اشاره گرها

در این بخش مبحث مهم اشاره گرها در C معرفی می شود. اشاره گر متغیری است که آدرس متغیر دیگری را نگه می دارد که اصطلاحا گفته می شود دارد به آن متغیری اشاره می کند. به دو عملگر * (address-of operator) و & (indirection operator) با اشاره گرها نیاز می شود. اشاره گرها روش قدرتمند و انعطاف پذیری برای کار با انواع داده ها در برنامه نظیر آرایه ها، رشته ها و ساختمان ها است. کاربرد اشاره گرها برای استفاده از حافظه پویا هم در اینجا توضیح داده خواهد شد.

متغیر اشاره گر
حافظه پویا
محاسبات روی اشاره گر
اشاره گر به آرایه
اشاره گر به ساختمان
اشاره گر به اشاره گر
اشاره گر به تابع
اشاره گر و پارامتر مرجع
ارگومان های خط فرمان

اشاره گرها بخش قدرتمند و دشوار C و C++ هستند. اکثر مبتدیان در درک مناسب اشاره گرها مشکل دارند. متغیرهای استاندارد برچسب هایی هستند که برای تعیین بخش هائی از حافظه کنار گذاشته می شوند تا داده از نوع خاصی را ذخیره کنند درحالیکه یک اشاره گر به بخش هایی از حافظه که توسط متغیر دیگری اشغال شده اشاره می کند. یک متغیر استاندارد یک مقدار را ذخیره می کند اما یک اشاره گر آدرس محلی در حافظه را ذخیره می کند.

متغیر اشاره گر

یک متغیر اشاره گر (pointer variable) متغیری است که حاوی آدرس داده ، متغیر دیگر یا تابع است. برای ذخیره آدرس اولین مرحله اعلان متغیر اشاره گر است. اعلان متغیر اشاره گر تقریبا مشابه متغیرهای دیگر است تنها باید قبل از نام متغیر علامت ستاره (*) برای نشان داده اینکه یک اشاره گر است اضافه شود. هنگام تعریف اشاره گر نوع داده ای که به آن اشاره می کند باید مشخص شود. اعلان اشاره گر از فرم کلی زیر تبعیت می کند:

```
typename *ptrname;
```

typename نوع داده است که اشاره گر به آن اشاره می کند. علامت (*) عملگری است که بیان می کند متغیر اشاره گری به داده ای از نوع typename است. اشاره گر می تواند همراه با متغیر های غیراشاره گری هم اعلان شود.

مثال. متغیر ptr اشاره گری به یک داده صحیح است.

```
int *ptr;
int* ptr;
```

مثال. در اعلان زیر age از نوع صحیح و ptr اشاره گری به داده صحیح است.

```
int *ptr, age;
```

بعد از اعلان متغیر اشاره گر باید آنرا به جائی اشاره داد یعنی آدرس مکانی از حافظه که حاوی داده مورد نظر است را به اشاره گر اختصاص داد. اگر یک اشاره گر فقط اعلان شود و مقداردهی نشود ممکن است به محل دلخواهی از حافظه اشاره کند و استفاده از آن بدون توجه به این مسئله می تواند مشکلات مهمی روی سیستم تولید کند. یک تکنیک کلی مقداردهی اشاره گر با مقدار NULL یا صفر است.

راه دیگر برای مقداردهی اشاره گر ذخیره آدرس متغیر دیگر در آن است. وقتی برنامه ای اجرا می شود کلیه اجزای آن در حافظه قرار می گیرد. بنابراین هر جز از برنامه از جمله متغیرها دارای آدرس هستند. عملگر & (address-of operator) آدرس این اجزا را می دهد. عملگر & آدرس عملوند خود را برمی گرداند که می توان این آدرس را درون یک متغیر اشاره گر ذخیره کرد.

