سایت مرجع دانشجوی پیام نور

- ✓ نمونه سوالات پیام نور : بیش از ۱۱۰ هزار نمونه سوال همراه با پاسخنامه تستی و تشریحی
- ✓ کتاب ، جزوه و خلاصه دروس
- ✓ برنامه امتحانات
- ✓ منابع و لیست دروس هر ترم
- ✓ دانلود کاملا رایگان بیش از ۱۴۰ هزار فایل مختص دانشجویان پیام نور

اصول کامپیوتر ۲

رشته علوم کامپیوتر

٤ واحد درسی

فصل اول

نام منبع و مؤلف :

- ✓ اصول کامپیوتر ۲
- ✓ دکتر داود کریم زادگان مقدم
- ✓ انتشارات دانشگاه پیام نور ۱۳۸۳

فصل اول : مفاهیم اولیه

اهداف کلی و رفتاری

الگوریتم

نوع داده مجرد

بررسی نحوهٔ اجرای یک برنامه

پیچیدگی فضای لازم

پیچیدگی زمان

چرخه زندگی یک سیستم

طراحی شیء گرا

طراحی تابعی

هدف کلی

آشنایی با مفاهیم اولیه و ویژگی های کلی برنامه نویسی و طراحی نرم افزار

هدف های رفتاری

- تعریف الگوریتم و معیارهای آن
- شرح نوع داده مجرد (ADT)
- عوامل مؤثر در ارزیابی برنامه ها
- پیچیدگی فضا و زمان لازم برای یک برنامه
- چرخه زندگی یک سیستم
- ویژگیهای طراحی شیء گرا و طراحی تابعی

الگوریتم Algorithm

تعریف :

الگوریتم ، مجموعه محدود و پایان پذیر ا ز دستورالعمل ها است که اگر به صورت متوالی دنبال شونـد موجب انجام کـار خاصی می گردد.

تمام الگوریتم ها باید شرایط و معیارهای زیر را داشته باشند :

ورودی :

یک الگوریتم می تواند هیچ یا چندین کمیت ورودی داشته باشد که از محیط خارج تأمین می شود.

خروجی :

الگوریتم بایستی حداقل یک کمیت به عنوان خروجی ایجاد کند.

فرزاد فرزانفر دستیار آموزشی گروه کامپیوتر دانشگاه پیام نور

قطعیت :

هر دستورالعمل بایستی واضح و بدون ابهام باشد.

محدودیت :

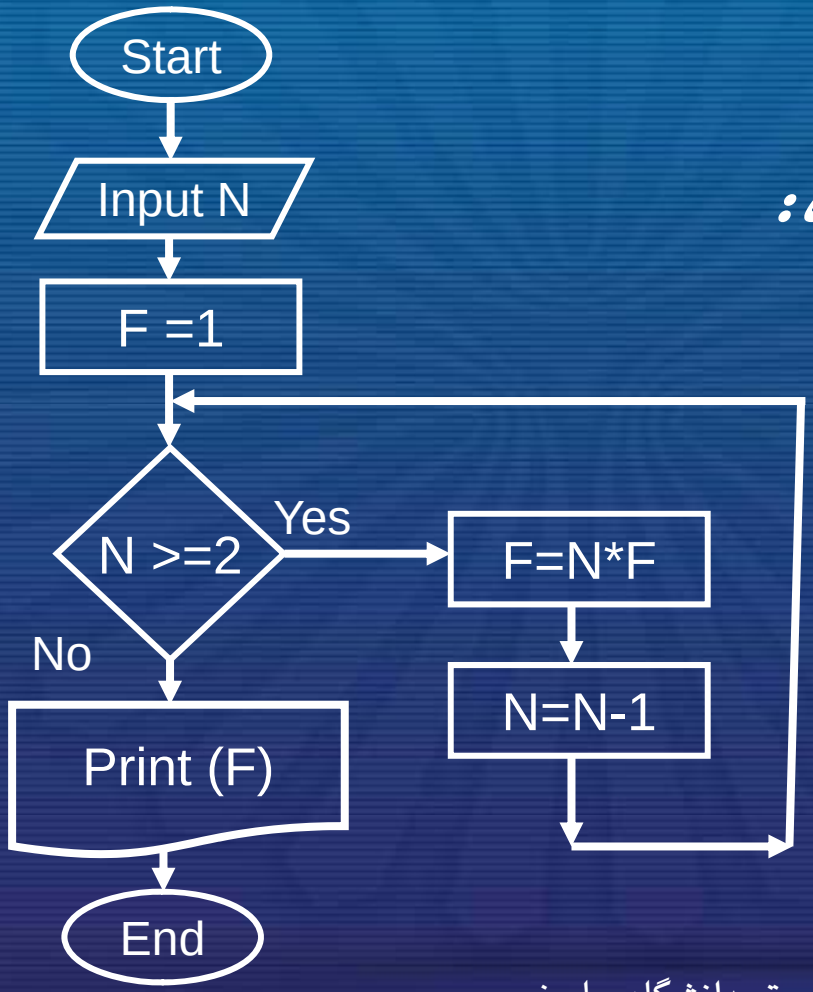
اگر ما دستورالعملهای یک الگوریتم را دنبال کنیم، برای تمام حالات ، الگوریتم باید پس از طی مراحل محدودی خاتمه یابد.

کارایی :

هر دستورالعمل باید به گونه ای باشد که با استفاده از قلم و کاغذ بتوان آن را با دست نیز اجرا کرد. در واقع تنها قطعیت کافی نیست، بلکه هر دستورالعمل نیز باید انجام پذیر باشد.

فلوچارت که نمودار گردشی نیز نامیده می شود یک روش دیگر است و معمولاً وقتی الگوریتم کوچک و ساده باشد به خوبی عمل می کند.

نمونه ای از یک فلو چارت ساده :



نوع داده مجرد Abstract Data Type

تعریف :

یک نـوع داده ، مفهوم مجـردی است که با مجموعـه ای از خواص منطقی تعریف شده است. نوع داده مجموعه ای از انـواع داده ی مقصـد (Object) و عملکردهـایی است که بر روی این نـوع داده ها عـمل می کنند.

نوع داده مجرد یا ADT وسیله مناسبـی برای مشـخص کردن خواص منطقی انواع داده هاست.

می توان توابع یک نوع داده را به چند گروه تقسیم و طبقه بندی کرد:

۱- ایجاد کننده/ سازنده

۲- تبدیل کنندگان

۳- مشاهده کنندگان/ گزارش کنندگان

معمولاً تعریف یک ADT حداقل شامل یک تابع از هرکدام از سه گروه بالا است.

نمونه ای از یک ADT ساده : (اعداد طبیعی)

Structure Natural-Number is

Objects: an ordered sub range of the integers starting at zero and ending at the maximum integer (INT_MAX) on the computer.

Functions:

for all x , y Є Nat-Number, True, False Є Boolean
And where +,-,< and == are the usual integer operations.

```
Nat-No Zero()              ::=  0
Boolean Is-Zero(x)         ::=  if (x) return False else return True
Nat-No Add(x,y)            ::=  if (( x + y ) <= INT_MAX) return x + y
                                else return INT_MAX
Boolean Equal(x,y)         ::=  if ( x == y ) return True else return False
Nat-No Successor(x)        ::=  if ( x == INT_MAX) return x
                                else return  x + 1
Nat-No Subtract(x,y)       ::=  if ( x < y ) return 0
                                else return x – y
```

End Natural-Number

بررسی نحوه اجرای یک برنامه

عوامل زیادی در ارزیابی برنامه ها مؤثرند ، از آن جمله :

- آیا برنامه اهداف اصلی کاری را که می خواهیم انجام می دهد؟
- آیا برنامه با همه مقادیر درست کار می کند؟
- آیا برنامه مستند سازی شده است تا نحوه استفاده و طرز کار با آن مشخص شود.

