گلنگور

کاردانی به کارشناسی

برنامه نویسی به زبان C

دکتر کریم پور

عضو هیئت علمی دانشگاه تبریز

آدرس:

دانشگاه تبریز- دانشکده علوم ریاضی- گروه علوم کامپیوتر- دکتر کریم پور

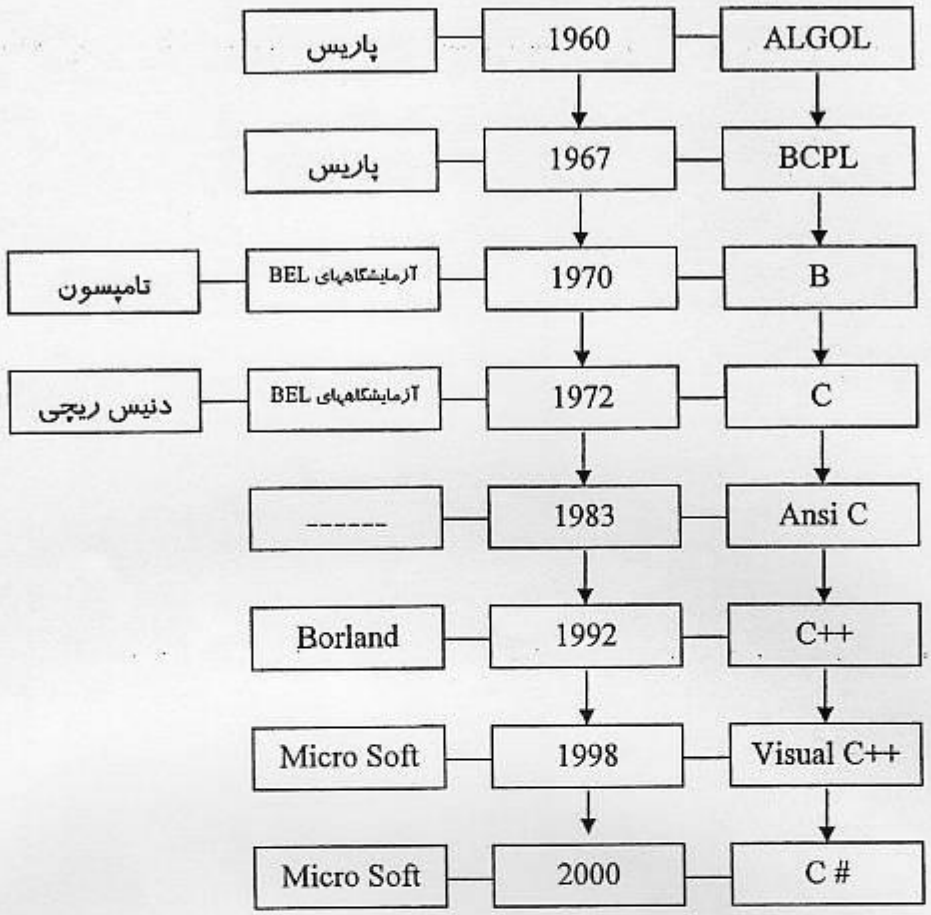
تلفن:

موبایل: ۰۹۱۴۱۰۱۰۰۷۴ محل کار: ۳۳۹۲۲۲۲۳

آدرس پست الکترونیک:

Jaber_Karimpour@Yahoo.Com
Karimpour@Tabrizu.ac.ir

تاریخچه زبان C :



قابلیت های زبان برنامه نویسی C :

۱- قابلیت حمل (Portable)

نکته : قابلیت حمل بر روی سورس برنامه ها مانند Example.C و Example.CPP تعریف گردیده است .

۲- زبانی کامپایلری است : فایلهای اجرائی تهیه شده توسط زبان C بدون نیاز به کامپایلر قابل اجرا میباشند .

Source→Compiler→Executable Files

۳- زبانی همه منظوره است : از این زبان میتوان برای نوشتن برنامه های سیستمی ، تجاری ، هوش مصنوعی ، پردازش تصویر و ... استفاده نمود.

٤- زبانی میانی است .

۵- تعداد کلمات کلیدی در زبان C کم میباشد (۳۲ کلمه) در حالی که در زبانی مانند Basic حدود ۱۵۰ کلمه کلیدی وجود دارد .

۶- زبان C نسبت به حروف کوچک و بزرگ حساس میباشد. (Case Sensitive)

۷- زبان C مبتنی بر توابع میباشد که هر تابع تکه ای از برنامه میباشد .

توابع تشکیل دهنده برنامه های C :

۱- توابعی که قبلا در فایلهای عنوان (Header File) تعریف شده و قابل استفاده میباشد .
۲- کلمات کلیدی
۳- توابعی که توسط برنامه نویس با استفاده از بند ۱و ۲ تعریف میشود .

ساختار برنامه های C :

```
# include <stdio.h>
# include <conio.h>
Void main ( )
{
تعریف متغیرهای محلی
---
---        بدنه تابع main
---
}
```

نکته :

هدر فایل هایی که داخل < > قرار دارند جستجو ابتدا در مسیر پیش فرض هدر فایل ها سپس در مسیر جاری انجام خواهد یافت .

هدر فایل هایی که داخل " " قرار دارند جستجو ابتدا در مسیر جاری سپس در مسیر پیش فرض هدر فایل ها انجام خواهد یافت .

نکته :

در صورتی که برنامه نوشته شده به زبان C بدون خطا کامپایل گردد ، ابتدا فایلی با پسوند OBJ سپس فایل EXE تولید خواهد شد در غیر اینصورت خطا اعلام خواهد شد .

انواع خطاها :

۱- Compile ERROR یا Syntax ERROR

این نوع خطاها هنگامی صادر میگردد که برنامه نویس دستوری را اشتباه نوشته باشد .

مثال :

1-

```
Int x,y ;
```

کلمات کلیدی با حروف کوچک نوشته میشوند

2-

```
void main ( )
{
int a;
a=25;
}
```

بعد از تابع main از ; استفاده نمیشود .

3-

```
int for ;
```

استفاده از کلمات کلیدی به عنوان نام متغیر غیر مجاز است

۲- خطا های زمان اجرا (Runtime ERROR) :

مثال :

```
int x,y ;
scanf("%d",x);
y = 125/(x-5);
```

اگر از صفحه کلید عدد ۵ وارد شود عدد تفسیم بر صفر خواهد شد و خطای زمان اجرا صادر میشود

نتایج Runtime ERROR : ۱- هنگ کردن سیستم ۲- خروج از برنامه و ...

۳- خطاهای معنایی Semantic ERROR :

در این نوع خطاها برنامه از نظر کامپیوتر به درستی اجرا میشود ولی کاربر متوجه میشود که نتیجه برنامه صحیح نمی باشد .

مثال : جمع واحد/(نمرات درسی * تعداد واحد) جمع= معدل

در صورتی که برنامه نویس بجای تقسیم از ضرب استفاده نماید برنامه از نظر کامپیوتر درست اجرا خواهد شد ولی نتیجه درست نخواهد بود .

سوالات تستی

۱) برنامه زیر چه نوع خطایی دارد ؟

```
int x , a , y ;
y = x/ (a-a) ;
```

الف) Syntax Error ب) Semantic Error ☑ ج) Runtime Error د) عدد صحیح است

۲) کدام تعریف برای فایل عنوان صحیح است ؟

الف) همان فایل کد است ب) فایل کامپایل شده OBJ است

ج) فایل اجرائی EXE است ☑ د) فایل متنی است که در آن توابع آماده وجود دارد

۳) کدام نوع از فایلهای C قابلیت Portable دارد ؟

الف) *.CPP ب)*.C ج) *.OBJ ☑ د) موارد الف و ب

نکته :

- در زبان C نوشتن تابع main لازم است و بدون تابع main نمی شود برنامه C نوشت .
- توابع دیگر را میتوان قبل از main یا بعد از آن نوشت .
- با علامت // میتوان توضیحات تک خطی را اضافه نمود .
- بین علامت /* */ میتوان توضیحات چند خطی را اضافه نمود .
- فرق بین C و C++ این است که C یک زبان ساخت یافته است ، منظور این است که زیر برنامه های C تحت عنوان توابع از هم جدا شده اند و از دستوراتی که ساخت یافتگی را به هم میزند در C کمتر استفاده میشود . اما در مقابل C++ علاوه بر ساخت یافتگی شیئ گرا میباشد و مفهوم شیئ مزایایی مانند ارث بری ، تعریف مجدد عملگرها و کپسوله سازی را بدنبال دارد .

انواع داده ها در زبان C :

۱- Char	کاراکتر	۱ بایت
۲- Int	عددی صحیح	۲ بایت
۳- Float	عددی اعشاری	٤ بایت
٤- Double	عددی اعشاری بزرگ	۸ بایت
٥- Void	برای فایلها کاربرد دارد	

نوع داده char :

```
Char ch ;
```

ch

هنگام ذخیره سازی کد اسکی حرف A که مساوی عدد ۶۵ است ذخیره خواهد شد .

```
Char ch='A' ;
```

ch

0	1	0	0	0	0	0	1

```
'A' +32 = 'a' ;
```

نکته :

توجه شود که "A" معادل کد اسکی 'A' و'\0' است و اگر در طول برنامه ch بصورت "char ch="A تعریف گردد در برنامه خطای کامپایل رخ خواهد داد .

نکته :

"A" دو بایت فضا اشغال می کند در حالی که 'A' تنها یک بایت فضا اشغال می کند.

نکته :

موارد زیر صحیح میباشد :

```
char ch=70 ;
char ch='B' - 1 ;
char ch='B' + 'A' ;
```

نحوه ذخیره سازی کاراکتر ها در حافظه :

الف) بدون علامت :

کوچکترین عدد ۰	0	0	0	0	0	0	0	0
بزرگترین عدد ۲۵۵	1	1	1	1	1	1	1	1

ب) با علامت :

در این حالت یک بیت بعنوان بیت علامت در نظر گرفته می شود .

0	1	1	1	1	1	1	1

\+ 127

1	1	1	1	1	1	1	1

\- 127

پس بازه تعریف شده [۱۲۷... ۱۲۷-] میباشد که یک بایت علامت دار در آن قابل ذخیره است . ولی در کتابها این بازه [۱۲۷... ۱۲۸-] تعریف شده است ولی علت چیست ؟

0	0	0	0	0	0	0	0	+0
1	0	0	0	0	0	0	0	-0

در علم ریاضی برای عدد صفر علامت تعریف نشده یعنی 0+ و 0- بدون معنی است به همین علت طی قراردادی در ذخیره سازی با روش علامت دار عدد 0- به 1- و عدد 1- به 2- و ... و عدد127- به 128- تبدیل می گردد . پس بازه ذخیره سازی اعداد بصورت [۱۲۷... ۱۲۸-] تعریف می گردد .

تعریف انواع داده جدید با اضافه کردن کلمات کلیدی :

char signed char long char short char	[-128 . . . +127]
unsigned char	[0 . . . 255]

سوال تستی :

۱- بازه کدام گزینه [۱۲۷... ۱۲۸-] می باشد .

الف) char ب) signed char ج) int ☑ د) موارد الف و ب

بعضی از کد های مهم اسکی :

۱- \a	7		
۲- \b	8	Back Space	
۳- \n	10	New Line	
٤- \r	13	Return	↵
٥- \0	0	NULL	

نوع داده صحیح :

int x ; short int x; signed int x ; signed short int x ;	[-32768 . . . +32767]	۲ بایت فضا اشغال خواهد شد
unsigned int x ; unsigned short int x;	[0 . . . 65536]	اعداد بدون علامت مثبت و۲ بایت فضا اشغال خواهد شد
long int x; unsigned log int x;		٤ بایت فضا اشغال خواهد

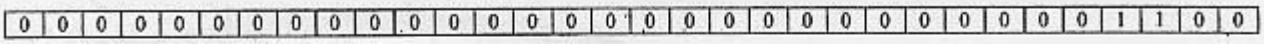
برای پیاده کردن ملغل ۲ که عدد از سمت راست شروع می کنیم وقتی به اولین
یک رسیدیم آن را می نویسیم ، اعداد ما بعد یک را برعکس می کنیم (یکها، صفرو صفرها یک)

۱۱۱۰۰۱۰۰۱۱۱۰۰۰

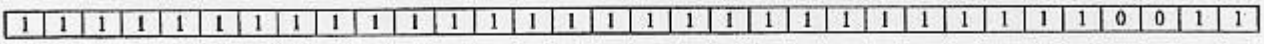
ملغل ۲: ۰۰۰۱۱۰۱۱۰۰۱۰۰۰

ذخیره اعداد منفی در RAM با استفاده از مکمل ۲ :

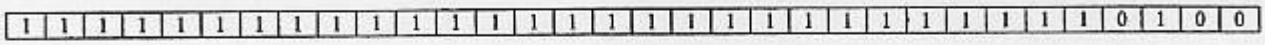
int x=12 ;



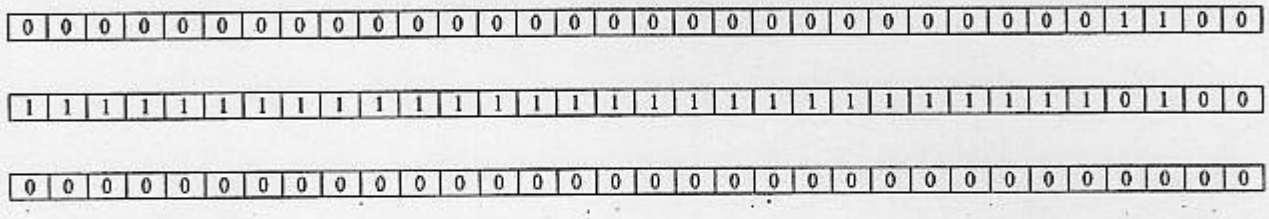
تبدیل ۰ ها به ۱ و ۱ ها به ۰



+1



در حالتی که اعداد در ۲ بایت ذخیره شوند عدد بدست آمده ۱۲- است .