مقداردهی یک اشاره گر می تواند فرم کلی زیر را داشته باشد:

```
pointer = &variable;
```

مثال. دستور زیر آدرس متغیر age را به اشاره گر ptr اختصاص می دهد.

```
ptr = &age;
```

مثال. در برنامه زیر اشاره گر j آدرس متغیر i را نشان می دهد.

```
#include <iostream.h>

int main(){
   int i;
   int* j;
   j = &i;

   i = 4;
   cout << "i is " << i;
   cout << "\n j is " << j << "\n";
   return 0;
}
```

اشاره گرها نوعدار هستند یعنی باید به کامپایلر بگوئید که نوع متغیری که اشاره گر به آن اشاره می کند چیست. نوع اشاره گر و متغیری که به آن اشاره می کند باید یکسان باشد.

مثال.. دستورات زیر خطا تولید می کند چون نمی توان اشاره گر int را به نوع char اشاره داد.

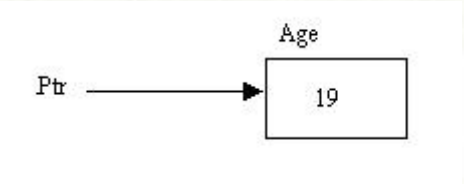
```
char c='0';
int *p=&c;
```

بعد از مقداردهی از طریق متغیر اشاره گر می توان با داده کار کرد. برای دسترسی به محلی که اشاره گر اشاره می کند از عملگر * استفاده می شود. عملگر* محتوای آدرسی از حافظه را برمی گرداند و عملگر مرجع (indirection operator) نامیده می شود چون در واقع یک ارجاع به آدرسی در حافظه است.

مثال. متغیر اشاره گر ptr به داده صحیح اشاره می کند. توجه کنید چگونه از typedef برای نامگذاری نوع اشاره گر استفاده شده است.

```
typedef *int IntPtr;
IntPtr ptr;
int pge;

age =19;
ptr = &age;     // get address of the AgeOfMary variable
cout << "ptr points to " << *ptr << endl;
cout << "age is " << age << endl;
```



برای اشاره گر ptr که به متغیر age اشاره کند موارد زیر صدق می کند:

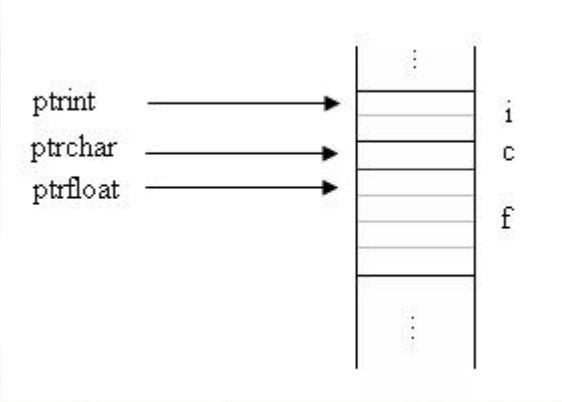
- ptr* و age هردو به محتوای متغیر age ارجاع می کنند.
- ptr و age & هر دو به آدرس متغیر age هستند.

عملگر * هم به عنوان عملگر مرجع و هم عملگر ضرب استفاده می شود. کامپایلر از نحوه به کار بردن عملگر * در دستور تشخیص می دهد منظور کدام است. وقتی عملگر * بعد از اسم اشاره گری می آید منظور محلی که است که اشاره گر به آن دارد اشاره می کند.

هر بایت حافظه دارای آدرس جداگانه ای است. بنابراین متغیرهائی مانند نوع int، float و ... که چندبایت در حافظه مصرف می کنند چند آدرس را اشغال می کنند. آدرس یک متغیر در حقیقت آدرس اولین بایت آن است.