- آیا برنامه برای ایجاد واحدهای منطقی ، بطور مؤثـری از توابع استفاده می کند؟
- آیا کدهای برنامه ، خوانا است؟
- آیا برنامه از حافظه اصلی و کمکی بطور مؤثری استفاده میکند؟
- آیا زمان اجرای برنامه ی هدف، قابل قبول است؟

پیچیدگی فضای لازم

میزان حافظه یا پیچیدگی فضای یک برنامه، مقدار حافظه مورد نیاز برای اجرای کامل یک برنامه است. فضای مورد نیاز یک برنامه شامل موارد زیر است:

مثال ۱-۲ تابع abc سه متغیر ساده را به عنوان ورودی دریافت می کند و مقدار ساده ای را به عنوان خروجی باز می گرداند. این تابع فقط نیازمندیهای ثابت دارد بنابراین $S_{abc}(I)=0$.

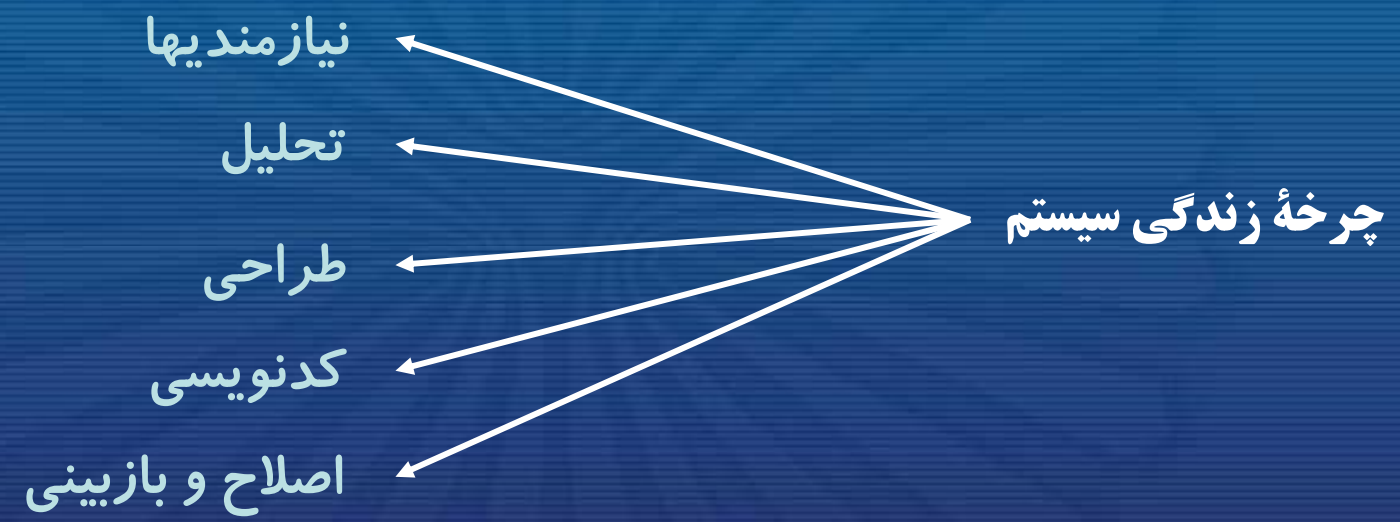
```
Float abc  ( float a, float b, float c )
{
  return   a + b + b * c + ( a + b + c ) / ( a + b ) + 4.00 ;
}
```

پیچیدگی زمان

میزان یا پیچیدگی زمان یک برنامه، مقدار زمانی است که کامپیوتر برای اجرای کامل برنامه لازم دارد.

چرخه زندگی یک سیستم

چرخهٔ زندگی سیستم یک فرآیند توسعه‌ای است که شامل موارد زیر است:



هرچند این مراحل را به‌صورت جداگانه درنظر می‌گیریم ولی هریک از آنها کاملاً به یکدیگر مرتبط هستند و هریک چهارچوب زمانی خاصی را دنبال می‌کنند .

۱- نیازمندیها :

ورودی و خروجی

تعریف نیازمندیهای نرم‌افزار، یک توصیف انتزاعی از سرویسهایی است که سیستم ارائه می‌کند و همچنین محدودیتهایی است که سیستم باید تحت آن شرایط عمل کند.

فرزاد فرزانفر دستیار آموزشی گروه کامپیوتر دانشگاه پیام نور

۲- تحلیل:

در این مرحله مسئله را به بخشهای قابل کنترل تقسیم می‌کنیم. در تحلیل یک سیستم دو شیـوه وجود دارد:

شیوه های تحلیل

- از پایین به بالا (bottom up)
- از بالا به پایین (top down)

۳- طراحی:

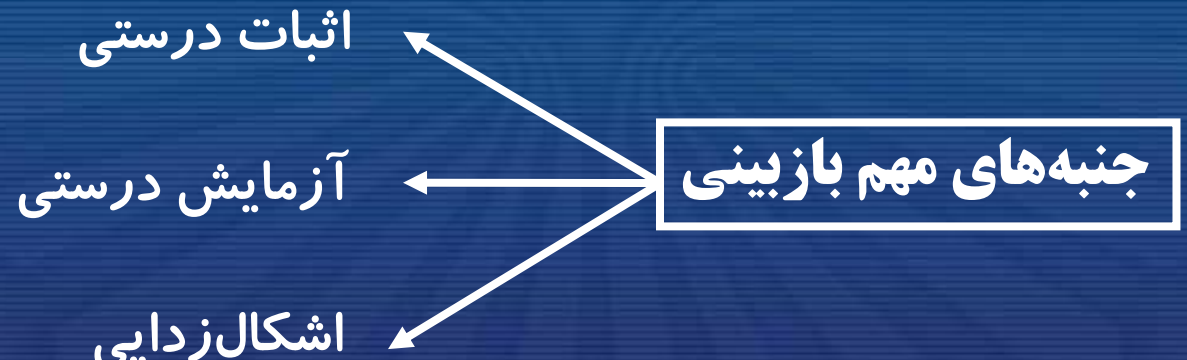
این مرحله، ادامه کاری است که در مرحله تحلیل انجام می‌شود. طراح، سیستم را هم از نقطه نظر داده‌های مقصود (data objects) مورد نیاز برنامه و هم از نظر اعمالی که بر روی آنها انجام می‌شود، بررسی می‌کند .

۴- پالایش(اصلاح) و کدنویسی :

در این مرحله ، نمایشی برای داده‌های مقصود خود انتخاب می‌کنیم و برای هر عملی که بر روی آنها انجام می‌شود ، الگوریتمهایی می‌نویسیم.

۵- بازبینی:

در این مرحلـــه درستی برنامه‌ها اثبات می‌شود و برنامه‌هـــا با انواع داده‌های ورودی مختلـــف آزمایش و خطاهای برنامه رفع می‌شود.



طراحی شیء گرا

طراحی شیءگرا بر مبنای پنهان‌سازی اطلاعات است.

در روش شیءگرا سیستم به‌صورت مجموعه‌ای از اشیا درنظر گرفته می‌شود.

ویژگیهای طراحی شیء گرا

- ناحیه مشترک داده ها حذف می شود.
- اشیاء براحتی قابل تغییر می باشند.
- اشیاء می توانند به صورت ترتیبی یا موازی اجرا شوند.

فرزاد فرزانفر دستیار آموزشی گروه کامپیوتر دانشگاه پیام نور

امتیازات طراحی شیء گرا

- نگهداری سیستمها آسانتر است.
- اشیاء میتوانند چندین بار و در چندین برنامه مورد استفاده قرار بگیرند.
- قابلیت درک سیستم افزایش می یابد.

