به نام یزدان پاک

Pointers in C++

💡آموزش اشاره گر ها در برنامه نویسی C++

گردآورنده:سعید اکبری

استاد راهنما:استاد علیاری

دوستان عزیز دقت داشته باشند که داریم کم کم از درب ورودی برنامه نویسی رد می شیم.

شاید پیش خودتون بگویید که پس مطالب قبلی چی بودند؟ باید عرض کنم که می شود گفت که اونا الفبای ابتدایی برنامه نویسی بودند. از این به بعد کم کم با مفاهیم اساسی برنامه نویسی روبرو خواهیم شد که درک هر چه بهتر این مفاهیم می تواند ما را در زمینه برنامه نویسی C++یا هر برنامه نویسی استاندارد دیگری به برتری برساند. اصول و مفاهیم کلی برنامه نویسی یکیه اما در زبانهای مختلف، راهها و قوانین نگارشی متفاوتی برای پیاده سازی کدها وجود دارد.

از این به بعد باید مفاهیمی را درک کنیم که در برنامه نویسی نقش بسزایی را ایفا می کنند، پس سعی کنید که مطالب را با دقت پیگیری کرده و به مثال ها و نوشتن برنامه بپردازید تا که به این مفاهیم تسلط پیدا نمایید.

تعریف مفاهیم

اشاره گر : متغیری است که آدرس خانه های سیستم را در خود نگه می دارد (آدرس هر متغیر در حافظه اشاره گر است).

متغیر هایی را که تا کنون با هم تعریف کردیم جدای از نوعشان همگی در خانه های حافظه ذخیره می شوند و هر کدام با توجه به طول نوع تعدادی از خانه ها را در حافظه اشغال می کنند.

آدرس خانه های حافظه : حافظه کامپیوتر از مجموعه ای از بایتهایی که هر کدام شامل 8 بیت هستند تشکیل شده است. برای دسترسی به هر یک از این بایتها که به آنها خانه های حافظه گفته می شود شماره آدرسی وجود دارد که به ترتیب به هر یک داده می شود. به این شماره ردیف ها آدرس خانه حافظه گفته می شود.

آدرس متغیر : آدرس اولین بایت از حافظه که به یک متغیر اختصاص می یابد آدرس آن متغیر نامیده می شود.

می دانیم که متغیر ها در حافظه ذخیره می شوند اما مقدار حافظه مورد نیازشان با یکدیگر متفاوت است:

- char : 1 Byte
- int : 2-4 Byte
- float : 4 Byte
- double : 8 Byte

مثلا اگر ما متغیری از نوع int را تعریف کنیم 2 یا 4 بایت را اشغال می کند و به این معنی است که تعداد 4 تا 8 خانه پشت سر هم از حافظه را اشغال می کند که آدرس آن اولین آدرس خانه حافظه در نظر گرفته خواهد شد.

در مورد مزایای استفاده از اشاره گرها در C++یا زبانهای برنامه نویسی دیگر همین قدر باید بگم که چیز خوبیه و سرعت، کارایی، دسترسی و ... را بالا می برد.

آدرس حافظه را فقط می توان در یک متغیر از نوع اشاره گر تعریف نمود:

```
Type *Name;

int *P;
```

چنانچه در بالا می بینیم برای تعریف متغیری از نوع اشاره گر، ابتدا نوع آن و سپس نام که از قانون نامگذاری برای متغیرها تبعیت می کند استفاده خواهیم کرد با این تفاوت که قبل از نام از علامت * برای نشان دادن تعریف اشاره گر بهره می گیریم.

تعریف اشاره گر بالا را می توان به طریق زیر تفسیر نمود:

- pمتغیر اشاره گری از نوع int است.
- pآدرس خانه هایی از حافظه را نگهداری می کند که محتویات آن خانه ها، مقادیری از نوع int هستند.
- pمتغیری است که به محل هایی از حافظه که محتویاتی از نوع int دارد اشاره می کند.

دقت کنید که نوع یک اشاره گر به نوع متغیرهایی که به آدرس آنها اشاره می کند بستگی دارد و باید با آن یکسان باشد. یعنی نوع اشاره گری که به آدرسهایی که شامل متغیر هایی از نوع duoble است نمی تواند int یا char یا هر چیز دیگری غیر از همان duoble باشد.