مثال. آدرس اولین بایت متغیر به اشاره گر اختصاص داده می شود بنابراین ptrint برابر با 1000، ptrchar برابر با 1002 و ptrfloat برابر با آدرس 1003 می شود.

```
int i = 12252;
char c = 90;
float f = 1200.156004;
int *ptrint;
char *ptrchar;
float *ptrfloat;
...
ptrint = &i;
ptrchar = &c;
ptrfloat = &f;
```



حافظه پویا

برای درک بهتر اشاره گرها نیاز است درباره نحوه ذخیره اطلاعات درحافظه بیشتر بدانید. حافظه RAM شامل مجموعه ای از محل های ذخیره سازی است که هر محل توسط یک آدرس منحصر بفرد مشخص می شود. وقتی برنامه ای را اجرا می کنید کد برنامه (دستورالعمل های زبان ماشین برنامه) و داده هائی که برنامه با آنها کار می کند در حافظه قرار می گیرند. اگر متغیری در برنامه تعریف شده باشد کامپایلر محلی از حافظه را برای آن کنار گذاشته آدرس آنرا به اسم متغیر مربوط می کند. بنابراین وقتی در برنامه از اسم متغیر استفاده می شود اتوماتیک به آدرس موردنظر مراجعه می شود.

اشاره گرها وقتی نقش ایفا می کنند که بخواهیم از حافظه پویا (یا heap) استفاده کنیم. ناحیه Heap فضای آزاد حافظه است که در اختیار هیچ برنامه ای نیست و می تواند به صورت پویا استفاده می شود، یعنی در حین اجرای برنامه در صورت نیاز اختصاص داده می شود و هنگامی که دیگر به آن احتیاج نباشد آزاد می شود .

دستورات new و delete برای تخصیص و بازپس گیری حافظه هنگام کار با حافظه پویا استفاده می شوند .

دستور new فضائی از حافظه آزاد را به برنامه اختصاص می دهد. میزان فضای موردنیاز توسط نوع داده ای که بعد از دستور ذکر می شود تعیین می شود. اگر دستور موفق باشد اشاره گری به فضای گرفته شده را برمیگرداند. اگر شکست بخورد مقدار NULL را برمیگرداند.

بعد از اینکه کار با حافظه پویا باید حافظه گرفته شده توسط دستور delete پس داده شود. به این ترتیب حافظه می تواند مجدد توسط برنامه های دیگر استفاده شود.

مثال.

```
IntPtr = new int;
if (IntPtr == NULL)
   {
   cerr << "Could not allocate sufficient space" << endl;
   exit(1);
   }
*IntPtr = 55;
delete IntPtr;
IntPtr = 0;
```

بعد از delete یک اشاره گر آنرا با مقدار 0 پر کنید تا تصادفا آن را دوباره استفاده نکنید.

اگر فضا های گرفته شده از حافظه آزاد پس داده نشود و برنامه خاتمه پیدا کند برنامه اصطلاحا دارای مشکل memory leak است. میزان حافظه موجود بعد از اجرای برنامه کمتر از قبل از اجرای آن است. مگراینکه کامپیوتر راه اندازی مجدد شود. بنابراین سعی کنید همیشه حافظه های گرفته شده را آزاد کنید.

دستورات new و delete دو حالت دارند؛ برای آرایه ها ان دو دستور به صورت زیر استفاده می شوند.

```
new[]() ;
delete[]() ;
```

مثال. برنامه زیر از حافظه پویا استفاده می کند. به آرایه a به اندازه ۱۰ عدد صحیح حافظه اختصاص داد ه می شود. به b عددی صحیحی اختصاص می دهد با مقدار اولیه ۸۹.

```
#include <iostream.h>

int main() {

   int * a= new int[10];
   int * b=new int(89) ;

   cout << "*b=" << *b << endl;
    *(a+5)=9;
   cout << "a[5]=" << *(a+5) << endl;

   delete [] a;
   delete b;

   return 0;
}
```

نکته. برای هر دستور new یک دستور delete باید باشد.
نکته. بررسی کنید دستور new خطا یا NULL برنگردانده باشد.
نکته. برای آرایه ها [] در دستورات new و delete لازم است.
نکته. توابع malloc() و free() معادل دستورات new و delete هستند.