حاصل در دو بایت ذخیره می شود یعنی صفر ذخیره خواهد شد . پس عدد مذکور ۱۲- است که با ۱۲+ جمع گردیده و حاصل صفر گردیده است .

سوال تستی :

۱- اگر

```
int x,y ;
x=16;
y=~x + 1 ;
```

مقدار y چیست ؟ (نقیض یک عدد بعلاوه ۱ می شود منفی آن عدد)

۲- اگر

```
int x=16,y ;
y= ~x ;
```

مقدار y چیست ؟

الف) ۱۶- ☑ ب)۱۷- ج)۱۵- د)نامعلوم

float const x = 15.25 ;

x = 15.25 می‌شود.

فرق ماکرو و تابع از نظر ماشین است.

نوع داده float اعشاری :

float x ;	٤ بایت
short float x ;	٤ بایت
long float x;	٨ بایت
double x;	٨ بایت
long double x ;	١٠ بایت

نکته :

در مورد داده float استفاده از signed و unsigned معنی ندارد و در صورت استفاده خطای کامپایل رخ خواهد داد .

متغییر :

مکانی از حافظه که دارای نام بوده و مقدار آن در طول برنامه میتواند عوض گردد متغییر نامیده می شود .

قوانین نام گذاری متغییر ها :

١- نام متغییر میتواند a..z و A..Z و # و _ باشد .
٢- شروع نام متغییر نمی تواند عدد باشد .
٣- در نام متغییر نباید از فاصله استفاده نمود .
٤- استفاده از کلمات رزرو یا کلیدی غیر مجاز میباشد .

برای مثال موارد زیر در نامگذاری متغییر ها غیر مجاز است .

١- int 2x ٢- int for ٣- int c$

موارد زیر نیز صحیح است :

int write,until

int pascal; ✗ int A B; ✗

int -ca; ✗
int a$m; ✗
int Int; ✓

ثابت ها :

مکانی از حافظه است که مقدار آن در طول اجرای برنامه عوض نمی شود .

const int x=100;

نکته :

const بطور پیش فرض نوع ثابت را int در نظر می گیرد.

```
const x=15;
const x=15.25;      مقدار x مساوی ١٥ خواهد بود
```

با علامت #define نیز میتوان ثابت و ماکرو تعریف کرد .

منظور از ماکرو تکه برنامه ای است که یک بار نوشته می شود و چندین بار استفاده می شود .

```
# define m(a,b)  2*(a+b)
void main ( )
{
printf("%d" , m(8,4)) ;
printf("\n %d" , m(10,5)) ;
}
```

→ ٢×(٨+٤) → ٢٤
→ ٢×(١٠+٥) → ٣٠

خروجی برنامه :
24
30

سوال تستی :

۱- تعریف زیر چند بایت اشغال می کند ؟

```
unsigned char x,y ;
long int z ;
long double t,p ;
```

تعریف فوق ۲۶ بایت اشغال می کند .

۲- کدام نام برای متغیرها در زبان C مجاز می باشد ؟

الف) 2ali ب) s!za ج) _Sum د) M$k

عملیات جایگذاری:

```
int x ;
x=12 ;
x=x+3 ;
```

در زبان C عملیات جایگذاری و مقایسه با یکدیگر متفاوت میباشند . (= و ==)

نکته :

اگر نوع اعشاری را در نوع صحیح قرار دهیم قسمت اعشاری حذف میشود .

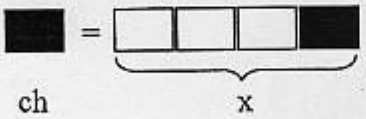
```
int i ;
float f=5.6 ;
i = f ;      → i = 5
i = -3.4 ;   → i = -3
i = 3.4 ;    → i = 3
```

در حالت کلی اگر متغیری با فضای کوچک را در متغیری با فضای بزرگ قرار دهیم هیچ مشکلی بوجود نمی آید .

float = int یا double = float

در حالتی که متغیری با فضای بزرگ در متغیری با فضای کوچک قرار دهیم ممکن است قسمتی از اطلاعات حذف گردد .

```
float x ;
char ch ;
ch = x ;
```



مثال ۱ :

```
char c='A' ;
int i = 52 ;
float f ;
f = i ;      → f = 52.0
i = c ;      → I = 65
f = c ;      → f = 65.0
```

مثال ۲ :

با فرض اینکه :

```
unsigned char c ;
c = -12 ;
```

محتوای C چیست ؟ الف) ۱۲- ب) خطای کامپایل میدهد ج) ۱۲+ د) ۲۴۴

کدام یک از شرط های زیر درست است؟

1) if (17%5) ✓
2) if (0)

به ظاهر به نظر میرسد که محتوای C باید عدد ۱۲- باشد در حالتی که C را بدون علامت تعریف کرده ایم بنابر این ۱۲- در C قرار نمی گیرد. $(12)_{10} = (00001100)_2$

حال ۱۲- را با روش مکمل ۲ بدست می آوریم : برای مکمل ۲ ، از سمت راست عدد تا اولین ۱ خود ۰ و ۱ ها را خواهیم نوشت در بقیه اعداد ۰ به یک و ۱ به صفر تبدیل میشود .

0	0	0	0	1	1	0	0

1	1	1	1	0	1	0	0

پس محتوای C عدد ۲٤٤ خواهد بود .

عملگرها در C و C++ :

۱- عملگرهای محاسباتی : * , / , + , - , %

۲- عملگرهای رابطه ای : == , != , >= , > , < , <=

۳- عملگرهای منطقی : ! , || , &&

٤- عملگر های بیتی : & , | , ~ , xor , >> , <<

۵- عملگرهای یکتایی : → , ++ , -- , sizeof , ? , * , & , , , ( )

۶- عملگرهای ترکیبی : += , *= , %=

عملگرهای محاسباتی :

```
int x=5 , y=17 , z ;
z = y / x ;   → z = 3
z = x / y ;   → z = 0
z = y % x ;   → z = 2
z = x % y ;   → z = 5
```

فرمول :

$$Y \% X = Y - [Y / X] * X$$

(/) حاصل این تقسیم همواره عددی صحیح است ، برای اینکه حاصل این عملگر اعشاری گردد باید به روش زیر از این عملگر استفاده نمود :

```
int x , y ;
float a ;
a= (float) x / y ;
```

اولویت عملگرها :

* , / , %
+ , -

عملگرهای رابطه ای :

در C و C++ هر عدد غیر صفر صحیح است و عدد صفر نادرست است .

```
int a=5 ;
if( a==5);      درست
if(5);          درست
if(0);          نادرست
```

حاصل عبارت زیر چیست؟

$5 > 2 \;!= 12 < 10$

1 ≠ 0

چون با هم مساوی نیستند: ①

به عبارتهایی که در آنها از عملگرهای رابطه ای استفاده میشود ، عبارتهای رابطه ای گویند و حاصل عبارتهای رابطه ای صفر یا یک میباشد .

اولویت عملگرها :

<, >, >=, <=
==, !=

نکته :

اولویت عملگرهای ریاضی بیشتر از عملگرهای رابطه ای است .

مثال ۱ : اگر داشته باشیم :

```
int a , b = 5 , c = 10 ;
a=b<3 == c>b ;
```

مقدار a چیست ؟

$$a = \underbrace{b < 3}_{0} == \underbrace{c > b}_{1}$$

$$\underbrace{0 \quad 1}_{0}$$

مثال ۲ : اگر داشته باشیم :

$a = 5 > 7 == 8 > 10 != 10 < 20$ و

مقدار a چیست ؟

$$a = \underbrace{5 > 7}_{0} == \underbrace{8 > 10}_{0} != \underbrace{10 < 20}_{1}$$

$$\underbrace{\underbrace{0 \quad 0}_{1} \quad 1}_{0}$$

عملگرهای منطقی :

p	q	P && q	P || q	! p
T	T	T	T	F
T	F	F	T	F
F	T	F	T	T
F	F	F	T	T

حاصل عبارات منطقی T و F میباشد که T معادل ۱ و F معادل صفر میباشد .

اولویت عملگرها :

!
عملگرهای ریاضی
عملگرهای رابطه ای
&&
||

```
a = !5 || !(-2) && !0+1;
a = 0 || 0 && (1-1);
a = 0 || 0 && 0
```

a = 0

مثال ۱ :اگر داشته باشیم :

```
int a = 5 , b = -1 , c ;
c = a && b ;
```

مقدار c چیست ؟ مقدار c ، True و معادل ۱ است .

مثال ۲ : اگر داشته باشیم :

```
a = 5 || -2 && 0
```

مقدار a چیست ؟

```
a = 5 || -2 && 0
    T    F
    T
```

عملگر های بیتی :

این نوع عملگرها همان نوع کوچک شده عملگرهای منطقی هستند که بر روی تک به تک بیت ها عمل میکنند نه بر روی ارزش کل متغییر . پس برای عمل روی بیت ها عملوند این عملگرها به مبنای ۲ تبدیل میشود .

نکته :

Xor یای انحصاری است و حاصل در صورتی True خواهد بود که تنها یکی از عملوند ها یک باشد

p	q	p^q
1	1	0
1	0	1
0	1	1
0	0	0

مثال ۱ :

```
int x=5, y=10,z ;
z = x & y ;     z=0
z = x | y ;     z=15
z = ~ x ;       z=-6
```

x 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1

y 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0

چون x+1~ مساوی x- است پس x~ مساوی x-1- خواهد بود .

مثال ۲ :

```
int x=50 ;
x= x>>3 ;
```

x 0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 0 — 50

x 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 — x>>1 — 25

x 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 — x>>1 — 12

x 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 — x>>1 — 6

x = x << n	به عبارت دیگر در هر بار شیفت به چپ عدد در ۲ ضرب میشود .	به عبارت دیگر در هر بار شیفت به راست عدد بر ۲ تقسیم میشود .	x = x >> n
$x = x*2^n$			$x = x/2^n$

حاصل عبارت زیر چیست؟

```
int x = 2 ;
y = (x++) + 2*(x+4) + 3x + 4*x++
```

غلط

$y = 5 + 2*5 + 3*5 + 4*5$

$y = 5 + 10 + 15 = 30$

توجه شود که x++ است نه ++x .

$y = 2 + 2\times2 + 3\times2 + 4\times2 = 20$

بعد از ای هر x++ یک واحد به x اضافه می شود. $x = 5$

مثال:

```
int x = 2;
x = ++x + (++x) + 2x;
```

مثال:

```
int x = 2
x = (x++) + (x++) + 2x
```

$x = 2 + 2 + 2\times2 = 8$

$x = 10$

چون ۲ تا x++ داریم ۲ عدد به مقدار x اضافه می کنیم.

مثال:

```
int x = 2;
y = (++x/3) + (++x/4) + 2*x
```

$y = 4 + 4 + 2\times4$

$y = 12 \quad x = 12$

```
int x = 3;
y = 2*x + 3*(++x) - x++ + 2*x;
printf("\n x=%d, y=%d", x, y);
```

حل: $x = 5$

$y = 2\times5 + 3\times5 - 5 + 20;$

$y = 30$

مثال ۳ : حاصل x بعد از اجرای عمل زیر کدام است ؟

```
int x = 100 ;
x >> 3 ;
```

الف) ۱۲ ب) ۲۵ ج) ۵۰ ☑ د) ۱۰۰

درست است که x را ۳ واحد شیفت می دهیم ولی چون نتیجه در x قرار نمی گیرد پس x همان ۱۰۰ خواهد بود .

عملگرهای ++ و -- :

++x ; → x=x+1

x++ ; → x=x+1 اول x را در نظر بگیر بعد

x++و++x در حالت تک دستوری فرقی با یکدیگر ندارند ولی در حالات دیگر متفاوت میباشند :

```
int x = 5 ;
++ x ;
x ++ ;
```

} در هر دو حالت x مساوی ۶ خواهد

```
int x = 5 ;
y = x ++ ;
```

→ y=5 , x=6

```
int x = 5 ;
y = ++ x ;
```

→ y=6 , x=6

اولویت عملگرها :

++x	یکتایی
~ , !	منطقی ، بیتی
* , / , %	محاسباتی
+ , -	محاسباتی
<< , >>	بیتی
> , < , >= , <=	رابطه ای
== , !=	رابطه ای
&	بیتی
^	بیتی
|	بیتی
&&	منطقی
||	منطقی
x++	یکتایی

نکته :

اگر یک عبارت حاوی چند x++ باشد ابتدا x++ ها اجرا میشوند و مقدار x بدست می آید و در نهایت بجای x+1 قرار میگیرد .

```
int i=2 , x=5 ;
y= i*x+(++i)+10+(++i) → y=4*5+4+10+4=38
```

۳ ۴

مثال:

```
int x = 2;
y = ++x - 2*(++x) + 3*(x++) - x++;
```

y = ٤ - ٢ × ٤ + ٣ × ٤ - ٤

y = ٤

x = ٦ ← ٤ + ٢ = ٦

اول x++ ها را اجرا کنید.
به جای تمام x ها عدد ٤ را قرار می دهیم.
بعد ++x را اجرا می کنیم. ٢ تا داریم.
چون آخرین مقدار x ها ٤ شد.