تعریف مفاهیم

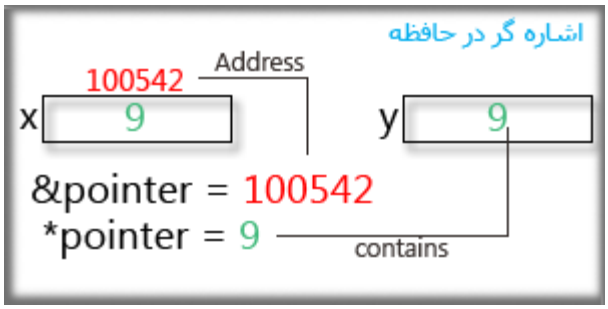
عملگرهای اشاره گر در C++

عملگر & : این عملگر دارای یک عملوند است و آدرس عملوند خود که نام یک متغیر است را مشخص می نماید.

عملگر * : این عملگر هم دارای یک عملوند است و محتویات (مقادیر) جایی که عملوندش (نام اشاره گر) به آنجا اشاره می کند را مشخص می کند.

```
int *pointer, x, y;   // define three variables whit int type
x = 9;                // equal x to 9
pointer = &x;        // pointer contains Address of x
y = *pointer         // y equal the contains of content address of pointer
```

در کد C++بالا در خط اول 3 متغیر از نوع int که یکی از آنها یک اشاره گر است تعریف شده اند. خط دوم مقدار 9 را در متغیر x قرار می دهد. متغیر x با مقدار 9 در جایی از حافظه ذخیره شده است که خط سوم آدرس آن محل را در متغیر اشاره گر pointer قرار می دهد یعنی pointer به جایی اشاره می کند که متغیری بنام x و از نوع int که مقداری برابر 9 را دارد اشاره می کند. در آخرین خط محتویات جایی که آدرس آن در اشاره گر pointer ذخیره شده است در متغیر y قرار می گیرد یعنی مقدار 9.




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

void main()
{
    int firstValue, secondValue;
    int *myPointer;

    myponter = &firstValue;
    *myponter = 10;       // firstValue = *myponter = 10
    myponter = &secondValue;
    *myponter = 20;       // secondValue = *myponter = 20
    cout << "The firstValue is " << firstValue;
    cout << "\nThe secondValue is " << secondValue;

    getch();
}
```

```
The firstValue is 10
The secondValue is 20
```

در خطوط 6 و7 برنامه C++بالا سه متغیر که 2 متغیر معمولی از نوع int تعریف شده است و یک متغیر pointer. دستور خط 9 آدرس متغیر firstValue را در اشاره گر قرار می دهد. در خط بعد محتویات جایی که اشاره گر به آنجا اشاره می کند را برابر با مقدار 10 در نظر می گیریم. چون جایی که متغیر myPointer به آنجا اشاره می کند آدرس متغیر firstvalue است پس مقدار 10 در آنجا و در واقع در متغیر firstValue ریخته خواهد شد و به همین ترتیب برای متغیر secondValue.

عملیات روی اشاره گرها

کلا 3 نوع عمل را می شود بر روی pointer انجام داد:

- 1.انتساب اشاره گر ها به همدیگر
- 2.اعمال محاسباتی جمع و تفریق

- .3مقایسه

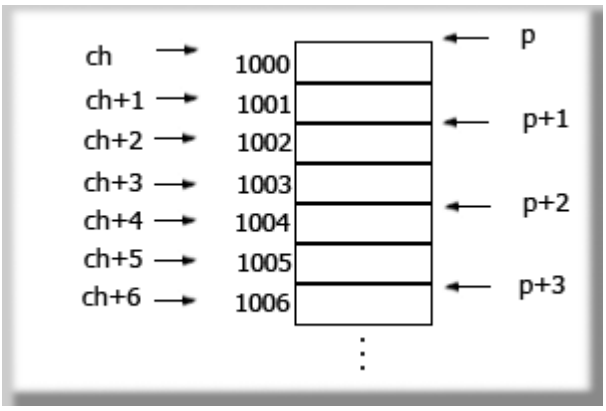
- انتساب اشاره گرها به همدیگر:

وقتی دو اشاره گر (pointer)را برابر با یکدیگر قرار دهیم با 2 حالت مواجه خواهیم شد، یکی آدرس و دیگری مقدار است. به کد زیر دقت نمایید:

```
int *p1, *p2, *p3, a, b, c;
a = 50;
b = 100;
c = 200;
p1 = &a;  // p1 = Address a and *p1 = 50 and assume that &p1 = 10000
p2 = &b;  // p2 = Address b and *p2 = 100 and assume that &p2 = 10020
p3 = &c;  // p3 = Address c and *p3 = 200 and assume that &p3 = 10040
*p1 = *p2;   // *p1 = *p2 = 100 and &p1 = 10000 and &p2 = 10020
p1 = p3;   // &p1 = &p3 = 10040 and *p1 = *p3 = 200
```

در کد بالا و در خط 5 اشاره گر p1 به جایی که متغیر a در حافظه ذخیره شده اشاره می کند بنابراین p1*برابر با مقدار آن آدرس یعنی 50 خواهد شد. در دستور خط بعدی آدرس جایی که متغیر b در آنجا ذخیره شده درون اشاره گر p2 قرار می گیرد یعنی به آن آدرس اشاره می کند و چون محتویات آن آدرس برابر با 100 است پس مقدار p2*برابر با 100 می شود. در سطر 8 با نوشتن این دستور فقط محتویات جایی که p1 به آنجا اشاره می کند درون درون جایی که p2 به آنجا اشاره می کند ریخته می شود و آدرس دو اشاره گر تغییر نخواهد کرد. اما در سطر 9 آدرس جایی که p3 به آنجا اشاره می کند در اشاره گر p1 قرار می گیرد، بنابراین جایی که p1 به آنجا اشاره می کرد برابر با جایی می شود که p3 اشاره می کند که به ازاء این تغییر بنابراین محتویات جایی که p1 به آنجا اشاره می کرد به مقدار 200 تغییر می کند?OK .

- اعمال جمع و تفریق:



وقتی اشاره گری را افزایش یا کاهش می دهیم در واقع اینکار را با آدرسها انجام خواهیم داد و متناسب با طول آن متغیر کم یا زیاد می کنیم. در شکل بالا p متغیری از نوع int است پس با هر واحد افزایش، pointer 2خانه یعنی 2 بایت در حافظه پیش خواهد رفت و متغیر char از نوع char در نظر گرفته شده که با هر واحد افزایش یا کاهش فقط یک خانه کم یا زیاد می شود.

- مقایسه اشاره گرها:

وقتی در مورد اشاره گرها صحبت می کنیم ذهن خود را به سمت آدرس حافظه ببرید. وقتی دو اشاره گر با یکدیگر برابرند که هر دو به یکجا اشاره نمایند، در این صورت هر تغییر در آن آدرس منجر به تغییر هر دو اشاره گر خواهد شد اما اگر مقدار آدرس دو اشاره کر برابر نباشد هر چند که محتویات آن دو آدرس مختلف با هم برابر باشد نمی توان گفت که آن دو با هم برابر هستند زیرا با تغییر در محتوا یا آدرس یکی از آن دو دیگری تغییری نمی کند.

```
int *p1, *p2, char1, char2;
char1 = char2 = 'Q';
p1 = &char1;
p2 = &char2;
if( p1 == p2 )
    ...
else
    ...
```

د C++ بالا چون اشاره گرها هر یک به آدرسهای مختلفی اشاره می کنند هر چند مقادیر آن آدرسها یکی است اما با هم برابر نیستند .

اشاره گر ها به عنوان آرگومان های توابع:

همچنان که پیشتر در فصل توابع در مورد آرگومان های توابع بحث شد می توان از اشاره گر ها بعنوان آرگومان تابع استفاده نمود. در کد زیر از تکنیک inline function بهره می گیریم:

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

inline void foo(int &y)
{
    cout << "y=" << y << "\n";
    y = 6;
    cout << "y=" << y << "\n";
}

void main()
{
    int x = 5;
    cout << "x=" << x << "\n";
    foo(x);
    cout << "x=" << x << "\n";
    getch();
}
x=5
y=5
y=6
x=6
```

در کد برنامه نویسی C++تابعی با نام foo که دارای یک آرگومان از نوع اشاره گر است را تعریف نمودیم. این تابع حاوی آدرس پارامتر ورودی خود است پس یک اشاره گر است. در درون تابع foo ابتدا یک بار مقدار متغیر ورودی که در این تابع نامش y و در تابع main نامش x است راچاپ می کند اما با استفاده از مراجعه به آدرس آن متغیر.