محاسبات روی اشاره گر

عملیات جمع، تفریق، افزایش و کاهش را می توان روی متغیرهای اشاره گر انجام داد. چون اشاره گر آدرسی در حافظه است وقتی محاسباتی روی آن انجام می گیرد رفتار متفاوتی نشان می دهد. وقتی عمل جمع عددی با متغیر اشاره گر صورت می گیرد اشاره گر به اندازه حاصلضرب عدد در تعداد بایت های نوع داده ای که اشاره می کند جلو می رود. همین برای عمل تفریق هم صدق می کند. اگر مقداری از متغیر اشاره گر کم شود در محاسبات تعداد بایت های نوع داده محسوب می شود.

عملگر افزایش (++) مقدار متغیر را یکی اضافه می کند در حالیکه متغیر اشاره گر را به تعداد بایت های نوع داده آن حرکت می دهد. اگر یک اشاره گر به عدد float دارید چون نوع float چهار بایت دارد با افزایش اشاره گر ۴ واحد به آن اضافه می شود. بنابراین به ۴ بایت بعدی حافظه اشاره می کند و دیگر به همان ۴ بایت قبلی اشاره نمی کند.

مثال. اگر int را چهاربایت درنظر بگیریم، اشاره گر p هشت بایت به جلو حرکت می کند.

```
int a;
int p;
p=&a;
p=p+2;
```

عمل دیگری که روی اشاره گر ها انجام می شو تفاضل است. می توان مقدار دو اشاره گر را از هم کم کرد و فاصله بین آنها را بدست آورد.

مثال. اگر ptr1 و ptr2 هر دو اشاره گر باشند عبارت زیر اختلاف فاصله آنها را می دهد.

```
ptr1 - ptr2
```

نکته. اگر اشاره گر به محل ناشناخته ای از حافظه حرکت کند و عملی روی آن انجام دهید ممکن است عملیات برنامه های دیگر کامپیوتر را با اشکال مواجه کنید.

مقایسه دو متغیر اشاره گر با هم تنها زمانی معتبر است که هردو به یک نوع داده اشاره کنند.

مثال. عبارت زیر زمانی درست است که اشاره گر ptr1 به آدرسی قبل از ptr2 اشاره کند.

```
ptr1 < ptr2
```

توجه کنید که عملیات ضرب و تقسیم روی یک اشاره گر انجام نمی گیرد و باعث بروز خطای کامپایلر می شود.

اشاره گر و آرایه

ارتباط خاصی بین اشاره گرها و آرایه ها در C++ وجود دارد. از اشاره گرمی توان برای پیمایش آرایه ها استفاده کرد. در حقیقت اسم یک آرایه بدون هیچ اندیسی اشاره گری به اولین خانه آن است. اگر آرایه ای به نام array [] تعریف کرده باشید array به اولین خانه آرایه اشاره می کند. بنابراین می توان به صورت غیر مستقیم توسط عملگر * به عناصر آن دسترسی پیدا کرد. یعنی *array اولین خانه آرایه است و *(array+1) خانه دوم و به همین ترتیب الی آخر.

```
*(array) == array[0]
*(array + 1) == array[1]
*(array + 2) == array[2]
...
*(array + n) == array[n]
```

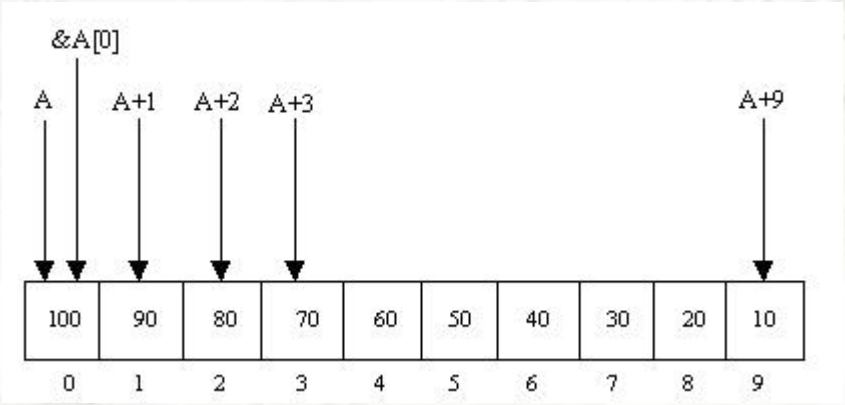
به عملگر & برای بدست آوردن آدرس آرایه نیاز نیست البته می توان توسط &array[0] هم آدرس اولین عنصر آرایه را بدست آورد یعنی array == &array[0].