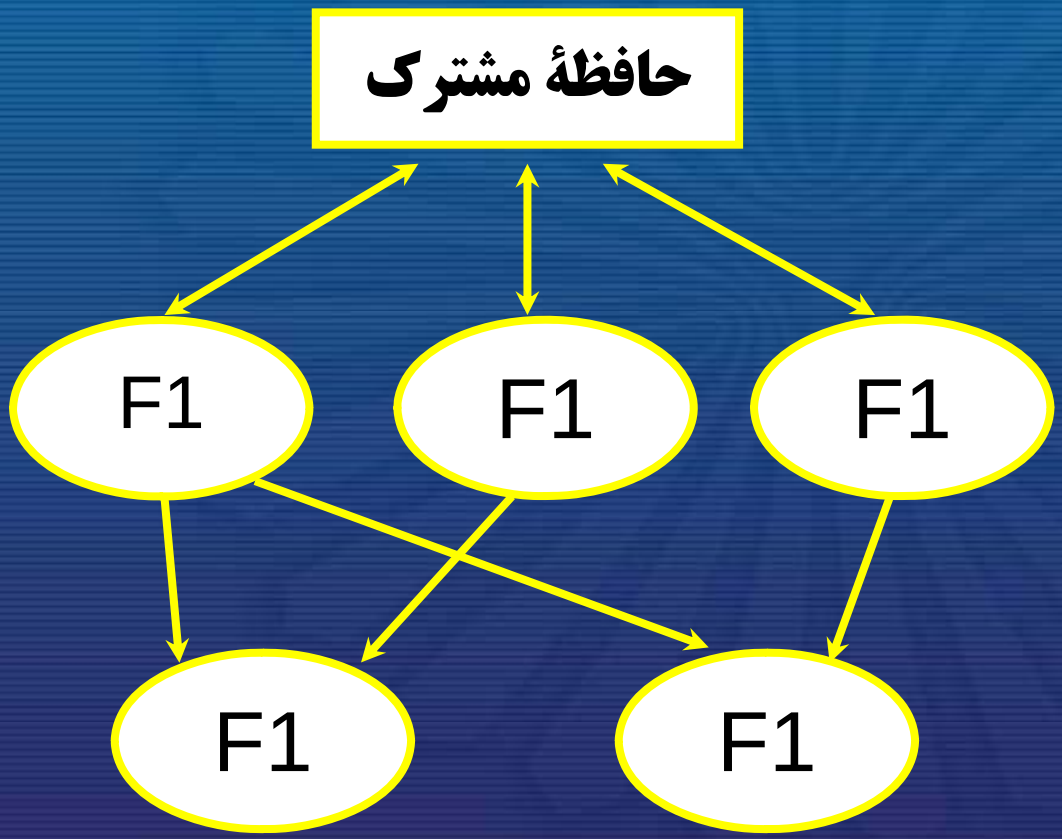
فرزاد فرزانفر دستیار آموزشی گروه کامپیوتر دانشگاه پیام نور

طراحی تابعی

یک روش طراحی تابعی ، طراحی ساخت‌یافته است .

این روش با استفاده از نمودارهای جریان داده‌ها، پردازش داده‌ای منطقی را توصیف می‌کند .

استراتژی طراحی تابعی، بر تجزیهٔ سیستم به مجموعه‌ای از توابع متأثر به هم تکیه دارد که حالت متمرکز سیستم، بین توابع مشترک است:



اصول کامپیوتر ۲

رشته علوم کامپیوتر

۴ واحد درسی

فصل دوم

نام منبع و مؤلف :

- ✓ اصول کامپیوتر ۲
- ✓ دکتر داود کریم زادگان مقدم
- ✓ انتشارات دانشگاه پیام نور ۱۳۸۳

فصل دوم : زبان برنامه نویسی

- عملگرهای محاسباتی
- عملگرهای انتساب
- عملگرهای یکانی
- عملگرهای رابطه ای (مقایسه ای)
- عملگرهای منطقی
- عملگر شرطی
- عملگرهای برنامه ها
- عملگرهای عملوندی اجرایی
- عملگرهای بیتی
- دستور

هدف کلی

آشنایی با مقدمات و اجزاء یک زبان برنامه نویسی پیشرفته

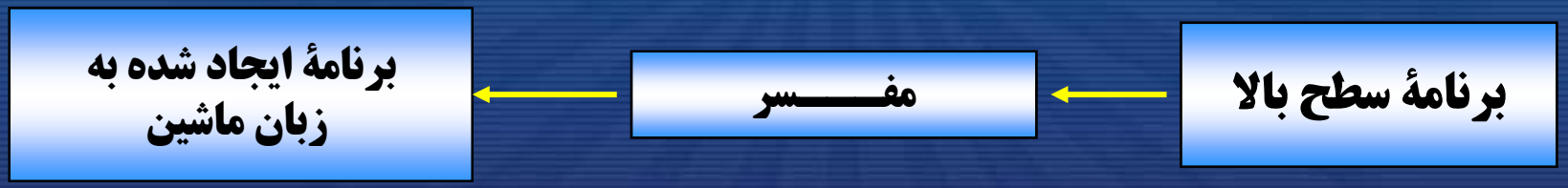
هدف های رفتاری

- تاریخچه و سیر تکاملی زبان C
- ویژگیهای مختلف زبان C
- کاراکترها ، شناسه ها و متغیرها در زبان C
- ساختار کلی زبان C
- انواع عبارت ، انواع دستور و انواع عملگرها و نحوه استفاده از آنها

مقدمه Introduction

- در اوایل دهه ۱۹۷۰ میلادی ، زبانC ، توسط دنیس ریچی و به عنوان زبان برنامه‌نویسی سیستم ها طراحی شد.
- این زبان از دو زبان قدیمی تر بنامهایBCPL و B حاصل شده است.
- زبانC تا سال ۱۹۷۸ منحصر به استفاده در لابراتوار کمپانیBELL بود تا اینکه توسط دو تن بنامهای ریچی و کرنیه نسخه نهایی این زبان منتشر شد

- آنچه در این اسلاید ارائه شده بر اساس استاندارد ANSI می باشد.
- مفسر خود برنامه‌ای کامپیوتری است که برنامهٔ سطح بالا، دادهٔ ورودی آن و برنامهٔ ایجاد شده به زبان ماشین، خروجی آن را تشکیل می‌دهد .



ANSI : American national standard institute

ویژگیهای مهم زبان C

- زبانC به طور گسترده‌ای در دسترس است .
- C زبانی است همه منظوره، ساخت‌یافته سطح بالا و انعطاف‌پذیر که برخی از خصوصیات زبانهای سطح پایین را نیز داراست .
- برنامه‌های نوشته شـــده به زبان C به طور کلی مستقـــل از ماشین یا نـوع کامپیوتر است و تقریباً تحت کنترل هر سیستم عاملی اجرا می‌شود.
- C روش برنامه‌سازی ماژولار را پشتیبانی می‌کند.

ویژگیهای مهم زبان C

- کامپایلرهای C فشرده و کم حجم‌اند و برنامه‌های هدف ایجاد شده با آنها خیلی کوچک و کارآمدند.
- برنامه‌هایC در مقایسه با سایر زبانهای برنامه‌سازی سطح بالا، به راحتی قابل انتقال‌اند .
- به طور کلی جامعیت، عمومیت، خوانایی، سادگی، کارآیی و پیمانه‌ای بودن که همگی از مشخصات برنامه‌ای ایده‌آل‌اند در زبانC پیاده‌سازی می‌شوند .

کاراکتر Character

زبان برنامه‌نویسی C مجموعه‌ای خاص از کاراکترها را شناسایی می‌کند.
این مجموعه عبارت‌اند از:

- **حروف بزرگ و حروف کوچک:** زبانC بین حروف بزرگ و کوچک تفاوت قائل می شود . (Case Sensitive)
- **ارقام دهدهی :** شامل ۰ تا ۹
- **جای خالی** یا Blank

- **کاراکترهای مخصوص :** شامل * \ / = - + ! @ # $ % ^ :
  & , . ; ? ( ) < > { } [ ]

- **کاراکترهای فرمت‌دادن:** که برای بیان کردن حالات ویژه ای بکار می روند و عبارتند از :

\t ,\v ,\n ,\b ,\f ,\r ,\0 ,...

شناسه Identifier

یک شناسه C دنباله‌ای است از حروف ، ارقام یا علامت زیر خط که با هر ترتیبی میتوانند قرار گیرند. اما اولین کاراکتر باید یک حرف باشد .

چند نمونه از شناسه های معتبر:

m1 , max , payam_noor , minimum

و شناسه های نامعتبر:

book-2 , 4s5 , $tax , "p" , computer science

متغیر Variable

- متغیرها در زبانC شناسه‌هایی هستند که محلهایی از حافظه را به خود اختصاص می‌دهند .
- در ساده ترین حالت یک متغیر جانشین یک قلم داده می گردد .
- تمامی متغیرهایی که در برنامه هایC بکار برده می شوند باید تعریف یا اعلان گردند .
- مقدار متغیر در طول اجرای برنامه می تواند تغییر کند .