مثال:

```
int x = 5;
y = ++x + ++x + ++x + ++x;
```

y = ٧ + ٨ + ٩ + ٩ = ٣٣

x = ٩

مثال: حاصل a و b بعد از اجرای این دستور چیست؟

```
b = (a+=6, a++, ++a, a++);
```

b = (a = ١١ , a = ١٢ , a = ١٣) ; b = ١٣ , a = ١٤

اولویت x++ خیلی بالا است و اولویت ++x خیلی پایین است.
= استثنا است یعنی در ، اولویت با ++x است.

```
a = 5;
y = (a++, a++, 2*a++, a+1);
```

y = ٩

٦ , ٧ , ٢×٧=١٤ / ٨ , ٩

اگر یک عبارت حاوی چند x++ باشد بجای x++ ها مقدار x قرار میگیرد و حاصل عبارت بدست می آید . اما مقدار x را باید به ازای هر x++ یک واحد اضافه گردد .

```
int i=2 , x=5 ;
y= i*x+(i++)+10+(i++) → y=2*5+2+10+2=24
i=2+2=4
```

عملگر ?

عبارت ۲ : عبارت ۱ ? (شرط) = متغییر

در صورت درستی شرط عبارت ۱ محاسبه شده و در متغییر قرار میگیرد ، در غیر اینصورت عبارت ۲ محاسبه گردیده و در متغییر قرار میگیرد .

مثال ۱ :

```
int x=5,y ;
y=(x) ? ++x : x+5 ;
```
x=6 , y=6

مثال ۲ :

```
int x=5,y ;
y=(x-5) ? ++x : x+++6 ;
```
x=6 , y=11

عملگر ,

متغییر=(عبارت ۱, عبارت ۲, ... , عبارت n);

از چپ به راست تک تک عبارتها اجرا خواهند شد و نتیجه هر عبارت در عبارات بعدی تاثیر گزار خواهد بود و در نهایت حاصل آخرین عبارت در متغییر ذخیره خواهد شد .

```
int x=5 , y ;
y = ( ++x , x=x+3 , x+4 , x-2 ) ;
      x=6   x=9
```
x=9 , y=7

نتیجه در x قرار نمیگیرد پس در عبارات بعدی دخالت ندارد

نکته :

اولویت عملگر , بعد از کلیه عملگرها میباشد یعنی حتی بعد از x++ پس اگر در عبارتی چندین x++ وجود داشته باشد بر خلاف قبل ابتدا x++ ها اجرا شده بعد عملگر , اجرا میشود .

مثال ۱ :

```
x=5 ;
y=(x=x+3 , x++ , x++ , x+5) ;
     x=7   x=8  x=9  14
```
x=9 , y=14

مثال ۲ :

```
a=8 ;
b=(a+=6 , a++)
```
a=15 , b=14

a=a+7

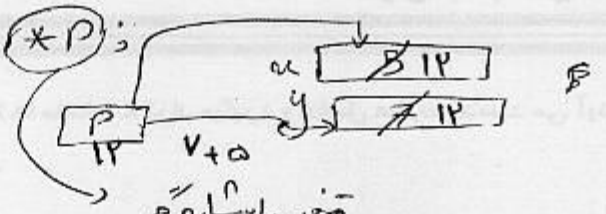
مثال ۳ :

```
a=8 ;
b=(a+=6 , a++ , a++ , a+2)
```
a=16 , b=18

مثال ٤ :

مثال: حاصل x بعد از اجرای دستورات زیر چه خواهد بود؟

```
int x=5 , y=7 , *p;
p=&y;
*p = *p + x;
x = *p;
```



x هم‌اندازه است با ۱۲.

* هر متغیر اشاره‌گر از هر نوعی باشد ۲ بایت جا می‌گیرد.

float *p;	۲ بایت
int *q;	۲ بایت
Char *r;	۲ بایت
Long Double *m;	۲ بایت
	۸ بایت

هر متغیری در RAM آدرس دارد حتی متغیر اشاره‌گر.

متغیر	محتوا
x	13.5
&x	۱۰۰۰
p	۱۰۰۰
*p	13.5
&q	200
q	200
*q	۱۰۰۰
**q	13.5

حل تست:

مقدار a و b چیست؟

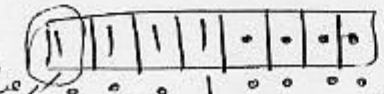
```
int a,b;
a=5;        b=5
b=a++;      a=6
```

```
int a,b;        a=6
a=5
b=++a;          b=6
```

نتیجه‌ی اجرای قطعه برنامه زیر چیست؟

```
int a=760;
char b=3;
main()
{
  b=(char)a;
  printf("%d",b);
}
```

۲۵۶ ۱۲۸ ۶۴ ۳۲ ۱۶ ۸ ۴ ۲ ۱
۰ ۱ ۱ ۱ ۱ ۰ ۰ ۰ ۰



الف) ۸- ب) ۲۴۸ ج) ۲۵۳ د) خطا دارد.

محتوای a چیست؟

```
Char a=۲۰۰;
```

حاصل 56-

۱۲۸ ۶۴ ۳۲ ۱۶ ۸ ۴ ۲ ۱
۱ ۱ ۰ ۰ ۱ ۰ ۰ ۰

a ۱ ۱ ۰ ۰ ۱ ۰ ۰ ۰ (بیت علامت)

۲۵۶ ۱۲۸ ۶۴ ۳۲ ۱۶ ۸ ۴ ۲ ۱
۰ ۱ ۱ ۱ ۰ ۰ ۰ ۰ = ۱۱۱۰۰۰ = 56-

```
a=8 ;
b=(a++ , a++ , a++) ;
```
a=11 , b=10

به عبارت دیگر آخرین عبارت استثنا میباشد .

آدرس متغییر :

حافظه RAM کامپیوتر از بایتها تشکیل شده است ، هر بایت در RAM دارای شماره ای است و هنگام تعریف متغییر بسته به اندازه آن چند بایت از حافظه RAM را اشغال میکند . به شماره اولین بایت تشکیل دهنده یک متغییر آدرس متغییر گویند . آدرس یک عدد صحیح است .

در رکوردها یا ساختمانها خواهیم دید متغییری میتواند بزرگ باشد . مثلا ۳۰۰ کیلوبایت فضا اشغال کند ولی دارای یک آدرس است .

نکته :

با وجود اینکه pointer(اشاره گر) یک عدد صحیح است ولی آن را در متغییری از نوع int نمیتوان ذخیره کرد زیرا روی pointer اعمال خاصی لازم است که در اعداد صحیح تعریف نگردیده .

مثال :

```
float x=3.15 ;
int a ;
a=&x ;
```

نکته :

آدرس متغییرها در متغییرهایی از نوع اشاره گر قرار می گیرد .

```
float x=13.5 ;
float *p ;
p=&x ;
```

متغییر اشاره گر — متغییر معمولی

1000	.	13	5	.

1000

متغییر	محتوا
x	13.5
&x	1000
p	1000
*p	13.5

- متغییر های اشاره گر در حافظه RAM تعریف شده و دو بایت فضا اشغال می کنند ولی دارای یک آدرس میباشند
- آدرس متغییرهای اشاره گر در متغییر های اشاره گر به اشاره گر قرار میگیرد .

```
float x=13.5,*p,**q ;
p=&x;
q=&p ;
```

200	→	1000	→		13	5	

متغییر اشاره گر به اشاره گر — متغییر اشاره گر — متغییر معمولی

200 — 1000

p — x

اگر متغییری به q اشاره کند بصورت (نام متغییر***) تعریف میگردد .

تمرین ۱ :

```
int x=1 , y=2 , m ,n ;
m=x && y ;  → m=1
n=x & y ;   → n= 0
```

تمرین ۲ :

```
unsigned char x=10,y ;
y=!x+1 ;  → y=1
x+= y ;   → x=11
```

unSigned char a = 200; محتوای a همان 200 می‌شود.

حاصل عبارت (-15 % 6) / (-15 / 6) چند است؟

-3 / -2 = 1

اگر می نوشت:

1.500000

(float) ((-15 % 2) / (-15 / 2))

1.000000E+00

خروجی این برنامه چیست؟

```
int a,b = 17;
a = (float) b/3 * 4;
printf("%d", a)
```

b/3 => 17/3 => 5.66

5.66 x 4 = 20.12

جواب = 20

```
int a = 5;
printf("%d %d %d %d %d", a, ++a, a--, --a, a++);
```

در printf محاسبات از سمت راست انجام می‌گیرد.

a=6	a=5	a=4	a=5	a=5
5	5	5	5	5

خروجی 5 5 5 5 5

تمرین ۳ :

```
int a=5, b=9 ,c=15 , x ;
x=a << 2 | b^7 & c >> 1 ;
x^=0
x&=0xFF
x|=0
x=~(~X)
```

حاصل عبارات زیر x است

نکته :

علامت مقسوم * علامت مقسوم علیه= علامت خارج قسمت

علامت مقسوم = علامت باقی مانده

دستورالعمل های ورودی و خروجی :

```
printf("عبارت۱" , عبارت ۲ ) ;
```

عبارت ۱ تشکیل شده است از :

۱- کاراکترهای فرمت :

%d	عدد صحیح
%i	عدد صحیح
%c	کاراکتر
%s	رشته
%p	پوینتر
%f	عدد اعشاری
%g	عدد اعشاری
%e	نماد علمی
%E	نماد علمی با E
%x	اعداد مبنای ۱۶
%X	اعداد مبنای ۱۶
%n	تعداد کاراکترهایی که قبل از این چاپ شده اند

مثال : 234.125=342125 -3E

۲- کاراکترهای کنترلی

\n	انتقال مکان نما به ابتدای سطر بعدی
\t	انتقال مکان نما به ۸ کاراکتر بعد
\r	انتقال مکان نما به ابتدای خط
\a	صدای بیپ
\"	چاپ علامت "
\\	چاپ علامت \

۳- عباراتی که عینا چاپ میشوند .

عبارت ۲ تشکیل شده است از :

۱- متغییرهایی که محتوای آنها چاپ میشود .

۲- محاسباتی که نتیجه آنها چاپ میشود .

مثال :

```
int a=5 , b=10 ;
printf("\nC is good language");
printf("\na=%d and b=%d and sum=%d",a,b,a+b);
```

خروجی :

```
C is good language
a=5 and b=10 and sum=15
```

نکاتی از printf( ) :

۱- تابع printf( ) در فایل عنوان stdio.h قرار دارد .

۲- در c++ بجای printf از شیئ cout استفاده می شود .

```
cout<<"\nC is a good language";
cout<<"a="<<a<<"and b="<<b;
cout<<"and sum="<<a+b ;
```

۳- اگر در printf چندین عبارت محاسباتی وجود داشته باشد که با , از یکدیگر جدا شده اند ، اولویت محاسبه از راست به چپ ولی عملیات چاپ از چپ به راست خواهد بود .

مثال : اگر داشته باشیم :

```
int i=5 ;
printf("\n%d%d%d%d",++i,i++,++i,++i);
```

خروجی چیست ؟

i=9 | 7 | i=7 i=6

| 9 | i=8 | 7 | 6 |

9776

٤- فرض کنید خروجی برنامه زیر خواسته شده است :

```
int i;
printf("\nI am a %n student",&i);
```

&i متغییری است که محتوای آن عدد برگردانده شده توسط %n (تعداد کاراکترهای چاپ شده تا %n) است .

۵- تابع printf( ) خودش تعداد کاراکترهای چاپ شده را برمی گرداند .

مثال : اگر داشته باشیم :

```
int i;
n= printf("I am a student");
```

مقدار n چه خواهد بود ؟ n=14

۶- اگر تعداد کاراکتر های فرمت از تعداد آرگومانها بیشتر باشد خطا رخ می دهد ولی برعکس آن درست است

مثال :

```
int a=5,b=1;
printf("%d%d",a); → غلط
printf("%d",a,b);  →خروجی=۵
```

خواندن اطلاعات از ورودی :

scanf("عبارت۲","عبارت۱");

عبارت ۱ شامل : ۱- کاراکتر فرمت ۲- کاراکترهایی که عینا چاپ می شوند .

عبارت ۲ شامل : متغییرهایی هستند که اطلاعات از صفحه کلید به آنها خوانده می شود .

مثال :

```
int a,b;
scanf("%d%d",&a&b);
```

دو عدد را از صفحه کلید خوانده و یکی را در a و دیگری را در b قرار می دهد .

1) 5 10↵ → a=5,b=10

2) 5↵ → a=5,b=10
   10↵

scanf کاراکترهای space وenter را به عنوان پایان ورودی می داند .

مثال :

```
scanf("a=%d and b=%d",&a,&b);
```

عبارت داخل " " باید عینا وارد گردد .

a=5 and b=10↵

طول میدان در تابع printf( ) :

۱- طول میدان اعشاری :

%w.df

طول میدان — تعداد اعشاری

مثال : اگر داشته باشیم :

```
double x=72.659
printf("%f%6.2f%-6.0f%06.0f%3.2f",x,x,x,x,x);
```

(1) (2) (3) (4) (5)

خروجی چیست ؟

(1) 72.659000 تعداد ارقام اعشار بصورت پیش فرض ۶ رقم میباشد .

(2) 72.66 طول میدان ۶ میباشد دو رقم برای اعشار و یکی برای نقطه اعشاری

(3) 73 منفی باعث میشود کاراکترهای خالی در سمت راست جمع شوند .

(4) 000073 0 باعث میگردد که خالی ها با 0 پر گردند .

(5) 72.66 اگر طول میدان کمتر باشد بطوری که برای قسمت صحیح جا نماند در آن صورت طول کل میدان نادیده گرفته میشود . اما دو رقم اعشار مد نظر است .

۲- طول میدان در رشته ها :

%m.ns

طول میدان — nکاراکتر از رشته s

```
char st[]="computer";
printf("%10.3s%5s,%0.6s",st,st,st);
```

① ② ③

① com

② computer

③ comput

۷ کاراکتر خالی در سمت چپ قرار میگیرد .

چون طول میدان داده شده کمتر از کل رشته است پس کل رشته چاپ میشود .

۶کاراکتر چاپ خواهد شد و از طول کل میدان صرف نظر میشود .

اگر گفته می شد %5.0s آنگاه ۵ خانه خالی چاپ می شد .