مثال. برنامه زیر عناصر آرایه A را توسط اشاره گر نمایش می دهد.

```
#define MAX 10
int A[MAX] = {100, 90, 80, 70, 60, 50, 40, 30, 20, 10};

for (int i=0 ;  i<10 ; i++)
   cout << *(A+i) << endl;
```

شکل زیر ارتباط آرایه و آدرس های را نشان می دهد.



نکته. استفاده از اشاره گر برای آرایه روش سریع تری نبست نوشتن به اندیس آرایه است.
نکته. هنگام کار کردن با آرایه توسط اشاره گر کامپایلر شروع و پایان آرایه را چک نمی کند بنابراین خودتان باید مواظب باشید از محدوده عناصر آرایه عبور نکنید.
نکته. بخاطر داشته باشید اسم آرایه یک ثابت اشاره گری است و نمی تواند تغییر کند و در طی اجرای برنامه ثابت می ماند.

اشاره گر و رشته

رشته یک آرایه کاراکتری است که به کاراکتر null ختم می شود. مانند آرایه نام رشته اشاره گری به اولین کاراکتر آن است بنابراین برای کار با رشته ها یک اشاره گر به کاراکتر بکار می آید.

مثال. متغیر Msg1 اشاره گری به کاراکتر است که با یک ثابت رشته ای مقداردهی اولیه شده است. Msg1 به اولین کاراکتر این رشته اشاره می کند.

```
char *Msg1 = "This is a message";
```

راه دیگر برای استفاده اشاره گر برای رشته ها اختصاص فضای پویا به متغیر اشاره گر است.

مثال. Msg2 متغیر اشاره گری است که در حافظه پویا ایجاد شده است.

```
Msg2 = new char[16];
if (Msg2 == NULL) {
   cerr << "Could not allocate sufficient space" << endl;
   exit(1);
   }

strcpy(Msg2, "A new message");
cout << Msg2 << endl;
delete [] Msg2;
```

اشاره گر و ساختمان

مشابه هر نوع داده دیگری می توان اشاره گری به ساختمان در برنامه اعلان کرد. اشاره گر به ساختمان معمولا برای ارسال ساختمان به تابع استفاده می شود. علاوه براین برای پیاده سازی ساختمان داده مهم لیست پیوندی هم بکار می رود.

برای دسترسی به عناصر ساختمان از طریق اشاره گر باید از عملگر-> (indirect membership operator) استفاده شود.

مثال. استفاده از اشاره گر برای دسترسی به ساختمان

```
#include <iostream.h>

typedef  struct account {
   float balance;
}
account *ptraccout;

int main() {
   ptraccount = new account;
   ptraccount->balance=2000;
   cout << ptraccount->balance;
   delete ptraccount;
   return 0;
}
```

راه دیگر برای دسترسی به اجزای ساختمان توسط اشاره گر استفاده از عملگر مرجع است. اشاره گر به همراه علامت * باید درون پرانتز قرار گیرند زیرا عملگر (.) الویت بیشتری نسبت به (*) دارد.

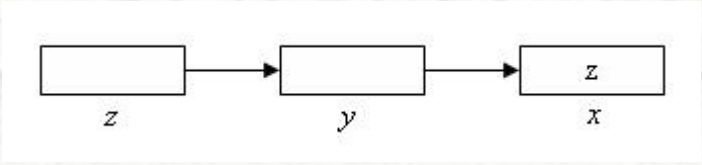
```
(*ptraccount).balance = 2000;
```

اشاره گر به اشاره گر

C++ اجازه می دهد که اشاره گری به اشاره گر دیگر داشته باشید. چون یک اشاره گر یک روش غیر مستقیم دسترسی به یک متغیر است به همین دلیل اشاره گر به اشاره گر غیر مستقیم چندگانه (multiple indirection) نامیده می شود. برای تولید اشاره گر به اشاره گر یک ستاره برای هر لایه از ارجاع اضافه می شود. بندرت اتفاق می افتد که اشاره گر به اشاره گر را در برنامه ای استفاده شود.