- بعضی از شناسه‌های زبانC کلمات رزرو شده یا کلیدی هستند.
- متداولترین کلمات کلیدی زبان C در زیر نشان داده شده است :

main	int	float	char	if	else	goto	for
do	double	while	default	signed	return	register	enum
static	coutinue	short	case	auto	struct	const	sizeof
break	long	switch	void	typedef	extern	unsigned	union

توجه داشته باشید که همه کلمات کلیدی با حروف کوچک نوشته می‌شود

علامت توضیح Comment

- در زبانC هر عبارتی که بین دو علامت/* و */ قرار گیرد صرفاً بعنوان توضیحات محسوب می‌گردد .
- در اغلب نسخه ها علامت // هم مجاز است.

مثال ۲ـ۳ در برنامه زیر از علامت توضیح استفاده شده است :

```
#include<stdio.h>
main ( )
 {
        int  m , n ;                    /* declare variables */
        m = 1 ;                         /* initialize */
        while (m <= 10 )                /* outer loop */
        {
          printf("%5d ", m ) ;
          for ( n=1; n<=10 ; n+ +)      // inner loop
              printf("%5d", m * n ) ;
          printf("\n") ;
          m + + ;
        }
 }
```

ساختار برنامه ها

Program Structure

همه برنامه‌هایC شامل یک یا چندین تابع هستند که فقط یکی از آنها تابع اصلی یا main نامیده می‌شود.

```
main()
{
    variable declaration;

    program statements;
}
```

تابع اصلی

شروع تابع اصلی

تعریف متغیر ها

دستورات برنامه

پایان تابع اصلی

دستورالعمل های اجرایی Statements

- دستورالعمل هایی که از متغیر استفاده می کنند باید بعد از تعریف متغیرها درج شوند .
- در زبان C تمام دستورالعمل ها باید با سمیکلون(;) خاتمه یابند.

مثال ۵ـ۲ برنامه زیر طول و عرض مستطیلی را از طریق ورودی استاندارد خوانده و با فراخواندن تابعی به نام rectangle مساحت آن را محاسبه می کند.

```
# include <stdio . h>
void main ( )
{
   int a , b , area ;
   int  Rectangle (int a , int b) ;
   scanf ("%d %d" , &a , &b) ;
   area = Rectangle (a , b);
   printf ("\n length = %d  width = %d  area = %d" , a , b , area) ;
 }
int  Rectangle (int a , int b)
{
   int  s ;
   s = a * b ;
   return (s) ;
 }
```

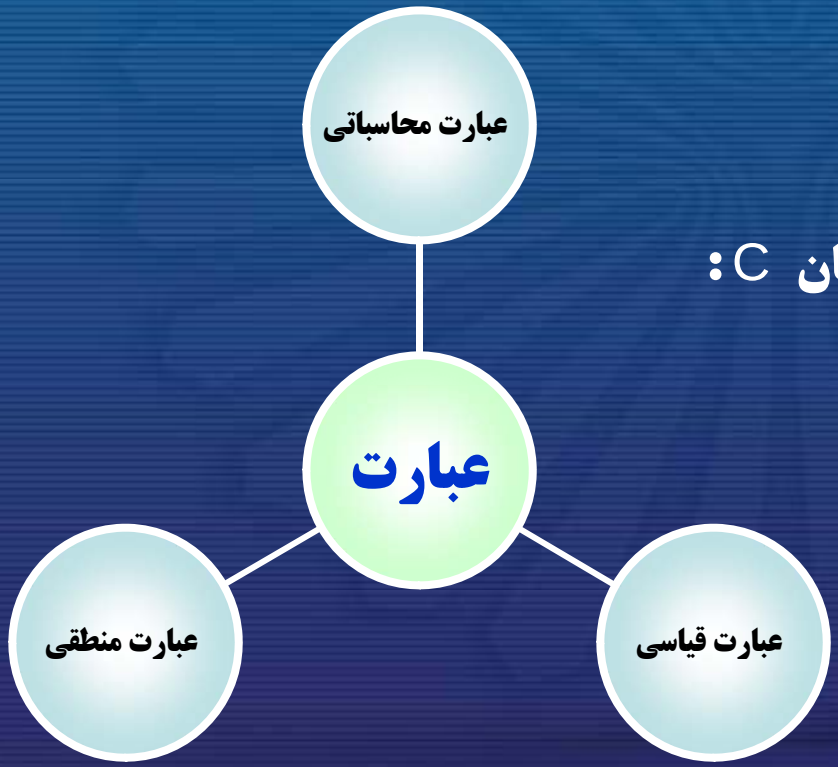
خروجی :

```
5  4

length = 5    width = 4    area = 20
```

عبارت Expression

- عبارت ، مجموعه ای معنی‌دار از داده‌ها (مقادیر عددی و متغیرها) است که با استفاده از نشانه ها یا عملگرهای محاسباتی ، قیاسی و منطقی با یکدیگر ترکیب شده‌اند .

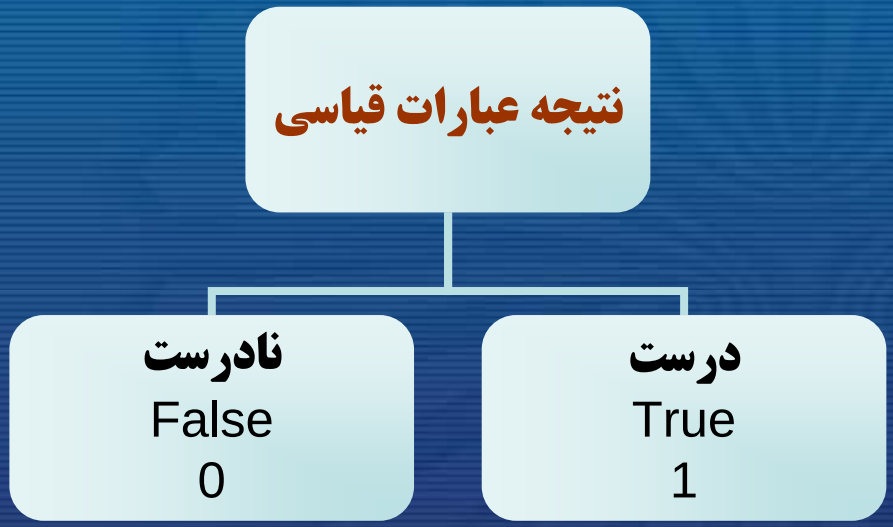
دسته بندی عبارات در زبان C :



عبارت محاسباتی: ترکیبی از مقادیر ثابت ، متغیرهای صحیح و اعشاری با استفاده از مجموعه عملگرهای محاسباتی است که تحت قاعده خاصی تشکیل می گردد:

عبارت ریاضی	معادل آن در زبان C
$\frac{ab}{c}$	a*b / c
$\frac{a}{b.c}$	a / (b*c)
$\frac{b^2 - 4ac}{2a}$	(b*b – 4*a*c) / (2*a)

عبارت قیاسی: عبارت قیاسی ترکیبی از عبارت های محاسباتی با استفاده از عملگرهای قیاسی و با رعایت قوانین مربوط به نحوه بکار بردن عملگرها است .



در عبارات قیاسی حق تقدم اجرا ، اول با عبارات محاسباتی می‌باشد و سپس عمل مقایسه مورد نظر انجام می‌گیرد

نمونه هایی از عبارات قیاسی

عبارت ریاضی	معادل آن در زبان C
a > 15	a >15
a + b - c ≤ 3.14	a+ b - c <= 3.14
3a + b ≠ 2	3*a + b != 2

اگر داشته باشیم: a=5 , b=1 , c=10 هر یک از عبارات فوق چه ارزشی خواهند داشت؟ درست یا نادرست؟

عبارت منطقی : عبارت منطقی مجموعه‌ای از عبارتهای محاسباتی و قیاسی است که در آن حداقل یک عملگر منطقی نیز بکار رفته است .

عبارت ریاضی	معادل آن در زبان C
5 > a > 3	(a>3) && (a<5) **یا** (a<5) && (a>3)
(a ≤ 3.14) U (a ≥ 15)	(a <= 3.14) || (a>=15)

- در اغلب زبانها ، عبارت های قیاسی که در طرفین یک عملگر منطقی قرار می‌گیرند، باید در داخل پرانتز محصور شوند .

دستور

Statement

دستور ، حکمی است که سبب می‌شود کامپیوتر ، عملی انجام دهد.

