یا مکان یک متغیر در حافظه را، ذخیره می کنند. نحوه ی نگارش کلی تعریف یک اشاره گر به صورت زیر است:

```
// General syntax
datatype *var_name;

// An example pointer "ptr" that holds
// address of an integer variable or holds
// address of a memory whose value(s) can
// be accessed as integer values through "ptr"
int *ptr;
```

برای آن که بتوانیم از اشاره گر ها در C استفاده کنیم، باید با دو اپراتور زیر آشنا شویم:

- برای دسترسی به آدرس یک متغیر، از اپراتور unary امپرسند (& - ampersand) استفاده می شود. برای مثال &x آدرس متغیر x را به ما می دهد. به مثال زیر توجه کنید:

```
// The output of this program can be different
// in different runs. Note that the program
// prints address of a variable and a variable
// can be assigned different address in different
// runs.

#include <stdio.h>

int  main()

{
    int x;
    // Prints address of x
    printf("%p", &x);
    return 0;`
}
```

- اپراتور دیگر، اپراتور unary ستاره (Asterisk - *) می باشد که برای دو مورد استفاده می شود:

1 . برای تعریف یک اشاره گر: هنگامی که یک اشاره گر در C/C++ تعریف می شود، باید یک * پشت اسم آن قرار دهیم.

```
// C program to demonstrate declaration of
// pointer variables.
#include <stdio.h>
int main()
{
    int x = 10;

    // 1) Since there is * in declaration, ptr
    // becomes a pointer varaible (a variable
    // that stores address of another variable)
    // 2) Since there is int before *, ptr is
```

```
    // pointer to an integer type variable
    int *ptr;

    // & operator before x is used to get address
    // of x. The address of x is assigned to ptr.
    ptr = &x;

    return 0;
}
```

2 . برای دسترسی به مقداری که یک اشاره گر به آن اشاره می کند نیز از همین اپراتور استفاده می کنیم، که مقدار ذخیره شده در آدرس عمل وندش را باز خواهد گرداند.

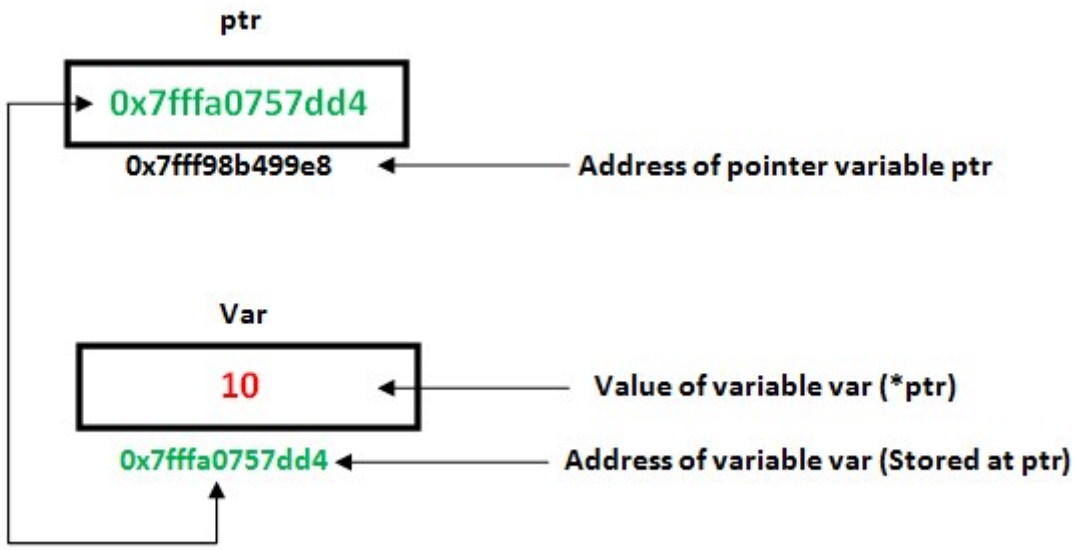
```
// C program to demonstrate use of * for pointers in C
#include <stdio.h>

int main()
{
    // A normal integer variable
    int Var = 10;

    // A pointer variable that holds address of var.
    int *ptr = &Var;

    // This line prints value at address stored in ptr.
    // Value stored is value of variable "var"
    printf("Value of Var = %d\n", *ptr);

    // The output of this line may be different in different
    // runs even on same machine.
    printf("Address of Var = %p\n", ptr);

    // We can also use ptr as lvalue (Left hand
    // side of assignment)
    *ptr = 20; // Value at address is now 20

    // This prints 20
    printf("After doing *ptr = 20, *ptr is %d\n", *ptr);
```

```
    return 0;
}
```

خروجی : (دقت کنید که آدرس ها ممکن است از سیستم به سیستم تفاوت داشته باشد)

```
Value of Var = 10
Address of Var = 0x7fffa057dd4
After doing *ptr = 20, *ptr is 20
```

در تصویر زیر می توانید نمایشی از برنامه ی بالا را ببینید:



عملیات حسابی روی اشاره گر ها

تعداد محدودی عملیات می توان بر روی اشاره گر ها انجام داد، یک پوینتر می تواند:

- اینکریمنت شود! (++)
- دیکریمنت شود! (--)
- عددی می تواند به یک اشاره گر افزوده شود (+ یا =+)
- عددی می تواند از یک اشاره گر کم شود (- یا =-)

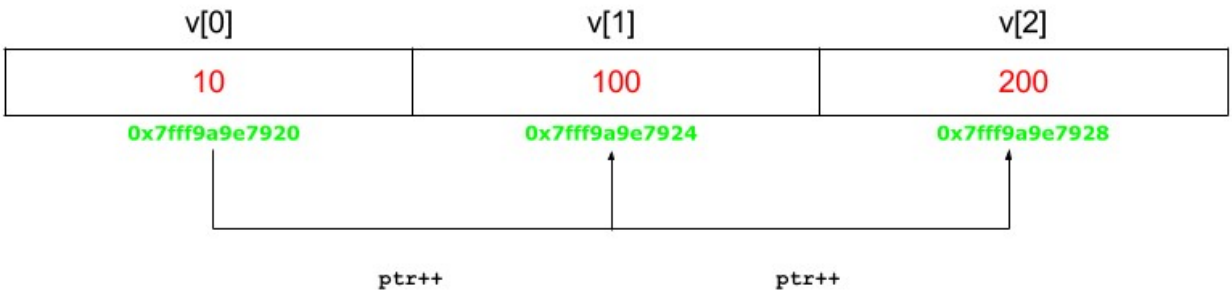
(دقت کنید که عملیات حسابی روی اشاره گر ها غیر از حین استفاده از آرایه ها، بی معنی هستند. در ادامه بیشتر توضیح خواهیم داد.)

```
// C++ program to illustrate Pointer Arithmetic
// in C/C++
#include <bits/stdc++.h>

// Driver program
int main()
{
    // Declare an array
    int v[3] = {10, 100, 200};

    // Declare pointer variable
    int *ptr;

    // Assign the address of v[0] to ptr
    ptr = v;

    for (int i = 0; i < 3; i++)
    {
        printf("Value of *ptr = %d\n", *ptr);
        printf("Value of ptr = %p\n\n", ptr);

        // Increment pointer ptr by 1
        ptr++;
    }
}
```

خروجی :

```
Value of *ptr = 10
Value of ptr = 0x7ffcae30c710

Value of *ptr = 100
Value of ptr = 0x7ffcae30c714
```

```
Value of *ptr = 200
Value of ptr = 0x7ffcae30c718
```



نام آرایه ها به عنوان اشاره گر

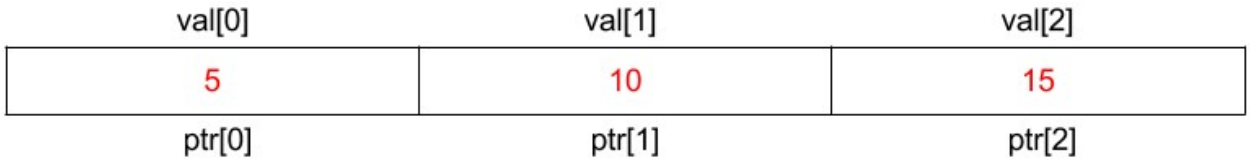
یکی از نکات جالب توجه در رابطه با اشاره گر ها، ارتباط آن ها با آرایه هاست. نام یک آرایه در واقع مانند یک اشاره گر به آدرس اولین عنصر آن آرایه عمل می کند. برای مثال، اگر آرایه ای با نام val داشته باشیم، آنگاه **val** و **&val[0]** می توانند به جای هم استفاده شوند! (هم معنی هستند) و یا برای مثال، **val+5** تفاوتی با **&val[5]** ندارد.

```
// C++ program to illustrate Array Name as Pointers in C++
#include <bits/stdc++.h>
using namespace std;

void gholi()
{
    // Declare an array
    int val[3] = { 5, 10, 20 };

    // Declare pointer variable
    int *ptr;

    // Assign address of val[0] to ptr.
    // We can use ptr=&val[0];(both are the same)
    ptr = val ;
    cout << "Elements of the array are: ";
    cout << ptr[0] << " " << ptr[1] << " " << ptr[2];
```

```
    return;
}

// Driver program
int main()
{
    gholi();
    return 0;
}
```

خروجی :

```
Elements of the array are: 5 10 20
```



حال اگر این "ptr" به عنوان یک آرگومان به یک تابع فرستاده شود، آرایه ی "val" به طریق مشابه قابل دسترسی خواهد بود.