مثال.

```
char x;
char *y;
char **z;
x='z';
y=&x;
z=&y;
```



اشاره گر به تابع

وقتی تابع کامپایل شده و در حافظه برای اجرا قرار می گیرد قسمتی از حافظه را اشغال می کند بنابراین دارای آدرس است که می توان آن را به اشاره گری اختصاص داد. اشاره گر به تابع راهی برای فراخوانی تابع به صورت غیرمستقیم است علاوه بر این برای ارسال تابع به عنوان پارامتر به تابع دیگر هم بکار می آید.

اعلان و استفاده اشاره گر به تابع درنظر اول کمی متفاوت است ولی از همان قاعده تبعیت می کند. برای اعلان اشاره گر به تابع نام تابع درون پرانتز قرار می گیرد و قبل از آن علامت * قرار می گیرد. مشابه زیر:

```
datatype (*pointerToFunction) (pElement) = NULL ;
```

*pointerToFunxtion اشاره گر به تابع است که درون پرانتز باید باشد. سمت چپ آن نوع برگشتی و سمت راست آن پارامترهای تابع قرار می گیرند. سپس با NULL مقداردهی می شود.

مثال. funcPtr اشاره گر به تابعی است که هیچ آرگومان و مقدار برگشتی ندارد.

```
void (*funcPtr)() = NULL ;
```

مشابه آرایه ها، برای بدست آوردن آدرس تابع نیازی به عملگر آدرس & نیست. فراخوانی تابع توسط اشاره گر مشابه فراخوانی عادی تابع است.

مثال. متغیر p اشاره گری به تابع square است. تابع به دو طریق فراخوانی شده است که خروجی هر دو فراخوانی یکسان است.

```
#include <iostream.h>
double square(double x);        // The function prototype.
double (*p)(double x) = NULL;   // The pointer declaration.

main() {
    p = square;                 // Initialize p to point to square().
    Cout <<  square(6) << p(6));
    return(0);
}
double square(double x) {
    return x * x;
}
```

اشاره گر و پارامتر مرجع

همانطور که قبلا در بخش تابع گفته شد آرگومان به دو صورت مقداری و مرجع می تواند به تابع ارسال شود. در حالت مرجع آدرس آرگومان به تابع داده مي شود اين كار بسادگي با اضافه كردن علامت & (عملگر ادرس) قبل از پارامتر در خط اعلان تابع انجام مي شود.

آرگومان های خط فرمان

هنگامی که یک برنامه از خط فرمان سیستم عامل فراخوانی می شود آرگومان هائی را می توان به تابع main ارسال کرد. پارامترهای تابع main به شکل زیر هستند.

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

argv همیشه آرایه ای رشته ای است که شامل دستوری است که در خط فرمان وارد می شود. فضای خالی، اجزای فرمان را از هم جدا و تبدیل به آرگومان های جداگانه در آرایه می کند. argc تعداد عناصر درون آرایه پارامتر دوم است. argv[0] شامل مسیر و نام خود برنامه است.

مثال. برنامه زیر کلیه آرگومان های خط فرمان را نمایش می دهد.

```
//CommandLineArgs.cpp
#include <iostream.h>

int main(int argc, char* argv[]) {
  cout << "argc = " << argc << endl;
  for(int i = 0; i < argc; i++)
    cout << "argv[" << i << "] = "
         << argv[i] << endl;
}
```

اسامی argv و argc برای آرگومان های خط فرمامن الزامی نیست و می توان از شناسه های دیگر استفاده کرد ولی این دو اسم متعارف هستند و استفاده از اسامی دیگر باعث گیج شدن افراد دیگر می شود.