عملگرهای محاسباتی Arithmetical Operators

عملگرها یا اپراتورها، نشانه‌ها یی هستند که در عبارت ها بکار می‌روند و به کمک آنها می‌توان اعمالی را روی انواع داده انجام داد.

عملگر های محاسباتی

نام عملگر	نشانه	فرم	نوع عمل
جمع	+	a + b	a جمع با b
تفریق	-	a - b	a منهای b
منهای یکانی	-	-a	منهای a
جمع یکانی	+	+a	مقدار عملوند a

نوع عمل	فرم	نشانه	نام عملگر
a ضرب در b	a * b	*	ضرب
a تقسیم بر b	a / b	/	تقسیم
باقیمانده تقسیم a بر b	a % b	%	باقیمانده تقسیم
افزایش یک واحد به مقدار a	a++ , ++a	++	یک واحد افزایش
کاهش یک واحد از مقدار a	a-- , --a	--	یک واحد کاهش

چهار عملگر + , - , * , / می‌تواند تقریباً روی همه نوع داده‌های استاندارد موجود در زبان C بکار برده شود .

- دو عملگر ++ و -- در سایر زبانهای برنامه‌نویسی وجود ندارند . عملگر ++ یک واحد به عملوند خود اضافه می‌کند و عملگر -- یک واحد از عملوند خود کم می‌کند
- دو دستور : ++a ; و a++ ;
- معادل این دستور هستند : a = a + 1 ;
- همچنین این دو دستور : --a ; و a-- ;
- معادل این دستور هستند : a = a – 1 ;
- سرعت عمل دو عملگریکانی ++ و -- از سرعت عمل عملگر انتساب بالاتر است.

بسته به اینکه عملگرهای ++ و -- در عبارتهای محاسباتی قبل از متغیر قرار بگیرند یا بعد از آن عملکرد متفاوتی خواهند داشت؛ به مثالهای زیر توجه کنید:

```
Int x, y ;
x = 10 ;
y = ++ x ;
```

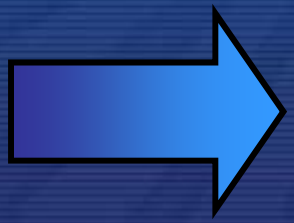
```
y = 11
x = 11
```

```
Int x, y ;
x = 10 ;
y =  x ++ ;
```

```
y = 10
x = 11
```

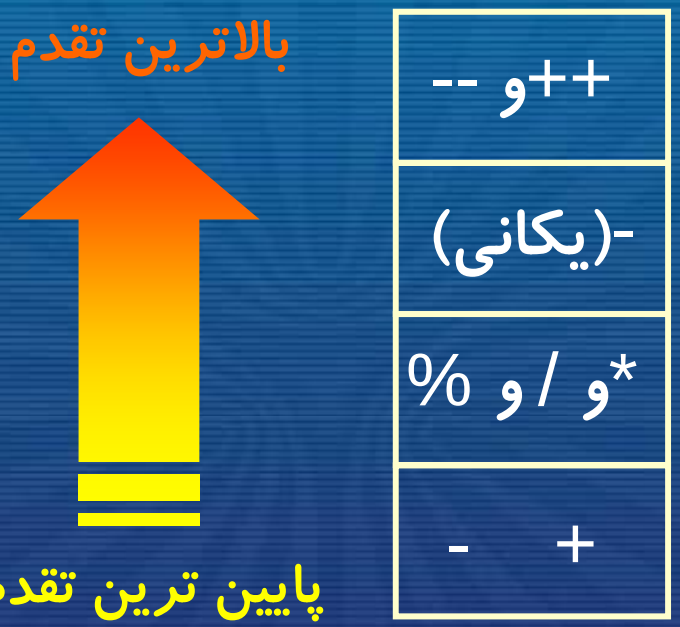
مقادیر x , y , m بعد از اجرای قطعه برنامه زیر چه خواهد بود؟

```
Int x, y, m ;
x = 10 ;
y = 15 ;
m = ++x + y++ ;
```



```
x = 11
y = 16
m = 26
```

ترتیب تقدم عملگرها

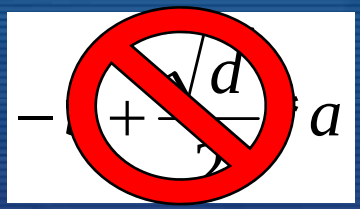


در مورد عملگرهای هم‌تقدم ترتیب تقدم از چپ به راست می‌باشد . درصورت وجود پرانتز ، تقدم پرانتز از تقدم همه عملگرها بالاتر است .

- عملگر پرانتز همیشه می تواند تقدم عملگرها را بهم بزند؛ عبارتی که داخل پرانتز قرارمیگیرد دارای بالاترین تقدم است.
- پس با عملگر پرانتز میتوان ترتیب اولویت محاسبه در عبارات را به دلخواه تغییر داد:

```
d = b*b – 4*a*c ;
x = -b + sqrt(d) / 2*a
```

عبارت ریاضی معادل ⟵



```
d = b*b – 4*a*c ;
x = (-b + sqrt(d)) / ( 2*a )
```

عبارت ریاضی معادل ⟵

$$\frac{-b+\sqrt{d}}{2*a}$$

عملگرهای انتساب Assignment Operators

در زبان C عملگر '=' عملگر انتساب است. این عملگر موجب می‌گردد که مقدار عملوند سمت راست آن ، در محل حافظه‌ای که با عملوند سمت چپ مشخص شده است ، قرار گیرد .

فرم کلی دستور انتساب به صورت زیر است :

identifier = expression ;

variable = expression ;

- در زبان C می‌توان دستورهای انتساب چندگانه بکار برد .

فرم کلی این گونه دستورهای انتساب به صورت زیر است:

V1 = V2 = ... = Vn = expression ;

- در چنین حالتی ، تقدم عمل انتساب از راست به چپ می‌باشد.

- در زبانC ، می‌توان عملگر انتساب را با عملگرهای محاسباتی ترکیب کرد:

+ = , -= , *= , /= , %=

عملگرهای یکانی Unary Operators

به عملگرهایی که در جلوی عملوند خود قرار می گیرند و فقط روی یک عملوند عمل می کنند عملگر یکانی گفته می شود.

عملگر sizeof

نام عملگر	نشانه (symbol)	فرم	نوع عمل
بزرگی (sizeof)	Sizeof	sizeof (t) sizeof x	بزرگی نوع داده t یا عبارت x را برحسب بایت برمی‌گرداند .

مثال ۲ـ۲۰ کاربرد عملگر sizeof :

```
float  a ;
printf ("%d ,%d", sizeof a , sizeof (float) ) ;
```

خروجی

4 ,4

عملگرهای رابطه‌ای (مقایسه‌ای) Relational Operators

نام عملگر	نشانه (symbol)	فرم	نتیجه
بزرگتر از	>	a > b	اگر a بزرگتر از b باشد ، نتیجه 1 وگرنه 0 است.
کوچکتر از	<	a < b	اگر a کوچکتر از b باشد ، نتیجه 1 وگرنه 0 است.
مساوی یا بزرگتر از	>=	a >= b	اگر a مساوی یا بزرگتر از b باشد ، نتیجه 1 وگرنه 0 است.
مساوی یا کوچکتر از	=<	a<=b	اگر a مساوی یا کوچکتر از b باشد ، نتیجه 1 وگرنه 0 است.
مساوی	= =	a = =b	اگر a مساوی b باشد ، نتیجه 1 وگرنه 0 است.
مخالف	!=	a!=b	اگر a مخالف b باشد ، نتیجه 1 وگرنه 0 است.

- ایده و مفهوم اصلی در مورد عملگرهای رابطه‌ای وابسته به مفهوم مقدار true و false است. در زبان C ، true هر مقدار غیر از صفر و false مقدار صفر است .

- **ترتیب تقدم عملگرهای رابطه‌ای**

بالاترین تقدم	> >= < <=
پایین ترین تقدم	= = !=

- تقدم عملگرهای محاسباتی، بالاتر از عملگرهای رابطه ای است.