از قبل یادگرفتیم که در استفاده از توابع با آرگومان های معمولی، با آدرس آن متغیر کامپایلر کاری ندارد بلکه کپی از آنرا تهیه نموده و از آن استفاده می کند، پس هر گونه تغییر در مقدار متغیر ها در این حالت با برگشت به تابع main از بین خواهد رفت چون کامپایلر آن کپی را دور می اندازد و اینبار و در تابع main یا هر حوزه جدید دیگری دوباره از آن متغیر کپی گرفته و استفاده می کند.

اما در مورد اشاره گر ها دیگر چون مستقیما در محل حافظه تغییرات صورت می گیرد پس با کپی گرفتن مجدد در هر حوزه یا تابع دیگری مقادیر جدید را بدست می آورد، بنابراین در کار با اشاره گر ها تغییرات آنی اعمال می شوند ومتغیر ها حوزه ای ندارند.

خوب، در سطر 7 مقدار متغیر به 6 تغییر می کند و در سطر بعدی عدد 6 بعنوان مقدار متغیر جاری درج می گردد. در بازگشت به تابع mainچون آرگومان اشاره گر بوده پس تغییرات در آدرس متغیر اعمال شده و با مراجعه کامپایلر به آدرس مورد نظر و چاپ مقدار آن دوباره عدد 6 چاپ می شود در صورتی که اگر آرگومان معمولی بود 5 چاپ می شد یعنی همان مقدار اولیه متغیر در این تابع. فراموش نکنید که برنامه از تابع اصلی آغاز می شود بنابراین در ابتدا مقدار x که 5 است چاپ خواهد شد.

برای درک تفاوت میان روش ارسال پارامتر ها با استفاده از مقادیر یا ارجاع (آدرس) مثالی دیگری را بررسی خواهیم نمود:

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

void pass(int, int *);

void main()
{
    int x = 1, y=1;
    pass(x, &y);
    cout << "x=" << x << "\n";
    cout << "y=" << y << "\n";
    getch();
}
void pass(int a, int *b)
{
    cout << "x=" << ++a << "\n";
    cout << "y=" << ++*b << "\n";
}
x=2
y=2
x=1
y=2
```

با اجرای کد برنامه C++بالا، کامپایلر در سطر 6 ابتدا وارد تابع main خواهد شد. دو متغیر از نوع int با نامهای x,y تعریف شده که هر کدام دارای مقدار برابر یک هستند. در سطر 8 کامپایلر با فراخوانی تابع pass به سطر 14 می رود. در این تابع دو پارامتر که یکی int و دیگری اشاره گر است را می بیند. ما با اینکار مقدار x را با ارسال پارامتر بروش مقدار و y را با استفاده از ارسال پارامتر بروش ارجاع انجام داده ایم، یعنی مقدار x را و آدرس y را به تابع می فرستیم.

در سطر 16 ابتدا یک واحد به x اضافه شده وسپس چاپ می شود که مقدار 2 را خواهیم داشت.

در سطر 17 ابتدا یک واحد به مقدار جایی که آدرس آن محل به عنوان پارامتر (اشاره گر) به تابع ارسال شده اضافه خواهد شد و بعد 2 چاپ خواهد شد.

با پایان بلوک تابع pass کامپایلر باز به تابع main بازگشته و ادامه دستورات را از سطر 10 از سر می گیرد.

در سطر 10 چون x با مقدار به تابع ارسال شده بود پس مقدار آن در ابن تابع همان 1 است پس اینبار 1 چاپ خواهد شد.

اما در سطر 11 چون با آدرس متغیر y کار کردیم پس مقدار کنونی در این تابع هم دستخوش تغییرات در تابع pass خواهد بود و عدد 2 برای آخرین دستور چاپ می شود و برنامه پایان میابد.

امیدوارم که توانسته باشم مفهوم ارسال با مقدار و ارجاع یا آدرس را رسانده باشم. اما اگر متوجه نشدید اصلا نگران نباشید و با کمی تمرین براحتی مطلب را درک خواهید نمود.

اشاره گر ها و آرایه ها:

دیدیم که عناصر آرایه پشت سر هم در حافظه قرار می گیرند و بدانید که نام آرایه یک اشاره گر است. در واقع نام آرایه در برگیرنده آدرس اولین عنصر آرایه در حافظه است و چونکه عناصر آرایه بدون فاصله و به ترتیب در حافظه قرار می گیرند بنابراین می توان با داشتن نام آرایه که در واقع یک نوع اشاره گر است به کلیه عناصر آن در برنامه نویسی زبان C++دست پیدا کرد.

در قطعه برنامه C++زیر به بررسی ارتباط اشاره گر ها و آرایه ها می پردازیم:

```
char array[] = {'c','s','h','a','r','p'};
char *pointer;
pointer = array;            // آرایه و اشاه گر هر دو به یکجا یعنی ابتدای آرایه اشاره می کنند
*pointer = array[0];        // = 'c'
*(pointer+1) = array[1];    // = 's'
pointer[3] = *(array+3);    // = 'a'
*pointer = *array;          // = 'c'
```

همانطور که در بالا ملاحظه می کنید به راحتی می توان از اشاره گر بجای آرایه ها استفاده کرد.

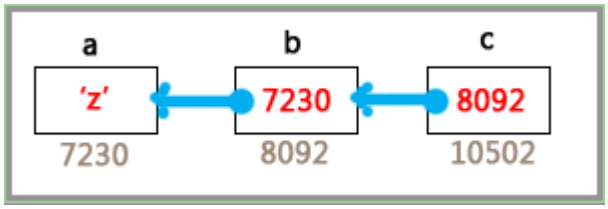
```
#include <iostream.h>
#include <conio.h>

void main()
{
    int numbers[5];
    int *p;

    p = numbers;
    *p = 10;
    p++;
    *p = 20;
    p = &numbers[2];
    *p = 30;
    p = numbers+3;
    *p = 40;
    p = numbers;
    *(p+4) = 50;
```

```
    for(int n=0 ; n<5 ; n++)
        cout << numbers[n] << ", ";
    getch();
}
10, 20, 30, 40, 50
```

ر قطعه برنامه ++Cبالا آرایه ای بنام numbers و یک اشاره گر بنام p تعریف کرده ایم. در سطر 9 اشاره گر را برابر با عنصر اول آرایه قرار می دهیم پس هر دو به یکجا اشاره می کنند. سپس در سطر 10 بجایی که اشاره گر اشاره می کند و برابر با عنصر اول آرایه است مقدار 10 را می دهیم. در سطر بعد اشاره گر را یکواحد افزایش می دهیم که با اینکار به خانه بعدی یا همان عنصر دوم آرایه اشاره می کند و باز مقدار محتوای آن خانه را برابر با 20 قرار می دهیم. در کد بالا سعی شده تا انواع برابری های آرایه و اشاره گر را نشان دهم تا برای شما عزیزان بحث جذابتر و روشنتر گردد. به همین روال ما به اشاره گر مقدار می دهیم و چون اشاره گر و آرایه با هم برابرند پس آرایه نیز با اینکار مقداردهی می شود. سپس با استفاده از یک حلقه تکرار for در سطرهای 19و20 مقادیر آرایه را چاپ می کنیم.

خدمت عزیزان باید عرض کنم که از اشاره گر ها نیز می توان بجای رشته ها استفاده نمود و مطالب مانند مطالب آرایه ها دنبال خواهد شد پس از بیان این مورد صرفنظر می کنم.

اشاره گر به اشاره گر : (pointer to pointer)



ر شکل بالا، aمتغیری از نوع char است که دارای مقدار z می باشد و b اشاره گری است که شامل آدرس a است و c نیز اشاره گر دیگری است که آدرس b را در بر دارد.

برای تعریف اشاره گری که به اشاره گر دیگری اشاره می کند کافی است که از یک * دیگری استفاده نمایید:

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

void main()
{
    char a = 'z';
    char *b, **c;

    b = &a;
    c = &b;

    cout << a << "\n"
    cout << *b << "\n";
    cout << **c << "\n";
    cout << b << "\n";
    cout << *c << "\n";
    getch();
}
```