سوال تستی ۱ :

اگر داشته باشیم :

```
int a=20,b;
b=a?a++,printf("%d",a):a+b;
```

ابتدا a++ میشود سپس ۲۱ چاپ شده و نتیجه printf که تعداد کاراکترهای چاپ شده است (۲) در b قرار میگیرد .

سوال تستی ۲ :

اگر داشته باشیم :

```
int a=20,b;
b=a<20 ? b+=7 : (a++)+printf("%d",a);
```

محتوای aوb چیست ؟ b=22 و a=21

نکته :

ch=getch( )	کاراکتری را خوانده در ch قرار میدهد
getch( )	منتظر میماند کلیدی فشار داده شود
ch=getche( )	کاراکتر خوانده شده روی صفحه نمایش داده میشود
ch=getchar( )	همانند getch است اما بعد از ورود کاراکتر↵نیز لازم است
ch=getchare	همانند getche است اما بعد از ورود کاراکتر↵نیز لازم است
putchar(ch)	چاپ محتوای ch در ابتدای سطر بعدی
putch(ch)	چاپ محتوای ch در ابتدای سطر بعدی نه لزوما در ابتدای خط

دستورالعمل های شرطی :

دستورالعمل if :

```
if(شرط)
{
مجموعه دستورات ۱
}
else
{
مجموعه دستورات ۲
}
```

شرط بررسی شده و در صورت درستی شرط مجموعه دستورات ۱ و در صورت غلط بودن شرط مجموعه دستورات ۲ اجرا میشود .

نکته :

- هر عدد به غیر از صفر درست است .
- اگر بعد از شرط فقط یک دستور باشد نیازی به { } نیست .

مثال :

```
if(-2)
  printf("\n****");
```

اگر چندین if وelse داشته باشیم هر else به نزدیک ترین if مربوط میباشد .

مثال ۱ : اگر داشته باشیم :

```
if(x>0)
  if(y>0)
    printf("1");
  else
    printf("2");
else
  if(x==0)
    printf("3");
  else
    printf("4");
```

خروجی برنامه به ازای x=5 ، y=-2 و x=0 ، y=2 چیست ؟ الف) ۲ ب) ۳

مثال ۲ : اگر داشته باشیم :

```
int a =5 ;
if(a=10)
  printf("AAA");
else
  printf("BBB");
```

توجه : a=10 عملیات انتساب میباشد نه تساوی پس محتوای a برابر ۱۰ است و if(10) درست است .

دستورالعمل if...else if ...else

```
if(شرط۱)
{
دستورات ۱
}
else if(شرط۲)
{
دستورات ۲
}
else if(nشرط)
{
دستورات n
}
else
{
دستورات n+1
}
```

در ابتدا شرط ها بررسی میشوند وقتی به شرط درست رسیدیم دستورات مربوط بـه آن اجـرا شـده و خاتمـه . اگـر تمامی شرط ها غلط باشند دستورات n+1 و اگر تمامی شرط ها درست باشد دستورات ۱ اجرا میشود .

مثال : اگر داشته باشیم :

```
int a=8;
if(a)
   if(a==8) puts("1");
   else if(a==8) puts("2");
   else if(a==8) puts("3");
else
   puts("4");
```

خروجی برنامه چیست ؟ عدد ۱ چاپ میشود .

دستورالعمل switch :

```
switch(مقدار عبارت)
{
   case <مقدار۱> : دستورات۱ ;
      break;
   case <مقدار۲> : دستورات۲ ;
      break;
   case <مقدار۳> : دستورات۳ ;
      break;

   case <مقدارn> : دستوراتn ;
      break;
   defult : دستورات n+1 ;
}
```

ابتدا مقدار عبارت ارزیابی میشود . اگر برابر مقدار ۱ باشـد دسـتورات جلـوی case اول اجـرا شـده و break موجـب خروج از بلوک switch خواهد شد . . اگر برابر مقدار ۲ باشد دستورات جلوی case دوم اجرا شـده و break موجـب خروج از بلوک switch خواهد شد و الی آخر و در صورتی که هیچ یک از مقادیر ۱ تـا n نباشـد defult و دسـتورات n+1 اجرا میشود .

چند نکته از switch :

۱- جلوی switch یعنی بجای مقدار عبارت عدد صحیح یا کاراکتر قرار می گیرد و یا میتوان از عبارتهای منطقـی ، رابطه ای یا محاسباتی استفاده نمود ، در نهایت نتیجه باید عددی صحیح باشد و نمیتوان از اعـداد اعشـاری و رشته ها استفاده نمود .

۲- اگر بعد از دستورات یک case دستور break را ننویسیم دستورات جلوی case بعدی نیز اجرا میشود .

مثال : اگر داشته باشیم :

```
int a=2,k=5;
switch(a)
{
case 1 : k=k+a;
case 2 : k=k-a ;
case 3 : k=k+3*a ;
case 4 : k=-k-a ;
}
```

مقدار k چیست ؟ مقدار k عدد ۱۱ خواهد بود .

۳- defult را میتوان در هرکجای switch نوشت یا اصلا ننوشت در هر حال defult زمانی اجرا میشود که مقدار عبارت switch با هیچ کدام از case ها برابر نشود.

```
switch(3)
{
 defult : printf("4");
          break;
 case 2 :
 case 3 : printf("3");
}
```

اگر a=3 → 3
اگر a=2 → 3
اگر a=5 → 4
اگر a=-2 → 4

اما باید مواظب باشیم که دو case یکسان نوشته نشود چون خطای کامپایل رخ می دهد .

٤- اگر جلوی case چندین مقدار را با , از یکدیگر جدا کنیم آخرین مقدار حساب است پس آخرین مقادیر نباید تکرار شوند .

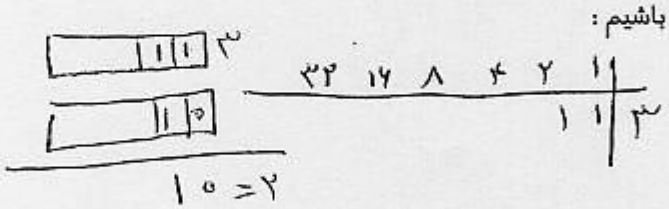
مثال ۱ : اگر داشته باشیم :

```
int x=5;
switch(x)
{
 case 1,3,7 : printf("AAA"); break;
 case 1,5,4 : printf("BBB"); break;
 case 4,2,5 : printf("CCC"); break;
}
```

حاصل چه خواهد بود ؟ عبارت "CCC" چاپ خواهد شد .

مثال ۲ : اگر داشته باشیم :

```
int i=3;
switch(i & 2)
{
 case 1 : printf("One");
 case 2 : printf("Two");
 default : printf("OK");
 case 3 : printf("Tree");
}
```



حاصل چیست ؟ عبارت "TwoOKTree" چاپ خواهد شد.

حلقه های تکرار :

حلقه for :

```
for(مقدار اولیه;شرط;گام حرکت)
{
مجموعه دستورات
}
```

اگر شرط غلط باشد حلقه خاتمه می یابد .

مثال : اگر داشته باشیم :

```
for(i=1;i<5;i++)
   printf("\n i=%d",i);

printf("\n i=%d",i);
```

خروجی چیست ؟

نکته :

- در گام حرکتی حلقه for عملگر i++ و ++i با یکدیگر تفاوتی ندارند .
- اگر بعد از حلقه ( )for علامت ; بیاید تنها دستور for تکرار خواهد شد تا به شرط غلط برسد .

مثال ۱ : اگر داشته باشیم :

```
for(i=1;i<5;i++);
  printf("%d",i);
```

خروجی چیست ؟ خروجی عدد ۵ خواهد بود . ولی اگر ; را از آخر حلقه برداریم نتیجه ۱۲۳٤ میشود .

مثال ۲ : اگر داشته باشیم :

```
for(i=0;i<10;printf("%d%d",++i,i++))
```

```
20
42
64
86
108
```

خروجی چیست ؟

نکته : در printf محاسبات از راست به چپ و چاپ از چپ به راست خواهد بود .

حلقه ها میتوانند تودرتو باشند :

مثال ۱ : اگر داشته باشیم :

```
for(i=1;i<=4;i++)
{
  for(j=1;j<=4;j++)
     printf("*");
}
```

```
****
****
****
****
```

خروجی چیست ؟ بعد خروج از حلقه i=5 و j=5 خواهد بود.

اگر حلقه بیرونی یکبار اجرا شود حلقه داخلی ٤ بار تکرار میشود .

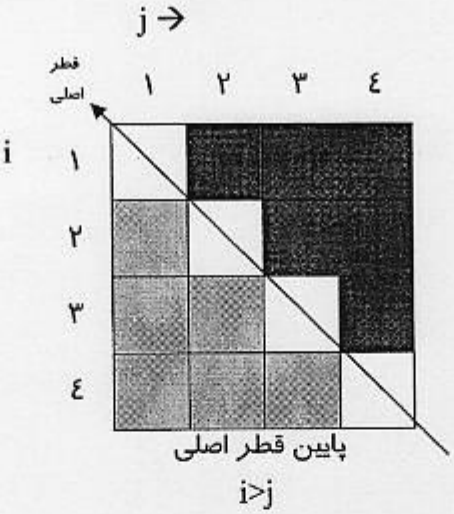
مثال ۲ : اگر داشته باشیم :

```
for(i=1;i<=4;i++)
{
  for(j=1;j<=I;j++)
     printf("*");
  printf("\n");
}
```

```
*
**
***
****
```

خروجی چیست ؟

نکته :



نکات حلقه for :

۱- در هرحلقه for دو سمیکالن داریم .

۲- for(; ;) حلقه بینهایت است .

۳- for(;0;) حلقه همیشه غلط است و اصلا تکرار نمیشود .

حلقه while :

```
while(شرط)
{
مجموعه دستورات
}
```

تا زمانی که شرط درست است مجموعه دستورات اجرا میشود .

مثال : اگر داشته باشیم :

```
int a=4;
while(a--)
{
printf("\n%d",a);
printf("---");
}
```

اول a بررسی شده سپس - - اعمال میگردد

```
3
---
2
---
1
---
0
```

خروجی چیست ؟

اگر بجای - -a از a- - استفاده کنیم خط آخر چاپ نخواهد شد .

حلقه do . . . while :

```
do
{
مجموعه دستورات
}while(شرط);
```

ابتدا داخل حلقه یکبار اجرا میشود سپس شرط بررسی شده و اگر شرط درست باشد دوباره اجرا خواهد شد و . . .

مثال ۱ : اگر داشته باشیم :

```
int n,r;
scanf("%d",&n);
do
{
r=n%10;
cout<<r;
n=n/10;
} while(n != 0);
```

خروجی چیست ؟ عددی را گرفته و بر عکس چاپ میکند. اگر n=325 باشد خروجی 523 خواهد بود .

مثال ۲ : اگر داشته باشیم :

```
int a,b=2;
while(b++<3)
{
  a=1;
  while(a<b)
  {
    if(a==b) break;
    printf("%d",a++);
  }
}
```

مرحله اول :

```
while(2<3)  b=3
while(1<3)
{
چاپ میشود1
a=2
}
```

مرحله دوم :

```
while(2<3)  b=3
while(2<3)
{
چاپ میشود2
a=3
}
```

⟹ ۱۲ چاپ میشود

سوالات تستی ۱ : اگر داشته باشیم :

```
int a=1;
(a-1) ? (a) ? puts("1") : puts("2") : (a+1) ? puts ("3") : puts("4");
```

عبارت ۱ — عبارت ۲

خروجی چیست ؟ عدد ۳ چاپ میشود .

سوالات تستی ۲ : برنامه زیر چکار میکند ؟

```
int k;
char c;
for(k=1;(c=getch())!='A';k--)
   putch('A');
```

کاراکتری را از صفحه کلید خوانده و اگر کاراکتر وارد شده A نباشد A را چاپ میکند . k تاثیری در حلقه ندارد .

میتوان برنامه را بصورت زیر هم نوشت :

```
1) for( ;(c=getch())!='A'; )
     putch('A');
2) for(char c ; (c=getch())!='A';putch('A'));
```

سوالات تستی ۳ : برنامه زیر چکار میکند ؟

```
while(!kbhit() && printf("OK"));
```

تابع kbhit() در فایل عنوان conio.h قرار دارد و کار این تابع این است که اگر کلیدی زده شود ۱ برمیگرداند در غیر اینصورت صفر برمیگرداند .

در نتیجه حلقه OK را چاپ میکند تا کلیدی زده شود .

آرایه ها :

خانه های متوالی از RAM را که هم نوع میباشند و تحت یک نام شناخته میشوند آرایه گویند .

نوع نام [طول ثابت];

```
int a[5];
```

اندیس	۰	۱	۲	۳	٤
a	۲۵	۱۸			۲۰

دسترسی به عناصر آرایه از طریق اندیس صورت میگیرد .

```
a[0]=25;  a[1]=18;  a[4]=20;
```

سرعت دسترسی به تمام خانه های آرایه یکسان است .

مقدار دهی اولیه به آرایه ها :

```
int a[5]={25,18,19,3};
```

	0	1	2	3	4
a	25	18	19	3	0

اگر در مقدار دهی اولیه مقادیر کمتر از تعداد خانه باشد مابقی خانه ها صفر میشود . ولی اگ مقادیر بیشتر باشند خطا خواهد داد .

آدرس عناصر :

```
int a[5];
&a[i]=α+( i – l ) w
```

اندازه هر خانه به بایت

اندیس اولین خانه که در c صفر است

ادرس اولین خانه

```
&a[i]=α+iw
```

مثال : در آرایه زیر آدرس اولین خانه ۲۰۰ است &x[10] و &x[15] را محاسبه کنید .

```
float x[20];

α=200    w=4
&x[10]=200+10*4=240
&x[15]=200+15*4=260
```

برای پردازش آرایه ها از حلقه استفاده میکنند .