عملگرهای منطقی Logical Operators

عملگر	علامت مفعول	فرم	نتیجه
و (AND) منطقی	&&	a&&b	اگر a و b هر دو برابر یک یا غیرصفر (true) باشند ، نتیجه برابر یک ، در غیراینصورت برابر صفر خواهد بود .
یا (OR) منطقی	||	a || b	اگر b یا a یا هر دو غیرصفر باشند نتیجه برابر با یک ، در غیراینصورت برابر صفر خواهد بود .
نفی یا نقیض (NOT) منطقی	!	!a	اگر a برابر صفر (false) باشد نتیجه برابر یک ، در غیر این‌صورت صفر خواهد بود .

در بین عملگرهای منطقی ، عملگر "!" بالاترین تقدم و عملگرد "||" پایینترین تقدم را دارد .

چند نمونه از کاربرد های عبارات منطقی و رابطه ای در برنامه های C :

```
while( (index != 10) && (i <= 5) )
{
   ..................
}

for ( i=n ; i >= 0 ; i--)
{
   ..................
}

if ( (counter == 0) || (flag != 1) )
{
   ..................
}
```

عملگر شرطی Condtional Operator

این عملگر دارای سه عملوند بوده و فرم کلی آن در جدول زیر نشان داده شده است.

عملگر	علامت	فرم	نحوه عمل
شرطی	?:	a ? b : c	اگر a غیرصفر باشد ، نتیجه b و گرنه نتیجه c است .

```
exp1 ?  exp2  :  exp3 ;
```

مثال ۲ـ۲۸ به قطعه برنامه زیر توجه کنید :

```
int   a =1 , b = 2 , c = 3 ;
c += ( a > 0  &&  a<= 10 ) ?  ++a  :  a/b ;
```

در اینجا مقدار عبارت شرطی برابر 2 می شود و با توجه به عملگر انتساب جمع داریم :

$$c = 3 + 2 = 5$$

معمولا این عملگر خوانایی برنامه را کاهش می‌دهد، لذا کمتر مورد استفاده است.

عملگر کاما Comma Operator

عملگر کاما "," این امکان را می‌دهد که چندین عمل در یک دستور انجام شوند .

عملگر	علامت	فرم	عمل
کاما	,	a , b ;	a ارزیابی می‌شود ، b ارزیابی می‌شود ، نتیجه b خواهد شد

```
variable = (statement1 , statement2);
```

مثال ۲ـ۲۹ عبارت مقابل را درنظر بگیرید :

```
a = (b=5 , b+15) ;
```

در عبارت مزبور ، ابتدا b برابر 5 قرار داده می‌شود و سپس عبارت b+15 محاسبه می‌گردد که نتیجه آن برابر20 خواهد بود و در پایان ، این مقدار به متغیر a نسبت داده می‌شود .

توجه کنید که ابتدا **عبارت ۱** اجرا می شود و **عبارت ۲** از نتیجه آن استفاده می کند.

عملگرهای حافظه Memory Operators

در زبانC چند عملگر وجود دارد که اجازه می‌دهند به خانه های حافظه و محتوای آنها دستیابی داشته باشید .

نام عملگر	علامت عملگر	مثال
آدرس	&	&a
محتوا	*	*a
عضو آرایه	[ ]	a[2]
نقطه	.	a.b
پیکان راست	->	p->a

هر یک از عملگرهای معرفی شده در بالا در فصول بعدی بطور مفصل تری مورد بحث قرار خواهند گرفت.

عملگرهای بیتی Bitwise Operators

عملگرهای بیتی برای تست کردن، مقدار دادن یا شیفت دادن و سایر اعمال بر روی مقادیری که در یک بایت (char) یا کلمه(int) ذخیره شده اند به کار می روند.

عملگر های بیتی را نمی توان با انواع float, double, long double و void یا سایر انواع پیچیده به کار برد.

عملگر بیتی	نام
&	و AND
|	یا OR
^	یای انحساری XOR
~	نقیض NOT
>>	شیفت به راست Right Shift
<<	شیفت به چپ Left Shift

عملکرد عملگرهای بیتی :

x	y	x & y	x | y	x ^ y	~x
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

مثال : به قطعه برنامه زیر توجه کنید(عملکرد عملگرهای شیفت) :

```
Unsigned char x;
   x=7;
   x = x << 1;
   x = x << 3;
   x = x << 2;
   x = x >> 1;
   x = x >> 2;
```

مقدار ده دهی x	مقدار دودویی x
7	00000111
14	00001110
112	01110000
192	11000000
96	01100000
24	00011000

مثال نمونه ای از عملکرد عملگرهای بیتی روی بایت ها:

```
11000001  &
01111111
--------
01000001
```

```
10000000  |
00000011
--------
10000011
```

```
01111111  ^
01111000
--------
00000111
```

```
01111111
~
--------
10000000
```

اصول کامپیوتر ۲

رشته علوم کامپیوتر

۴ واحد درسی

فصل سوم

نام منبع و مؤلف :

- ✓ اصول کامپیوتر ۲
- ✓ دکتر داود کریم زادگان مقدم
- ✓ انتشارات دانشگاه پیام نور ۱۳۸۳

فصل سوم : انواع داده ها

- مقدمه
- مقدار دهی اولیه متغیرها
- تعریف ثابت ها const
- نوع void صحیح
- پیشوند پرتابل صحیح
- دستورهای # include
- دستورهای # define

هدف کلی

آشنایی با انواع داده‌ها در زبان برنامه‌نویسی C و شیوه‌های معرفی آنها

هدف های رفتاری

- انواع داده های زبان C
- نحوه تعریف متغیرها و ثابت ها
- شیوه نمایش داده های صحیح ، اعشاری و کاراکتری
- مقادیر ثابت رشته ای
- مقداردهی اولیه متغیرها
- عملگر cast و نحوه کاربرد آن
- داده‌های از نوع void
- نحوه تعریف و کاربرد پیش پردازنده

مقدمه Introduction

زبان C مجموعه کاملی از انواع داده‌ها را پشتبانی می‌کند.

انواع داده ها

- مقادیر ثابت
- مقادیر متغیر
- داده‌های از نوع کاراکتر character
- داده‌های از نوع صحیح integer
- داده‌های از نوع اعشاری floating point

انواع داده های اسکالر

انواع داده حسابی
arithmetic data type

- داده‌های از نوع کاراکتر
  character
- داده‌های از نوع اعشاری
  floating point
- داده‌های از نوع صحیح
  integer
- اشاره گر
  pointer
- نوع شمارشی
  enumerated

انواع داده های غیر اسکالر(مجموعه ای)

- آرایه
- رکورد
- ساختار
- اجتماع

اعلان متغیرها Variable Declaration

یک تعریف یا اعلان، گروهی از متغیرها را به نوع داده خاصی مربوط می سازد.

```
int  a , b , c ;
```

کلمه رزرو شده **int** داده‌هایی از نوع صحیح را مشخص می‌کند.

کلمات کلیدی برای تعریف متغیرها

اصلی	اصلاح کننده
int float char double enum	short long signed unsigned

داده های صحیح

Integer Type

در هنگام تعریف متغیرهای از نوع int ،توصیف‌کننده‌های زیر به کار می روند:

short , long , signed , unsigned

ترکیبی از آنها نیز ممکن است بکار برده شود . به مثالهای زیر توجه کنید:

```
long  int  temp , Pnoor ;
short  int  y1 , y2 , y3 ;
unsigned  int   m , n ;
unsigned  long  sum , average ;
unsigned  short  pt , qt ;
```

مقادیر ثابت صحیح Constant Integer Type

یک مقدار ثابت صحیح عدد و یا دنباله‌ای از ارقام است که میتواند در مبنای ۸ ، مبنای ۱۰ و یا مبنای ۱۶ تعریف شده باشد.

برای تعریف ثابت های صحیح در مبنای ۱۶ یا ۸ از پیشوندهای 0 برای مبنای ۸ و 0x برای مبنای ۱۶ استفاده می‌شود .

0x1df3, 0xcb2, 0457, 036

داده‌های اعشاری Floating-point Type

داده های اعشاری به دو صورت نمایش داده می شوند:

۱– ساده

0.996 , 15.0 , 3.1415 , 7. , .275

۲– ممیز شناور (نماد علمی) :

3E2 به مفهوم $3*10^2$ و -125.7E-3 به مفهوم $-125.7*10^{-3}$ می‌باشد.