اشاره گر ها و آرایه های چند بعدی

آرایه ی زیر را در نظر بگیرید:

```
int nums[2][3]  =  { {16, 18, 20}, {25, 26, 27} };
```

به طور کلی، nums[i][j] معادل است با *(*(nums+i)+j)، به جدول زیر دقت کنید:

Pointer Notation	Array Notation	Value
*(*nums)	nums[0] [0]	16
*(*nums+1)	nums[0] [1]	18
*(*nums+2)	nums[0] [2]	20
((nums + 1))	nums[1] [0]	25

((nums + 1)+1)	nums[1] [1]	26
((nums + 1)+2)	nums[1] [2]	27

دو مورد از کاربرد های اشاره گر ها تولید ساختمان های داده ی بهینه و یا تخصیص حافظه ی پویا می باشد که با مثالی از هر دو مورد در ادامه ی جلسات کلاستان آشنا خواهید شد.

به عنوان تمرین، برنامه ای بنویسید که تعدادی عدد را در یک آرایه ذخیره کند و اعداد را با استفاده از اسم آرایه عنوان اشاره گر دریافت کنید. سپس، باز هم با استفاده از اشاره گر ها، آرایه را به صورت صعودی مرتب کنید.

خطا در اشاره‌گر ها

که کد پیوست‌شده حاوی خطاهایی در استفاده از اشاره‌گرهاست که مانع از کارکرد صحیح آن شده و ممکن است باعث بروز خطا به هنگام اجرای برنامه شوند. کد زیر را با حداقل تغییر اصلاح کرده و بارگذاری کنید. اصلاحات انجام‌شده روی کد را کامنت کنید.

تابع `main()` فاقد هرگونه اشکال نوشته شده است تا Prototype توابع را بدانید. تغییری در آن ایجاد نکنید. همچنین تمام کتاب‌خانه‌های قابل استفاده برای حل سوال Include شده‌اند و اجازه اضافه‌کردن به آن‌ها را ندارید.

نکته: اشاره‌گرها را با عملگرهایی همچون `[]` جایگزین نکنید.

کد :

```c
#include <stdio.h>
#define SIZE 100

void read(int *array, int size) {
    int i;
    for (i = 0; i < size; i++)
        scanf("%d", *array + i);
}

void swap(int *a, int *b) {
    int *temp = a;
    a = b;
    b = temp;
}

void print(const int *array[], int size) {
    int i;
    for (i = 0; i < size; i++)
        printf("%d ", **(array + i));
}


```

```c
void sort(int array[], const int size) {
    int *end = array + size, i;
    for (i = array; i != end - 1; i++)
        if (i > (i + 1))
            swap(i, i + 1);
}

int isSorted(const int array[], const int *size) {
    int i;
    for (i = 0; i < *size - 1; i++)
        if (*(array + i) > *(array + i + 1))
            return 0;
    return 1;
}

int main() {
    int arr[SIZE], len;
    scanf("%d", &len);
    read(arr, len);
    sort(arr, &len);
    printf("%s\n", isSorted(arr, &len) ? "All is good!" : "This isn't goi
    print(arr, len);
    return 0;
}
```

مرتب‌سازی رشته‌ها

- محدودیت زمان: ۱ ثانیه
- محدودیت حافظه: ۵۰ مگابایت

در این برنامه شما باید به تعداد نامعلومی رشته از کاربر بگیرید. روش دریافت ورودی به این شکل است که کاربر شروع به وارد کردن کلمه‌های موردنظر خود می‌کند. این عملیات تا جایی ادامه پیدا می‌کند که کاربر `0` را وارد کند. برنامه باید تمام کلماتی را که تا اینجا وارد شده است، به ترتیب حروف الفبای انگلیسی چاپ کند.

تذکر:

- حروف وارد شده تنها شامل حروف کوچک و بزرگ انگلیسی هستند.
- حروف بزرگ به حروف کوچک اولویت دارند.
- برای اطلاعات بیشتر درباره‌ی نحوه‌ی ذخیره‌سازی کاراکترها، عبارت `ASCII Table` را گوگل کنید.
- کلمات با کاراکتر کمتر، به کلمات با کاراکتر بیشتر اولویت دارند؛ به عنوان مثال، `a` زودتر از `aa` چاپ می‌شود.
- در مرتب سازی کلمات، تنها دو حرف اول آن نگاه می‌کنیم؛ بنابراین اگر دو کلمه در حرف اول مشترک بودند، به سراغ حرف دوم می‌رویم. اگر در حرف دوم هم یکسان بودند، هر دو کلمه از نظر مرتب‌سازی هم‌ارزش‌اند؛ بنابراین کلمه‌ای که در ورودی زودتر وارد شده است، زودتر چاپ می‌شود.
- استفاده از توابع آماده مرتب‌سازی رشته‌ها در این سوال، قابل قبول نیست و کد مرتب‌سازی به طور کامل باید توسط شما نوشته شده باشد.

ورودی

در خط اول ورودی تعدادی کلمه می‌آید که با صفر خاتمه می‌یابد.(تعداد و مجموع طول کلمات کمتر از ۱۰۰۰ است.)

خروجی

در خروجی تمامی کلمات داده شده را به ترتیب حروف الفبای انگلیسی چاپ کنید.

مثال

نمونه ورودی:

```
alireza Mohammad Arash anahita sarah Milad john Alireza Maryam 0
```

نمونه خروجی :

```
Alireza Arash alireza anahita john Maryan Milad Mohammad sarah
```