مثال ۱ : محتوای آرایه a[ ] چیست ؟

```
int a[5];
for(i=0;i<5;i++)
  a[i]=2*i;
```

a	0	2	4	6	8

مثال ۲ : اگر داشته باشیم :

```
for(i=4;i<=0;printf("%5d",a[i--]);
```

حاصل چیست ؟ جواب : محتوای آرایه را برعکس چاپ میکند .

مثال ۳ : اگر داشته باشیم :

```
int a[10]={0,1,2,3,4,5,6,7,8,9};
for(i=0,j=9 ; i<5 ; i++ , j--)
{ k=a[i];  a[i]=a[j];  a[j]=k; }
```

حاصل چیست ؟ این برنامه محتوای آرایه a را معکوس میکند .

مثال ٤ : اگر داشته باشیم :

```
int k[20],i;
for(i=0,k[i]=i; i<20;k[i]=i , k[++i]=i,i++);
for(i=0;i<20;i+=6)
  printf("%d%d",k[i],k[++i]);
```

حلقه اول آرایه را با اندیس های آرایه پر میکند .

a	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

حلقه دوم نیز ۱۱۸۸۱۵۱۵ را چاپ میکند .

آرایه های دوبعدی:

پردازش آرایه های دوبعدی با دو حلقه تودرتو میباشد.

```
int m[3][4];
for(i=0 ; i<3 ; i++)
  for(j=0 ; j<4 ; j++)
    m[i][j]=i*j ;
```

به ازای هر حلقه بیرونی i حلقه داخلی ٤ بار تکرار میشود .

مثال :

```
int m[3][4]={7,8,9,3,4,12,16,17,5};
```

7	8	9	3
4	12	16	17
5	0	0	0

```
int m[3][4]={{7,8,9,3},{4,12,16,17},{5}};
```

انتساب یک آرایه در آرایه دیگر معنی ندارد .

```
int a[10],b[10];
a=b;
a[]=b[];
```

کپی و انتساب مقادیر آرایه B به آرایه A تنها بصورت خانه به خانه امکان پذیر است.

```
for(i=0 ; i<10 ; i++)
  a[i]=b[i];
```

i=0 → A[0]=b[0]
i=1 → A[1]=b[1]

i=9 → A[9]=b[9]

این حلقه عناصر آرایه B را به عناصر متناظر ارایه A کپی میکند .

حلقه بالا میتواند بصورت زیر نیز نوشته شود :

```
for(i=0 ; i<10 ; a[i]=b[i++]);
```

ابتدا محتویات b[i] را در a[i] قرار میدهد سپس به i یک واحد اضافه می گردد.

در هنگام مقداردهی به آرایه های چند بعدی براکت سمت چپ میتواند خالی باشد.

```
int a[][4]={{1,2,3,4},{5,6,7,8}};
```

1	2	3	4
5	6	7	8

محتوای a[1][3] برابر ۸ میباشد .

مثال ۱ : اگر داشته باشیم :

```
for(i=0 ; i<n ; i++)
  for(j=i+1 ; j<n ; j++)
    a[i][j]=0;
```

چه عملی روی ماتریس a_{n*n} انجام میگیرد ؟ جواب : عناصر بالای قطر اصلی را صفر میکند .

اگر برنامه بالا را بصورت زیر تغییر دهیم عناصر پایین قطر اصلی صفر خواهند شد .

```
for(i=0 ; i<n ; i++)
  for(j=0 ; j<i ; j++)
    a[i][j]=0;
```

مثال ۲ : اگر داشته باشیم :

```
int a,b=0;
int c[10]={1,2,3,4,5,6,7,8,9,0}
for(a=0 ; a<10 ; a++)
  if (c[a] %2 == 0)
     b+=c[a];
```

مقدار نهایی b چیست ؟ جواب : مقادیر زوج آرایه با هم جمع گردیده و در b ذخیره میشوند و در نهایت b=20 میشود

مثال ۳ : به فرض آرایه n با جمله زیر اعلان شده باشد :

```
int n[10]={7,5,3,1,9,6,0,4,2,8};
```

نتیجه اجرای روتین زیر چیست ؟

```
int i,j,t;
for(i=0,j=9;i<5;i++,j--)
   t=n[i]; n[i]=n[j]; n[j]=t;
```

این تکه برنامه آرایه را معکوس میکند واضح است که ابتدا i به خانه صفر و j به خانه ۹ اشاره میکند ، بعد از اینکه محتوای خانه صفر و ۹ عوض شد i=1 و j=8 میگردد حال محتوای خانه ۱ و ۸ عوض میگردد سپس i=2 و j=7 میشود چون i تا خانه ٤ جلو میرود در نتیجه آرایه معکوس میشود . اگر i تا خانه ۹ جلو میرفت آرایه دوباره معکوس میشد و در نتیجه تغییری در آرایه بوجود نمی آمد .(بجای شرط میتوان j<5 را نوشت)

در حلقه بالایی کنترل روی i بود ، میتوانیم کنترل را روی j داشته باشیم و بجای i<5 بنویسیم j>5 . در صورتی که از j>0 استفاده گردد در آرایه تغییری رخ نخواهد داد .

```
int array[10]={0};
```

دستور فوق آرایه ای تعریف میکند که تمام مقادیر آن صفر است.

رشته ها String :

در زبان C نوع داده رشته نداریم و رشته را بصورت آرایه ای از کاراکترها تعریف میکنیم .

مثال : `char st[10];`

st آرایه ای از نوع کاراکتری است که حداکثر ۹ کاراکتر را ذخیره میکند . البته کاراکتر \0 بطور اتوماتیک در آخر رشته قرار میگیرد . اگر در این آرایه رشته studet قرار گیرد خواهیم داشت

st	s	t	u	d	e	n	t	\0

مقداردهی اولیه به رشته ها :

```
char st[10]="student";
char st[10]={'s','t','u','d','e','n','t','\0'};
char st[10]={113,114,120,100,101,...,0};
```

میتوان در مقداردهی اولیه طول رشته را مشخص ننمود.

```
char st[]="student";
char st[]={'s','t','u','d','e','n','t'};
```

زبان C طول رشته ها را کنترل نمی کند و وظیفه کنترل طول رشته ها بر عهده برنامه نویس است . برای مثال اگر آرایه ای به طول ۱۰ خانه تعریف گردد و برنامه نویس بخواهد در اندیس ۱۵ این آرایه کاراکتری را ذخیره کند ، خطایی رخ نخواهد داد . فقط ممکن است کاراکترهای اضافی پس از آرایه در حافظه قرار گیرند و محتوای متغییرهایی که بعد از آرایه در حافظه گرفته اند را از بین ببرد.

```
char s[5];
```

	0	1	2	3	4
s					

اگر در این آرایه رشته student را قرار دهیم خواهیم داشت :

	0	1	2	3	4			
s	s	t	u	d	e	n	t	\0

کاراکترهای اضافی در حافظه بعد از آرایه قرار میگیرد و اگر متغییر دیگری در این محل حافظه باشد محتوای آن از بین خواهد رفت . این عمل موجب خطای کامپایلری نمیشود ولی خطای منطقی رخ میدهد.

نکته :

انتساب رشته ها به یکدیگر معنی ندارد زیرا رشته ها آرایه ای کاراکتری هستند .

```
char s1[]="ali",s2[4];
s2=s1;
s2[]=s1[];
```

برای انتساب رشته ای به رشته دیگر میتوان عناصر رشته را خانه به خانه به یکدیگر کپی کرد .

```
for(i=0;s1[i];i++)   → حلقه تا زمانی ادامه می یابد که s1[i] کاراکتر و مخالف \0 باشد
  s2[i]=s1[i];
s2[i]='\0';          → کاراکتر \0 در آخر رشته دوم قرار گیرد
```

البته برای راحتی کار با رشته ها در زبان C توابع مخصوصی در فایل عنوان string.h تعریف گردیده است .

این توابع عبارتند از :

1) strcpy	2) strlen	3) strcmp	4) strcmpi	5) strcat
6) strset	7) strupr	8) strlwr		

تابع strcpy( ) : برای کپی یک رشته در رشته دیگر :

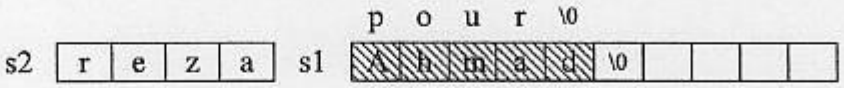
```
char s1[10];
strcpy(s1,"reza");
```

	0	1	2	3	4					
s	r	e	z	a	\0					

نکته : همانطور قبلا گفتیم در زبان C کنترلی بر روی طول رشته انجام نمی گیرد و این وظیفه بر عهده برنامه نویس است . مثال زیر اهمیت کنترل طول رشته را نشان می دهد .

```
char s1[10],s2[4];
strcpy(s1,"Ahmad");
strcpy(s2,"Rezapour");
puts(s1);
```

در RAM متغیر ها از راست به چپ تعریف میشوند .



عبارت pour چاپ خواهد شد .

تابع ( )strlen : این تابع طول رشته را بر میگرداند ، در طول رشته 0\ محاسبه نمی گردد .

```
char s1[]="BOOK";
printf("%d",strlen(s1));
```

خروجی عدد ٤ خواهد بود .

توابع ( )strcmp و( )strcmpi : برای مقایسه دو رشته بصورت زیر بکار میرود :

strcmp(st1,st2)=
- عدد مثبت st1>st2
- 0 st1==st2
- عدد منفی st1<st2

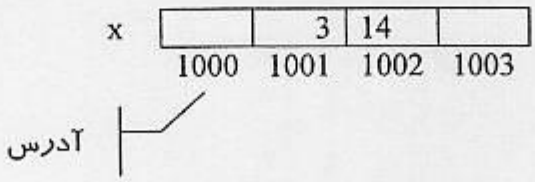
مثال :

```
char st1="Ali";
char st2="Ali";
if(!strcmp(st1,st2))
  printf("باهم برابرند");
```

اشاره گرها :

حافظه RAM کامپیوتر از بایت ها تشکیل شده است هر بایت دارای شماره منحصر به فردی است . زمانی که متغییری تعریف می گردد بسته به نوع و اندازه اش چند بایت RAM را اشغال می کند ، به شماره اولین بایت متغییر آدرس آن متغییر گویند .

```
float x=3.14;
```



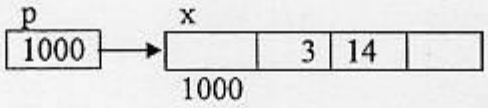
اگر متغییری بزرگ هم باشد شماره اولین بایت آدرس آن متغییر خواهد بود .

متغییرهای اشاره گر که آدرس ها را ذخیره میکنند اعدادی صحیح هستند و معمولا در مبنای ۱۶ بیان میشوند ، اما به دلیل اینکه روی اشاره گرها اعمال خاصی تعریف گردیده نمی توان اشاره گرها را از نوع int تعریف کرد .

متغییر اشاره گر به صورت زیر تعریف می گردد :

نوع * نام ;

```
float *P;
p=&x;
```



p* محتوای مکانی است آدرس آن در p است .

متغییرهای اشاره گر نوع ندارند و هر متغییر اشاره گر دو بایت فضا اشغال میکند .

float *p	در p آدرس متغییرهای اعشاری قرار میگیرد و دوبایت فضا میگیرد
int *q	در q آدرس متغییرهای اعشاری قرار میگیرد و دوبایت فضا میگیرد
char *r	در r آدرس متغییرهای اعشاری قرار میگیرد و دوبایت فضا میگیرد

```
float *p[10][20];
```

آرایه ای ۲۰۰ خانه ای ازنوع اشاره گر تعریف شده است و هر خانه اشاره گر بوده و ۲ بایت اشغال میکند پس کل آرایه ٤۰۰ بایت فضا خواهد گرفت .

اعمال روی اشاره گر ها :

بر روی اشاره گرها سه عمل جمع وتفریق ، اعمال مقایسه ای و انتساب تعریف شده است .

الف) جمع و تفریق : اگر به متغییر اشاره گر یک واحد اضافه گردد به محتوای آن به اندازه طول نوع اضافه میشود .

مثال : فرض کنید داریم float *p و محتوای pعدد ۱۰۰۰ است اگر بنویسیم p=p+1 در آن صورت محتوای p عدد ۱۰۰٤ خواهد بود .

float *p	int *q	char *r	100	
		r+1	101	
q+1		r+2	102	
		r+3	103	
p+1	q+2		104	
			105	
	q+3		106	
			107	
p+2	q+4		108	

مثال : فرض کنید داریم float *p و محتوای p عدد ۵۰۰۰ است ، اگر بنویسیم p; -- محتوای p چقدر است؟ ٤۹۹۶

ب) اعمال مقایسه ای :

```
int *p,*q;
if(p==q)
{
---
---
}
```

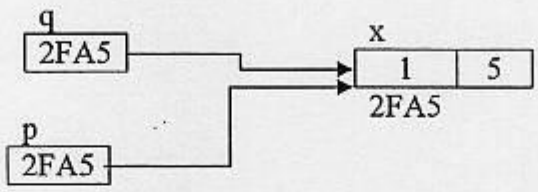
یعنی اگر p و q به یکجا اشاره کنند

```
if(p<q)        if(p>=q)       if(p!=q)       if(p!=NULL)    if(p)          while(p!=NUUL)
{              {              {              {              {              {
---            ---            ---            ---            ---            ---
---            ---            ---            ---            ---            ---
---            ---            ---            ---            ---            ---
}              }              }              }              }              }
```

ج) انتساب :

```
int a,*p,*q;
p=&a;

*p=15;
q=p;
printf("%d",*q);
```



اعمالی مانند ضرب ، تقسیم ، باقیمانده ، بیتی و ... روی روی متغییرهاب بیتی تعریف نشده اند .