- برای اعلان متغیرهایی از نوع floating point از دو کلمه کلیدی "float" و "double" استفاده می‌شود مانند:

```
float   a , b , c  ;
double   x , y , z  ;
```

- یک متغیر توصیف شده با"float" به‌طور متعارف ۴ بایت حافظه اشغال می‌کند و متغیر از نوع "double" ۸ بایت.

داده های کاراکتری Character Type

کاراکتر یک مقدار صحیح یک بایتی است که می‌تواند هم برای نگهداری اعداد و هم برای نگهداری کاراکترها بکار برده شود .

ثابتهای حرفی در داخل یک زوج گیومه قرار می‌گیرد.

```
char  a , b ;
b = 0 ;
a = '0';
```

مقدار a برابر ۴۸ ، یعنی برابر اسکی کد کاراکتر '0' و مقدار b برابر عدد صفر خواهد بود .

در مجموعه کدهای اسکی ، کد کاراکترها دارای ترتیبی براساس همان ترتیب کاراکترها هستند :

ASCII Value	Character	ASCII Value	Character
065	A	081	Q
066	B	082	R
067	C	083	S
068	D	084	T
069	E	085	U
070	F	086	V
071	G	087	W
072	H	088	X
073	I	089	Y
074	J	090	Z
075	K	091	[
076	L	092	\
077	M	093	]
078	N	094	□
079	O	095	_
080	P	096	`

ASCII Value	Character	ASCII Value	Character
097	a	113	q
098	b	114	r
099	c	115	s
100	d	116	t
101	e	117	u
102	f	118	v
103	g	119	w
104	h	120	x
105	i	121	y
106	j	122	z
107	k	123	{
108	l	124	|
109	m	125	}
110	n	126	~
111	o	127	DEL
112	p		

جدول انواع داده‌ها در زبان C

نوع داده	اندازه به بیت	محدوده قابل قبول
char	۸	۱۲۸- تا ۱۲۷
unsigned char	۸	۰ تا ۲۵۵
signed char	۸	۱۲۸- تا ۱۲۷
int	۱۶	۳۲۷۶۸- تا ۳۲۷۶۷
short int	۱۶	۳۲۷۶۸- تا ۳۲۷۶۷
signed int	۱۶	۳۲۷۶۸- تا ۳۲۷۶۷
signed short int	۱۶	۳۲۷۶۸- تا ۳۲۷۶۷
unsigned int	۱۶	۰ تا ۶۵۵۳۵
unsigned short int	۱۶	۰ تا ۶۵۵۳۵
long int	۳۲	۲،۱۴۷،۴۸۳،۶۴۷ تا ۲،۱۴۷،۴۸۳،۶۴۸-
signed long int	۳۲	۲،۱۴۷،۴۸۳،۶۴۷ تا ۲،۱۴۷،۴۸۳،۶۴۷-
unsigned long int	۳۲	۰ تا ۴,۲۹۴,۹۶۷,۲۹۵
float	۳۲	$۴/۳ \times ۱۰^{-۳۸}$ تا $۴/۳ \times ۱۰^{+۳۸}$
double	۶۴	$۷/۱ \times ۱۰^{-۳۰۸}$ تا $۷/۱ \times ۱۰^{+۳۰۸}$
long double	۸۰	$۴/۳ \times ۱۰^{-۴۹۳۲}$ تا $۱/۱ \times ۱۰^{+۴۹۳۲}$

منبع : Borland C++ 3.2 / Help

مقادیرثابت رشته‌ای Constant String Type

یک رشته یا ثابت رشته‌ای ، از تعدادی کاراکتر متوالی تشکیل شده که بین دو گیومه قرار می گیرند.

"university" , "256" , " payam noor" , "1380-02-06" , "five$" , "p4"

- کامپایلر بطور اتوماتیک یک کاراکتر null (\0) در پایان هر ثابت رشته‌ای قرار می‌دهد که آخرین کاراکتر در داخل رشته (قبل از بسته شدن گیومه) خواهد بود.

“computer” → | c | o | m | p | u | t | e | r | \0 |

- یک ثابت حرفی مانند 'P' با یک ثابت رشته‌ای تک حرفی متناظر آن مانند "P" هم‌ارز نمی‌باشد .

- در واقع یک رشته تک حرفی متشکل از دو کاراکتر می‌باشد که کاراکتر دوم همان کاراکتر null است که پایان رشته را مشخص می‌سازد .

char ch='P';

char ch[2]="P";

مقداردهی اولیه متغیرها Variable Initialization

میتوانیم به هنگام تعریف متغیر یک مقدار اولیه به آن اختصاص دهیم.

برای این کار در تعریف متغیرها، به دنبال نام آن، اپراتور جایگزینی '=' را همراه با مقدار اولیه بکار می‌بریم.

- برای جلوگیری از اشتباه بهتر است متغیرهایی که مقدار اولیه می‌پذیرند، جدا از سایر متغیرها توصیف گردند، مانند مثال زیر:

```
int  a =12 , b =13 , c =25 ;
int  d , e , f ;
```

عملگر cast

cast Operator

با این عملگر می‌توان تبدیل یک نوع به نوع دیگر را به صورت صریح انجام داد .

برای این کار کافی است نوع جدید داده مورد نظر را در داخل پرانتز مستقیماً جلوی عبارت قرار دهیم.

(data type) expression

مثال: نتیجه اعمال عملگر cast :

```
int k=3, i=2 ;
float h ;
h = k / i ;
```

h = 1.0

```
int k=3, i=2 ;
float h ;
h =(float) k / i ;
```

h = 1.5

نوع void

void Type

داده از نوع void (تهی) در استاندارد ریچی وجود نداشت و بعد به استاندارد ANSI افزوده شده است .

هدف از معرفی این نوع داده معرفی توابعی است که مقداری را برنمی‌گردانند، بلکه فقط عمل خاصی را انجام می‌دهند.

مثال ۳ـ۸ تابعی به نام Pnoor که دارای دو آرگون x , y است به صورت زیر تعریف شده است:

```
void  Pnoor( x , y )
int  x , y ;
 {
  ------------
  ------------
  ------------
 }
```

قرار گرفتن void در جلوی نام تابع به این دلیل است که این تابع چیزی را به عنوان خروجی برنمی‌گرداند .

پیش پردازنده Preprocessor

- هنگامی که شما یک برنامه را کامپایل می‌کنید، پیش‌پردازنده بطور اتوماتیک اجرا می‌گردد .
- تمام فرامین پیش‌پردازنده با علامت " # " شروع می‌گردند که باید اولین کاراکتر خط باشد.
- وظیفهٔ اصلی و مهم پیش‌پردازنده آن است که فایل درخواستی را آماده ساخته و وارد برنامه کند.
- برخلاف دستورات C که به سمیکلون ختم می‌شوند، پایان دستورات پیش پردازنده با خط جدید مشخص می‌گردد .

دستور # include

include Statement

- دستور #include <filename> باعث می شود محتویات filename در فایل مبنا (فایل شامل دستور #include) وارد شود.
- استفاده از این دستور به ویژه در حالتی که بیش از یک فایل مبنا، اطلاعاتی یکسان را سهیم شوند و بکار ببرند، مفید است .
- تعریف دستور #include دارای دو فرم زیر است :

```
# include < filename >
```

```
# include "filename"
```

بعد از دستور #include سمیکلون قرار نمی گیرد.

- وقتی کامپایلر به دستور #include می رسد محتویات فایل مشخص شده در دستور را با دستور #include داخل فایل مبنا جایگزین می کند. به فایل هایی که می توانند در دستور #include قرار بگیرند فایل سرآیند(header file) گفته می شود.
- پسوند فایل های سرآیند معمولا h. است.
- فایل سرآیند می تواند شامل تعاریف متغیرها، ثوابت و تعاریف و بدنه توابع باشد.

دستور # define

define Statement

✦ **فرم کلی دستور:**

```
#define  constant_name  constant_value
```

اگر بخواهیم به یک مقدار ثابت اسم خاصی را نسبت دهیم و در داخل برنامه به جای مقدار ثابت از اسم آن استفاده کنیم ؛ از این دستور استفاده می کنیم.

به ثوابتی که بدین طریق تعریف می شوند ثوابت سمبلیکی گفته می شود.

در هنگام کامپایل تمام اسامی بکار رفته در برنامه با مقدار ثابت سمبلیک متناظر جایگزین می شود.