حافظه پویا :

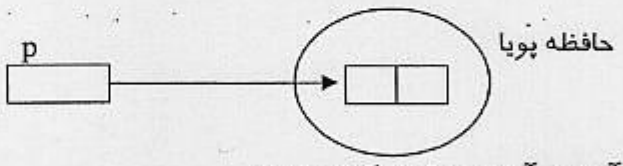
می توان هنگام اجرای یک برنامه از RAM فضا اخذ کرده و با آن کار کنیم ،پس از پایان عملیات میتوان این فضا را آزاد نمود. به عبارت دیگر به حافظه RAM برگرداند . به این نوع حافظه ، حافظه پویا می گویند .

اخذ حافظه در زبان C با تابع malloc و در C++ با new انجام می گیرد.

مثال :

```
int *p;
p=(int *)malloc(sizeof(int));
```

ابتدا تابع malloc به اندازه ۲ بایت حافظه اخذ می کند . سپس محتوای آن را با (* int) از نوع int تعریف کرده و در آخر آدرس آن در p قرار می گیرد .



نام ندارد و فقط آدرس آن در p قرار گرفته است .

```
*p=17;
*p=*p+3;
```

چون حافظه از نوع int تعریف شده کلیه عملیات مربوط به int را میتوان انجام داد.

free(p) : حافظه پویا آزاد میشود .

نکته : متغییر p خودش پویا نیست بلکه حافظه اختصاص یافته پویا است پس هنگام free(p) خود p حذف نمی گردد بلکه تنها آدرس اختصاص یافته به آن آزاد می گردد .

مثال ۱ : اگر داشته باشیم :

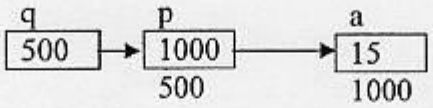
```
int a=5,b=10,*r;
r=(int*)malloc(sizeof(int));
*r=a;
a=b;
b=*r;
```

محتوای a و b چقدر است ؟ a=10,b=5

متغییرهای اشاره گر به شاره گر :

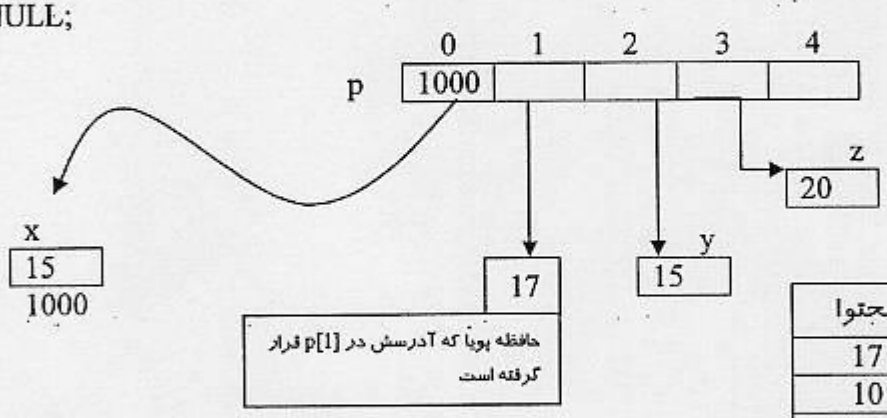
متغییرهای اشاره گر در حافظه ۲ بایت فضا میگیرند پس دارای آدرس هستند و می توان آدرس آنها را در متغییرهای اشاره گر به اشاره گر قرار داد .

```
int a=15,*p,**p;
p=&a;
q=&p;
```



می توان آرایه ای از متغییرهای اشاره گر تعریف کرد.

```
int *p[5];
int x=15,y=10,z=20;
p[0]=&x;
p[1]=(int*)malloc(sizeof(int));
*p[1]=x+2;  x+2 را قرار بده در جایی که p[1] به آن اشاره میکند
p[2]=&y;
p[3]=&z;
p[4]=NULL;
```



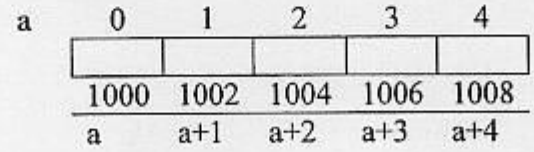
محتوا	
17	*p[1]
10	*p[2]
20	*p[3]
&z	p[3]

دستورات مقابل اشتباه بوده و خطای کامپایل دارد : p[2]=y; p[2]=25;

آرایه ها و اشاره گرها :

اسم آرایه در C حاوی آدرس اولین خانه آرایه است .

```
int a[5];
```



&a[0] همان a و &a[1] همان a+1 و &a[2] همان a+2 و &a[i] همان a+i است .

	محتوا
*a	a[0]=15
*(a+1)	a[1]=25
*(a+i)	a[i]

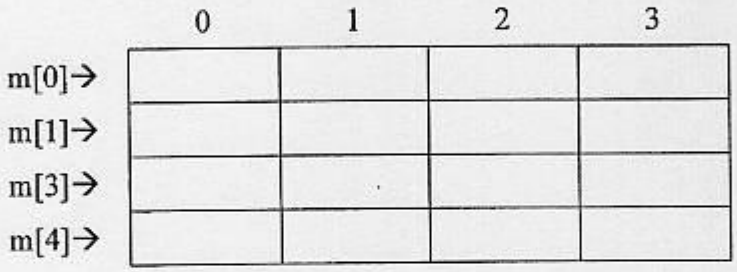
نکته :

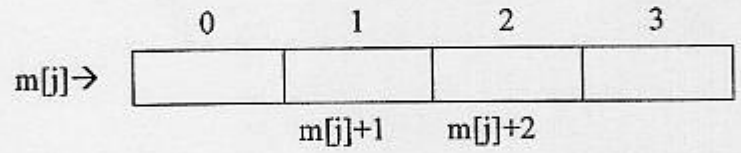
درست است که اسم آرایه مانند متغییرهای اشاره گر است ولی ما نمی توانیم در اسم آرایه آدرس متغییرهای دیگر را قرار دهیم .

```
int a[15],x=15;
a=&x; غلط است
```

```
int a[15],*p;
p=a;
p[0]=35;   این دستور یعنی اینکه در عنصر صفر آرایه عدد ۳۵ ذخیره می گردد
p[1]=30; ≡ a[1]=30 ≡ *(p+1)=30;  هر سه دستور معادل هستند
```

آرایه های دوبعدی و اشاره گرها :





آدرس :

&m[0][0]	m[0]	
&m[0][1]	m[0]+1	
&m[2][0]	m[0]+2	
&m[2][3]	m[2]+3	*(m+2)+3
&m[i][j]	m[i]+j	*(m+i)+j

محتوا :

m[0][0]	*m[0]	
m[0][1]	*(m[0]+1)	
m[2][0]	*(m[0]+2)	
m[2][3]	*(m[2]+3)	*(*m+2)+3)
m[i][j]	*(m[i]+j)	*(*(m+i)+j)

مثال : دستور m[i][j]=25 معادل کدامیک از موارد زیر است ؟

☑ الف) *(m[i]+j)=25

ب) *m[i]+j=25

☑ ج) *(*(m+i)+j)=25

د) *(*m+i+j)=25

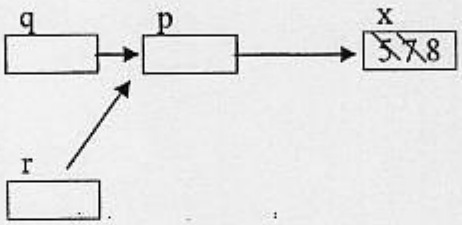
مثال ۲ :

```
float *p[4][2][16];
```

این آرایه ۱۲۸=۱۶*۲*۴ خانه دارد و ۲۵۶=۲*۱۲۸ بایت فضای اشغال شده توسط آرایه P است . در تمامی خانه های این آرایه آدرس متغییرهای اعشاری قرار می گیرد .

مثال ۳ : با اجرای برنامه زیر مقدار p* چه خواهد بود ؟

```
int x=5,*p,**q,**r;
p=&x;
q=&p;
*q=&x;
r=&p;
*p=**r+2;
**q=x+1;
```



نکته :

متغییرهایی که از نوع کلاس حافظه ثبات(register) تعریف می شوند آدرس ندارند .

```
register int x;
int *p;
p=&x;
```

توابع :

انواع تابع :

الف) توابع کتابخانه ای : در فایلهای عنوان تعریف گردیده .

ب) توابعی که برنامه نویس می نویسد .

پارامترهای مهم در بررسی توابع :

الف) الگوی تابع

ب) بدنه یا ساختار تابع

ج) فراخوانی تابع

الف) یک تابع چه پارامترهایی را قبول می کند و چه نوع اطلاعاتی برمی گرداند .

مثال : void f1(void)

چیزی نمی گیرد و چیزی بر نمی گرداند ، مثلا خودش اطلاعات را از صفحه کلید می خواند یا محاسبات انجام میدهد یا چیزی را چاپ میکند .

```
int f2(float,float);
```

دو تا اعشاری می گیرد و عدد صحیح بر می گرداند .

مثال :

```
# include <stdio.h>
# include <conio.h>
//--------------------------------
void func1(void);
void func2(int,int);
int func3(int,int);
//--------------------------------
void main(void)
{
int a=10,b=20,c;
func1();
func2(a,b);
c=func3(a,b);
printf("\nc=%d",c);
getch();
}
//--------------------------------
void func1(void)
{
printf("Welcome");
}
void func2(int x,int y)
{
int s;
s=x+y;
printf("\nsum=%d",s);
return;
}
int func3(int a,int b)
{
return a*b;
}
```

الگوی تابع

فراخوانی تابع

بدنه تابع

متغییرهایی که در داخل یک تابع تعریف می شوند ، متغییرهای محلی نامیده میشوند . در تابع main() متغییرهای a,b,c در تابع func2() متغییرهای s,x,y و در تابع func3() متغییرهای a,b بصورت محلی تعریف شده اند.

خاصیت متغییرهای محلی این است که به محض ورود به تابع بصورت اتوماتیک ایجاد می شوند و به محض خروج از تابع حذف می شوند . اگر برنامه بالا را اجرا کنیم اول main() اجرا شده و به متغییرهای آن حافظه اختصاص می یابد با فراخوانی هر تابع به متغییرهای آن حافظه اختصاص یافته و با اتمام کار تابع حافظه اختصاص داده شده آزاد می گردد و متغییرها از بین می روند.

مواردی که در متغییرهای محلی دارای اهمیت هستند :

۱) حوزه تعریف :

تابعی که در داخل آن تعریف شده اند و در جاهای دیگر قابل استفاده نیستند .

۲) طول عمر :

از زمان فراخوانی تابع تا اتمام کار آن

مقایسه متغییرهای محلی و سراسری (عمومی) :

همانطور که گفتیم متغییرهای محلی در داخل تابع تعریف میشوند ، به این متغییرها از نوع auto نیز می گویند . اما متغییرهای عمومی یا سراسری قبل از تابع main() تعریف میشوند و حوزه آنها کل برنامه است .

در صورتی که در برنامه ای متغییری محلی همنام با متغییر سراسری تعریف گردد اولویت با متغییرهای محلی خواهد بود .

مثال : اگر داشته باشیم :

```
void f(void);
int a=50;
void main()
{
printf("\na=%d",a);
f();
printf("\na=%d",a);
}
//----------------------------
void f(void)
{
a=15;
printf("\na=%d",a);
}
```

خروجی برنامه چه خواهد بود ؟

```
a=50
a=15
a=15
```

هر تابع میتواند به متغییرهای سراسری دسترسی داشته باشد و هر تابع میتواند مقدار آن را عوض کند . اگر تابعی متغییر سراسری را عوض کند توابع دیگر متغییر عوض شده را خواهند دید .

اگر تابع f() بالا را بصورت زیر بنویسیم :

```
void f(void)
{
int a=15;
printf("\na=%d",a);
}
```

در این صورت تابع f() با متغییر سراسری a کار نمی کند بلکه خودش متغییر محلی دارد و مقدار آن را برابر ۱۵ می کند زیرا محلی ها بر سراسری مقدم است . تابع f() متغییر خودش را ۱۵ می کند ، نه متغییر سراسری را لذا متغییر سراسری همچنان ۵۰ است .

کلاس های حافظه :

منظور از کلاس های حافظه،طبقه بندی متغییرها هنگام تعریف است. متغییرها در چهار کلاس طبقه بندی گردیده است.

الف) اتوماتیک	auto int x; متغییرهای محلی
ب) خارجی	extern int x; متغییرهای سراسری
ج)استاتیک	static int x;
د) ثبات	register int x;

کلاس حافظه استاتیک :

۱- استاتیک محلی : دارای خواص زیر هستند :

۱) در داخل تابع تعریف میشوند و فقط در داخل آن تابع قابل استفاده هستند و در بیرون از تابع نمی توان از متغییرهای استاتیک محلی استفاده نمود .

۲) اولین بار که وارد تابع شویم ایجاد میشوند ، با خروج از تابع از بین نمی روند ، بلکه آخرین مقدارشان را حفظ می کنند .

مثال :

```
void test(void);
void main()
{
int i;
for(i=0;i<5;i++)
  test();
getch();
}
//-------------------------
void test(void)
{
int x=0;
static int y=0;
printf("x=%d,y=%d",++x,++y);
}
```

x هر بار حذف شده و دوباره ایجاد میگردد ولی y یکبار ایجاد شده و مقدارش را حفظ می کند .

۲- استاتیک عمومی : بیرون توابع تعریف میشوند و در توابع بعدی قابل استفاده اند .

```
void main()
{
------
------
}
//---------------
static int x=5;
void f1()
{
------
------
}
//---------------
static int y;
void f2(void)
{
-----
-----
}
```

حوزه تعریف x

حوزه تعریف y

اگر متغییرهای استاتیک عمومی را قبل از تابع main تعریف کنیم و برنامه شامل یک فایل باشد ، در آن صورت همانند متغییرهای سراسری خواهد بود اما اگر برنامه بیش از یک فایل باشد در آن صورت استاتیک ها فقط در فایلی معتبرند که در آن تعریف شده اند.

کلاس حافظه ثبات :

متغییرهای نوع کلاس حافظه ثبات در داخل CPU تعریف میشوند لذا سرعتشان زیاد است. پیشنهاد میگردد متغییرهایی که محاسبات زیادی روی آنها انجام میشود از نوع ثبات تعریف شوند . واضح است که تعداد متغییرهای کلاس ثبات انگشت شمار است زیرا در CPU تعداد محدودی ثبات داریم.

نکته ۱:

فقط متغییرهای int و char و اشاره گرها را میشود از نوع ثبات تعریف کرد .

نکته ۲:

آدرس متغییرهای ثبات بی معنی است ، زیرا این متغییرها در RAM نیستند .

نکته ۳:

متغییرهای از نوع ثبات را نمیتوان سراسری تعریف کرد.

مثال۱ :

```
register int x;
int *p;
p=&x;
```

مثال۲ :

```
register int *p;
int x;
p=&x;
```

روشهای فراخوانی توابع :

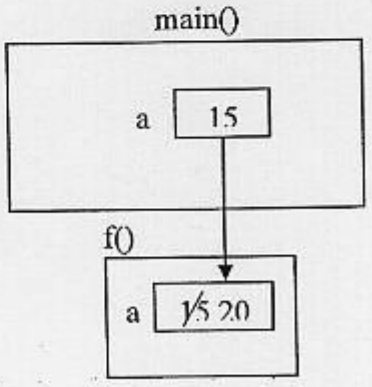
۱) فراخوانی با مقدار : همانطور که قبلا گفتیم در این روش مقدار آرگومان به پارامتر ارسال میگردد.

مثال :

```
void f(int);
void main()
{
int a=15;
f(a);
printf("\na=%d",a);
}
//------------------------
void f(int x)
{
x=20;
}
```

آرگومان

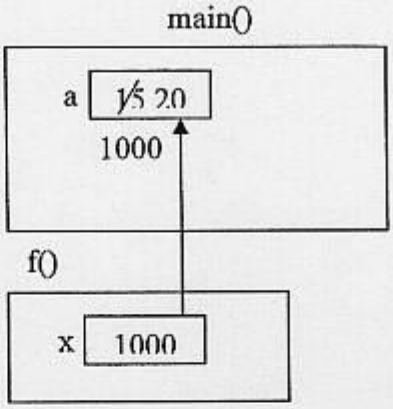
پارامتر



مقدار آرگومان a در پارامتر x کپی شد ، حالا اگر x را تغییر دهیم در a تاثیری ندارد .

۲) فراخوانی با ارجاع :

```
void f(int *);
void main()
{
int a=15;
f(&a);
printf("\na=%d",a);
}
//-------------------------
void f(int *x)
{
*x=20;
}
```

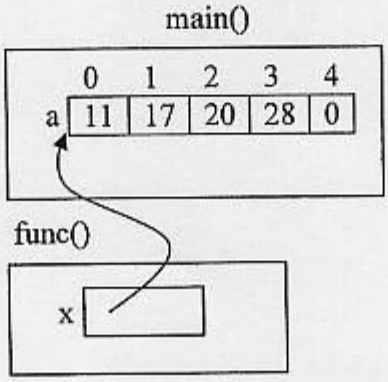


آدرس آرگومان a در پارامتر اشاره گر x کپی می شود یعنی x حاوی آدرس a است و در صورت کار با x با خود a کار می کنیم .

آرایه ها از طریق فراخوانی با ارجاع به توابع ارسال می شوند ، یعنی آدرس آرایه ارسال می شود و آدرس آرایه همان اسم آرایه است.

مثال ۱: خروجی برنامه زیر چیست ؟

```
void func(int *x);
void main()
{
int a[5],i;
func(a);                    → آدرس اولین خانه آرایه
for(i=0;i<5;i++)
printf("%5d",a[i]);         ≡ printf("%5d",*(a+i));
}
//-------------------------
void func(int *x)
{
x[0]=11;                    ≡ *(x+3);
x[1]=17;
x[2]=20;
*(x+3)=*x+x[1];
x[4]=*(x+3)-x[3];
}
```



مثال ۲ : خروجی برنامه زیر چیست ؟

```
void test(int *,int *,int);
main()
{
int x=2,y=3,z=4;
test(&x,&y,z);
printf("%d%d%d",x,y,z);
}
//--------------------------------
void test(int *a,int *b,int c)
{
*a+=c-*b;                    ≡ *a=*a+c-*b;
c+=5;
*b-=*a-c;                    ≡ *b=*b-(*a-c);
}
```

خروجی : 394

ساختارها ، اتحادها ، نوع داده شمارشی :

برای تعریف متغییرهایی که از فیلدهای غیر هم نوع تشکیل میشوند از ساختارها استفاده می شود .

تعریف نوع ساختمان :

```
struct نام ساختمان
{
   نوع     متغییر۱;
   نوع     متغییر۲;
   نوع     متغییرn;
};
```

با این تعریف فقط نوع یا قالب ساختمان مشخص می شود و هیچ متغییری در RAM ایجاد نخواهد شد و حافظه ای هم اشغال نمی شود.

```
struct test
{
int x;
int y;
float a,b;
};
```

این ساختار صفر بایت فضا اشغال می کند چون قالب بوده و متغییری از نوع این ساختمان تعریف نشده است.

به عنوان مثال می خواهیم متغییری تعریف کنیم که شامل نام ، نام خوانوادگی ، شماره دانشجویی و معدل باشد . برای اینکار از ساختار استفاده می کنیم .

```
struct student
{
char name[10];
char family[20];
int num;
float avg;
};
```

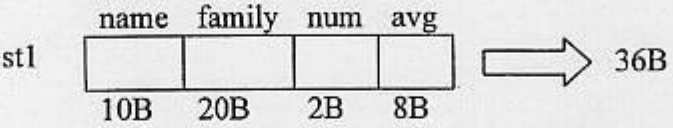
تعریف متغییرهایی از نوع ساختمان :

این نوع متغییرها را می توان به دو روش تعریف کرد .

الف) بلافاصله بعد از تعریف نوع لیست متغییرها را بنویسیم:

```
struct student
{
char name[10];
char family[20];
int num;
float avg;
}st1,st2;
```

st1 و st2 در RAM ایجاد می شوند .



ب) ابتدا قالب را تعریف کرده بعد در هرجای برنامه لازم باشد متغییرهایی از نوع این ساختار تعریف می کنیم .

مثال :

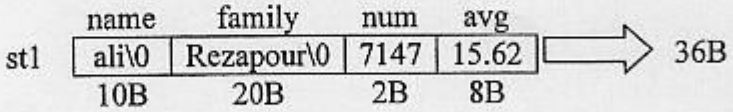
```
struct student st1,st2;        ≡ student st1,st2;
```

هز کدام از متغییرهای بالا ۳۶ بایت از فضای RAM را اشغال می کنند .

دسترسی به فیلدهای ساختمان :

```
; نام فیلد.نام متغییر
strcpy(st1.name,"ali");
strcpy(s1.family,"Rezapour");
st1.num=7147;
st1.avg=15.62;
puts(st1.name);
```

عمل انتصاب در رشته ها از طریق strcpy() انجام می گیرد.



خروجی : ali

مقداردهی اولیه به متغییرهای از نوع ساختمان :

مانند آرایه از طریق آکولاد امکانپذیر خواهد بود .

```
struct student st1={"ali","Rezapour",7147,15.62};
```

در حین تعریف st1 فیلدها نیز پر میشوند .

نکته :

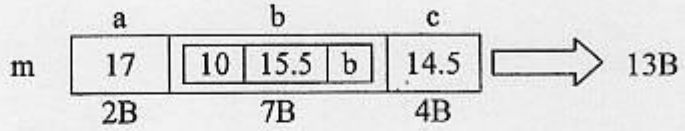
درست است که انتساب با = در آرایه ها معنی ندارد ، اما درمورد متغییرهای ساختمان یا رکوردها درست است . ولی باید دو طرف انتساب از یک نوع باشند .

```
st1=st2;
```

	name	family	num	avg
st2	ali\0	Rezapour\0	7147	15.62

ساختمان های تو در تو :

```
struct AA
{
int x;
float y;
char z;
}
struct BB
{
int a;
struct AA b;
float c;
}m;
```




```
m.a=17;   m.b.x=10;   m.b.y=15.5;   m.b.z='b';   m.c=14.5;
```

آرایه ای از ساختمانها :

در هر متغییر از نوع ساختمان فقط میتوان مشخصات یک چیز را قرار داد مثلا در st1 میتوان مشخصات یک دانشجو را قرار داد . برای اینکه بتوان مشخصات چندین دانشجو را ذخیره کرد باید آرایه ای از ساختمان تعریف نمود .

```
struct student st[10];
```

فرض بر این است که ساختمان student قبلا تعریف شده است .

```
strcpy(st[0].name,"jaber");
strcpy(st[0].family,"karimpour");
st[0].num=1719;
st[0].avg=17.19;
```

st

0	jaber	karimpour	1719	17.19
1				
2				
3				
4				
5				
6				
7				
8				
9				

10*36=360B

متغییرهای اشاره گر از نوع ساختمان :

متغییرهایی از نوع ساختمان همانند سایر متغییرها دارای آدرس هستند و می توان متغییرهای اشاره گر از نوع ساختمان تعریف کرد ، تا آدرس متغییرهای ساختمان را در آن قرار داد .

مثال :

```
struct student st1,*p;
p=&st1;
strcpy(st1.name,"ali"); strcpy(st1.family,"Rezapour");
st1.num=7161; st1.avg=15.26;
```

هر فیلد آدرس جداگانه ندارد ، کل رکورد یک آدرس دارد و آن شماره اولین بایت رکورد است .

دسترسی به فیلدها از طریق اشاره گر :

; نام فیلد.(*p) ≡ ; نام فیلد → متغییر اشاره گر

strcpy((*p). name,"ahmad"); ≡ strcpy(p→name,"ahmad"); یعنی در فیلد name رشته ahmad قرار میگیرد

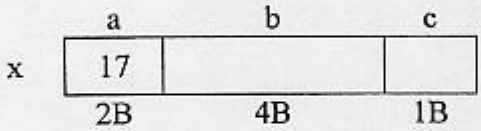
ارسال ساختمان به توابع :

متغییرهایی از نوع ساختمان را به دو روش فراخوانی با مقدار و فراخوانی با ارجاع میتوان به توابع ارسال نمود. قوائد ارسال ساختمان ها به توابع همانند ارسال متغییرهای معمولی است .

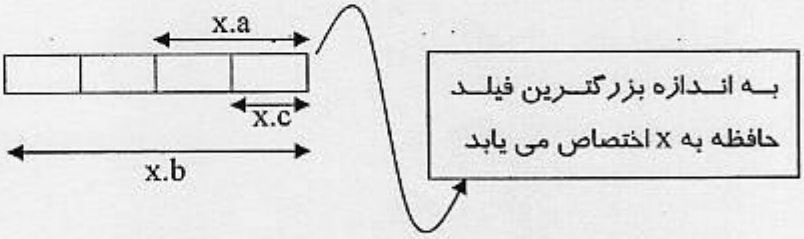
اتحاد unionها :

union همانند ساختمان است ، تنها تفاوت آنها در نحوه استقرار متغییرها (فیلدها) در حافظه است به این صورت که در ساختمان به هر فیلد حافظه جداگانه داده میشود در حالی که در union تمامی فیلدها از حافظه مشترک استفاده می کنند .

```
struct AA
{
int a;
float b;
char c;
}x;
```

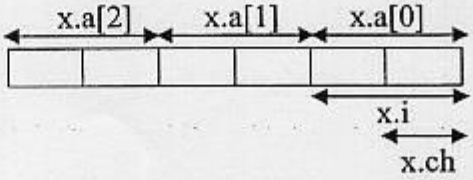



```
union AA
{
int a;
float b;
char c;
}x;
```



واضح است که در یک لحظه مقدار یک فیلد را می توانیم در حافظه داشته باشیم و تغییر مقدار یک فیلد در فیلدهای دیگر تاثیر گذار است .

```
union un
{
char ch;
int i;
int a[3];
}x;
x.a[0]=65;
x.a[1]=66;
x.a[2]=67;
printf("%d%d%c",x.i,x.a[0],x.ch);
x.ch='F';
printf("%d%d%c",x.i,x.a[0],x.ch);
```



خروجی :

```
65 65 A
70 70 F
```

با فرض اینکه 'A'=65 ، 'B'=66 ، 'C'=67 و 'F'=70 است خروجی چیست ؟

نوع داده شمارشی enum :

هدف این است که بتوانیم عناصر یک مجموعه متناهی را شماره گذاری کنیم بدین صورت که داده اول دارای شماره صفر و داده دوم ، شماره یک و الی آخر .

enum نام {داده nام , ... , داده دوم, داده اول};

مثال : خروجی برنامه زیر چیست؟

```
enum color{red,green,blue,brown}x,y,z;
x=red;
y=blue;
z=brown;
printf("%d%d%d",x,green,z);
```

خروجی : 013

نکته :

۱- داده ها نباید رشته ، کاراکتر یا عدد باشند .

```
enum color{"red",20,"b"};
```

۲- متغییرهایی از نوع enum مانند x و y در مثال بالا همانند int هستند یعنی ۲ بایت جا می گیرند و با %d چاپ میشوند .

مثال : مقدار نهایی x چه خواهد بود ؟

```
enum color{red,green,blue=5,blak,brown};
color x;
x=black;
```

مقدار نهایی x برابر 6 است .

ساختمان بیتی :

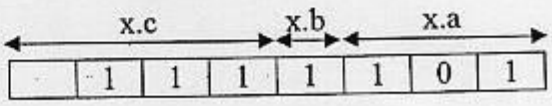
هدف از ساختمان های بیتی ، کار با بیت های حافظه است و به صورت زیر تعریف میشود :

```
struct نام{
طول : فیلد۱ نوع;
طول : فیلد۲ نوع;
طول : فیلد۳ نوع;
طول : فیلدn نوع;
نام متغییرها};
```

مثال :

```
struct byte{
unsigned char a:3;
unsigned char b:1;
int c=4;
}x;
x.a=5;
x.b=1;
x.c=7;
```

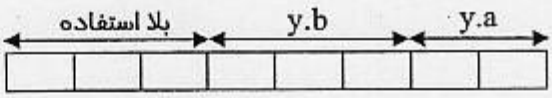
بدان معنی نیست که محتوای فیلد c عدد ٤ شود بلکه c چهار بیت اشغال میکند .



ساختار x یک بایت فضا می گیرد.

میتوان رکوردی تعریف نمود که چند بیت آن بلا استفاده باشد .

```
struct test{
unsigned char a:2;
int b=3;
}y;
```



جهت استفاده از RAM باید از بایت ها استفاده کنیم و نمیتوان بیت به بیت استفاده کرد . اگر تعداد بیت ها کمتر از ۸ بیت باشد یک بایت و اگر کمتر از ۱۶ بیت باشد ۲ بایت و ... فضا اشغال می گردد.

نکته :

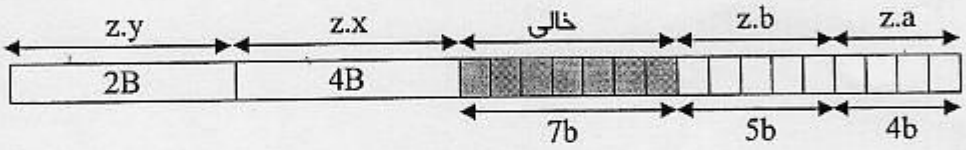
۱- فیلدها فقط می توانند int,char,signed,unsigned و enum باشند و نمی توانند اشاره گر ، آرایه ، اعشاری و ساختمان باشند.

۲- اگر فیلدی از نوع signed تعریف شوند حتما باید ۲ بیت یا بیشتر باشند زیرا در یک بیت نمی شود هم علامت و هم مقدار ذخیره کرد.

```
struct BB{
signed char a=1;
}
```

۳- یک ساختمان بیتی می تواند فیلدهای غیر بیتی هم داشته باشد.

```
struct AA{
int a:4;
int b:5;
float x;
int y;
}z;
```



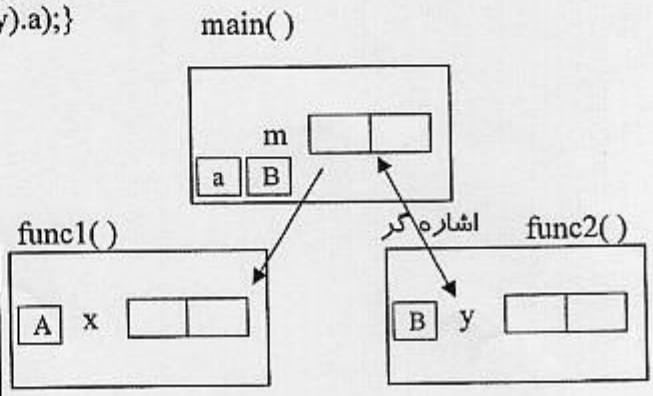
در ساختمان بالا کلا ۶۴ بیت اشغال می گردد.

مثال : کدام گزینه در مورد ساختار داده بیتی درست است .

```
struct s{
unsigned a:5;
signed b:1;
}x={24,12};
```

دستور signed b:1 دارای خطا است چون در یک بیت نمیتوان هم علامت و هم مقدار ذخیره کرد . در صورتی که دستور درست نوشته می شد عدد ۲۴ در a و عدد ۱۲ در b قرار می گرفت.(برای ذخیره عدد ۱۲ به ٤ بیت نیاز است) مثال :

```
union u{int a,char b};
void func1(union u x) {x.a=65;  printf("%c",x.a);  }
void func2(union u *y){y→b='B'; printf("%c",(*y).a);}
void main()
{
union u m;
m.a=97; func1(m);  printf("%c",m.a);
m.b='b'; func2(&m);printf("%c",m.b);
}
```



تابع ( )func1 بصورت مقدار فراخوانی شده ، یعنی محتوای m را به ( )func1 ارسال کردیم و در x کپی شد محتوای x عوض شده و A چاپ شد سپس در ( )main کاراکتر a چاپ شد چون x در m تاثیر ندارد . تابع ( )func2 از طریق ارجاع فراخوانی شده یعنی آدرس m را به y فرستادیم و از طریق y محتوای m را عوض کردیم .

فایلها :

انواع فایل :

۱- فایل برنامه exam.cpp

۲- فایل عنوان conio.h، stdio.h

۳- فایل اجرایی exam.exe

در این فصل در مورد فایل داده بحث خواهیم کرد.

اگر داده ها را در آرایه یا ساختمان ذخیره کنیم با قطع برق یا اتمام اجرای برنامه داده ها از بین می روند زیرا تمامی ساختارهایی که تا اکنون خوانده ایم در RAM قرار می گیرند .

داده ها را به کمک فایل بر روی hard,CD,flash و ... می نویسیم .

انواع فایل داده :

۱- فایل متنی text

۲- فایل باینری یا دودویی binary

در فایل متنی داده ها بصورت کاراکتر قرار می گیرند برای مثال عدد ۱۳۸۶ بصورت '6' '8' '3' '1' ذخیره می گردد. و کاراکترهای ویژه ای برای نشان دادن انتهای هر خط و آخر فایل ذخیره می گردد بطوری که برای مثال Eof نشان دهنده آخر فایل متنی است .

در فایل های دودویی داده ها بصورت مبنای دو قرار می گیرند . برای مثال عدد ۱۳۸۶ چون int است دو بایت فضا می گیرد .

در کل سرعت ذخیره و بازیابی اطلاعات در فایل های دودویی بیشتر از فایل های متنی است . اما فایل های متنی را میتوان مشاهده و چاپ کرد .

تعریف فایل :

```
FILE *fp;
```

بازکردن فایل :

```
fp=fopen("نام درایو:\\نام فایل.پسوند فایل\\","نوع فایل");
if (fp==NULL)
{
printf("\n cannot open file");
exit(1);
}
```

با تابع fopen که در stdio.h قرار دارد ، فایلی را باز می کنیم .

روش های باز کردن فایل :

۱- خواندن read : فایلی موجود را به منظور ورودی باز می کنیم.

۲- نوشتن write : فایلی را به منظور خروجی باز می کنیم .

۳- پیوست کردن append : فایلی را باز کرده و اطلاعات جدید را به آخر آن اضافه می کند .

نوع	معنی
w یا wt	یک فایل متنی جدید را به منظور نوشتن باز می کند
r یا rt	یک فایل متنی موجود را فقط برای خواندن باز می کند
a یا at	یک فایل متنی را به منظور اضافه کردن به آخر آن باز می کند
w+ یا w+t	یک فایل متنی جدید را برای خواندن و نوشتن باز می کند
r+ یا r+t	یک فایل متنی موجود را برای خواندن و نوشتن باز می کند
a+ یا a+t	یک فایل متنی را برای اضافه کردن به آخر آن یا خواندن باز می کند

اگر به جای t شما b بنویسید فایل باینری خواهد شد.

مثال :

```
FILE *fp;
fp=fopen("c:\\student.","w+");
```

فایل جدید student.txt را روی :c به منظور نوشتن و خواندن ایجاد می کند .

بستن فایل :

```
fclose(fp);
```

اگر یک فایل را به منظور نوشتن باز کنیم و اطلاعاتی را به آن اضافه کنیم ، موقعیت سنج به آخر فایل می رود . حال برای خواندن همان اطلاعات دو راه وجود دارد.

الف) فایل را ببندیم و دوباره بصورت خواندنی باز کنیم .

ب) با (rewind(fp موقعیت سنج را به ابتدای فایل منتقل کنیم .

دستورات خواندن و نوشتن اطلاعات در فایل :

الف) ()fprintf و ()fscanf : هر نوع فرمت از قبیل رشته ، کاراکتر ، عدد صحیح و اعشاری را در فایل نوشته و از آن می خواند.

ب) ()fwrite و ()fread : رکورد به رکورد در فایل نوشته و از آن می خواند .

ج) (آدرس فایل)getc ، (آدرس فایل,کاراکتر)fputs ، (آدرس فایل,کاراکتر)puts : کاراکتر به کاراکتر خوانده و در آن می نویسد.

مثال : فرض کنید فایل student.h به منظور خواندن و نوشتن باز شده است و آدرس آن در fp قرار گرفته است .

ch=getc(fp); یک کاراکتر را از فایل student.txt خوانده و در ch قرار می دهد.

put('A',fp); کاراکتر A را در فایل می نویسد.

د) ()fgets و ()fputs : رشته به رشته از فایل خوانده و در آن می نویسد.

مثال :

```
FILE *r ;
r=fopen(r,"ali.dat","r+w");
```

فایل باینری ali.dat موجود در مسیر جاری را بصورت خواندنی و نوشتنی باز می کند.

fprintf() :

```
fprintf(r,"x=%d and y=%f",15,13.5);
```

تابع fprintf() همان printf() است با این تفاوت که fprintf() بجای صفحه نمایش اطلاعات را در فایل می نویسد.

fscanf() :

```
fscanf(r,"%d%d",&a&b);
```

این دستور دو عدد صحیح را از فایل ali.dat خوانده و در متغییرهای a و b قرار می دهد.

تابع fscanf() همانند scanf() است با این تفاوت که fscanf() اطلاعات را از فایل و scanf() از صفحه کلید می خواند.

با توابع fprintf() و fscanf() می توان رکوردی را فیلد به فیلد نوشت یا خواند . در مواقعی که تعداد فیلدها زیاد باشد این توابع خسته کننده خواهد بود . برای ذخیره کامل یک رکورد در فایل از تابع fwrite() استفاده می کنیم .

fwrite() :

fwrite(آدرس فایل,تعداد کپی,اندازه بایت,آدرس متغییر ساختمان);

```
struct student
{
char name[10];
char family[20];
int num;
float avg;
}st;
fwrite(&st,sizeof(student),1,r);
```

آدرس ساختمان

تعداد کپی

این دستور محتوای رکورد st را یک بار در فایل ali.dat می نویسد.

مثال : کدام یک از دستورات زیر اشتباه است.

```
student st[10];
```

1) fwrite(stu,36,1,r);
2) fwrite(stu+3,36,1,r);
3) fwrite(&stu[5],36,1,r);
4) fwrite(stu[5],36,1,r);

جواب : گزینه ٤ درست نیست چون stu[5] آدرس نیست.

fread() :

همانند fwrite() است و یک رکورد را از فایل به متغییر ساختمان می خواند.

مثال : فرض کنید اندازه فایلی ۱۰۰ بایت باشد و تعریف کنیم

```
char s[500];
fread(s,300,1,fp);
```

یعنی ۳۰۰ بایت از فایل را به رشته s بخوان ، در آن صورت تا آخر فایل یعنی همه ۱۰۰ بایت خوانده می شود و خطا هم نمی دهد و اضافه هم نمی خواهد.

```
fgets(s,300,,fp);
```

۳۰۰ بایت از فایلی به آدرس fp خوانده و در رشته s قرار می دهد .

تغییر موقعیت اشاره گر فایل :

با تابع fseek() میتوان موقعیت اشاره گر فایل را حرت داد.

fseek(آدرس فایل,اندازه جابجایی به بایت,مبدأ);

موقعیت		
SEEK_SET	ابتدای فایل	0
SEEK_CUR	محل کنونی	1
SEEK_END	انتهای فایل	2

مثال :

```
fseek(fp,100,0);
```

موقعیت سنج را نسبت به ابتدای فایل ،۱۰۰ بایت به جلو می برد.

```
fseek(fp,-10,0);
```

موقعیت سنج را نسبت به محل کنونی ۱۰ بایت به عقب بر می گرداند یعنی به ۹۰ می رود.

```
x=ftell(fp);
```

موقعیت جاری مکان نمای فایل را بر می گرداند . یعنی x برابر ۹۰ می شود.

سوال تستی ۱ : اگر داشته باشیم :

```
struct u{
char name[20];
int n;
}r,t[30];
file *fp;
fp=fopen("test.txt","w+");
```

کدام دستور نادرست است.

```
1)fwrite(t[5],....
2)fwrite(t+2,....
3)fwrite(&r,....
4)fwrite(t,....
```

نتیجه : گزینه ۱ نادرست است ، چون اولین پارامتر تابع ()fwrite حتما باید آدرس ساختمان باشد.

دلایل باز نشدن فایل :

۱) فایل را برای خواندن باز کنیم در حالی که فایل موجود نباشد.

۲) فایل را روی درایوی باز کنیم که فضا ندارد.

اگر به هر دلیلی فایلی باز نگردد در آن صورت خروجی تابع()fopen ، NULL خواهد بود.

نکته :

اگر تعریف کنیم :

```
struct AA
{
int a;
float b;
}*x;
```

در این صورت x دو بایت فضا در RAM می گیرد و x فقط متغییر اشاره گر است پس رکورد نیست.

اما اگر بنویسیم :

```
x=(struct AA)malloc(sizeof(AA));
```

رکوردی از RAM اخذ و آدرس آن در x قرار خواهد گرفت.

```
x→a=15;
x→b=3.5;
x→a++;
x→b--;
printf("%d%f",x→a,x→b);
free(x);
```