مزایای استفاده از دستور # define

- به بعضی مقادیر ثابت می‌توان اسم با معنی اختصاص داد.

```
#define  Pi  3.1415
```

- اگر مجبور باشیم در یک برنامه یک مقدار ثابت طولانی نامانوس را چندین بار بکار بریم، میتوانیم با دستور#define یک نام مناسب برای آن انتخاب کنیم و بجای ثابت مزبور از آن نام استفاده کنیم .

- تغییر در مقادیر ثابت برنامه های طولانی براحتی با تغییر آن مقدار ثابت در یک دستور (دستور#define) امکانپذیر خواهد بود.

اصول کامپیوتر ۲

رشته علوم کامپیوتر
۴ واحد درسی
فصل چهارم

نام منبع و مؤلف :

- ✓ اصول کامپیوتر ۲
- ✓ دکتر داود کریم زادگان مقدم
- ✓ انتشارات دانشگاه پیام نور ۱۳۸۳

فصل چهارم : توابع ورودی و خروجی

اهداف کلی و رفتاری

مقدمه

تابع printf()

تابع scanf()

تابع getchar()

تابع putchar()

تابع getche()

تابع getch() و putch()

توابع gets() و puts()

هدف کلی

آشنایی با چند تابع ورودی و چند تابع خروجی در زبان C

هدف های رفتاری

- ویژگی و کاربرد تابع خروجی printf()
- ویژگی و کاربرد تابع ورودی getchar()
- ویژگی و کاربرد تابع خروجی putchar()
- ویژگی و کاربرد تابع ورودی getch()
- ویژگی و کاربرد تابع خروجی puts()
- ویژگی و کاربرد تابع ورودی gets()
- تفاوت و تشابه توابع ورودی scanf() و printf()

مقدمه Introduction

- در این فصل برخی از توابع کتابخانه ای ورودی و خروجی متداول که اغلب در کتابخانه یا فایل"stdio.h" قرار دارند مورد بررسی قرار می گیرند .
- توابع فرمت دار scanfو printf اجازه انتقال اطلاعات میان کامپیوتر و دستگاههای ورودی و خروجی استاندارد را می‌دهند .
- دو تابع getchar و putcharموجب انتقال یک کاراکتر به حافظه و بالعکس می گردد .

تابع printf( )

- فرم کلی تابع printf() به صورت زیر است :

```
printf ("control string", arguments list) ;
```

و یا :

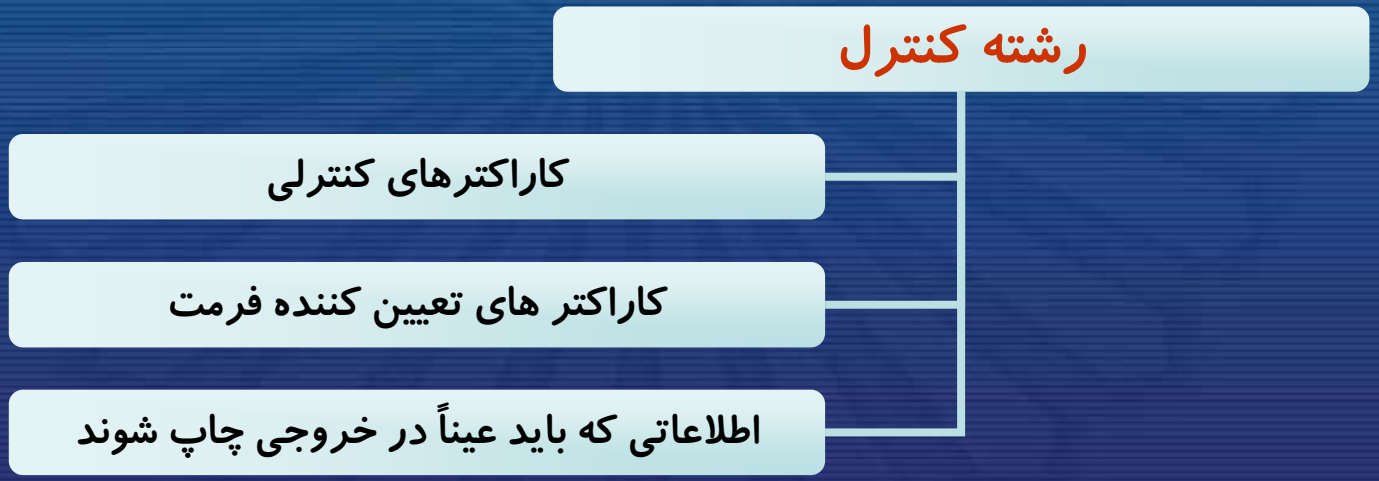
```
printf ("control string", arg1, arg2,…, argn) ;
```

- این تابع داده های خروجی را از کامپیوتر به دستگاه خروجی استاندارد مثل مانیتور می فرستد.

- این تابع می‌تواند به تعداد دلخواه آرگومان بپذیرد که آرگومان اول رشته فرمت و یا رشته کنترل نامیده می‌شود .

- رشته کنترل در داخل زوج گیومه قرار می‌گیرد و شامل اطلاعات قالب بندی است.

- رشته کنترل می تواند شامل موارد زیر باشد:



کاراکتر های تعیین کننده فرمت برای تابع printf ( )

کد تبدیل	مفهوم فرمان فرمت
%c	داده به صورت تک‌کاراکتر نمایش داده می‌شود.
%d	داده به صورت عدد صحیح دهدهی علامت‌دار نمایش داده می‌شود.
%i	داده به صورت عدد صحیح علامت‌دار نمایش داده می‌شود.
%f	داده به صورت عدد اعشاری، ممیز شناور بدون نما یا توان نمایش داده می‌شود.
%e, %E	داده به صورت floating-point ولی به فرم نمایی یا علمی نمایش داده می‌شود.
%g, %G	داده به صورت floating-point به فرم %f یا %e (هرکدام کوتاهتر باشد) نمایش داده می‌شود.
%u	داده به صورت عدد صحیح دهدهی بدون علامت نمایش داده می‌شود.
%s	داده به صورت رشته نمایش داده می‌شود.
%o	داده به صورت عدد صحیح در مبنای ۸ نمایش داده می‌شود.
%x, %X	داده به صورت عدد صحیح در مبنای ۱۶ و بدون علامت، نمایش داده می‌شود.
%l	پیشوندی که با %d , %u , %x , %o برای معرفی مقدار صحیح بلند یا طولانی بکار برده می‌شود. (مثل ld)
%p	در مورد داده های از نوع اشاره‌گر بکار برده می‌شود .
%%	یک علامت % چاپ می‌کند .

کاراکتر های کنترلی برای تابع printf()

کاراکتر	عملی که انجام می دهد
\f	موجب انتقال کنترل به صفحه جدید می شود
\n	موجب انتقال کنترل به خط جدید می شود
\t	انتقال به ۸ محل بعدی صفحه نمایش
\”	چاپ کوتیشن(“)
\’	چاپ تک کوتیشن (‘)
\0	NULL(رشته تهی)
\\	Back slash
\v	انتقال به ۸ سطر بعدی
\N	ثابت های مبنای ۸(N عدد مبنای ۸ است)
\xN	ثابت های مبنای ۱۶ (N عدد مبنای ۱۶ است)

مثال : برنامه زیر داده های از نوع مختلف را با فرمت های متفاوت در خروجی چاپ می کند.

```
#include <stdio.h>
int main()
{
  int x= 10;
  float y= 15.5;
  char ch= ‘a’;
  printf(“\n x= %d, y= %f, ch= %c”,x,y,ch);
  printf(“\n address of x is : %p”,&x);
  return 0;
}
```

خروجی

```
x = 10, y= 15.500000, ch= a
address of x is : FFF4
```

کاراکتر فرمت %n

- دستور printf() مانند بیشتر توابع کتابخانه ای c خروجی تولید میکند.
- اگر تابع با موفقیت اجرا شود تعداد کاراکتر های چاپ شده را بر می گرداند و در صورتی که با شکست مواجه شود یک مقدار منفی بر می گرداند.
- با کاراکتر کنترلی %n می توان تعداد کاراکترهای چاپ شده پیش از این کاراکتر کنترلی را به یک متغیر نسبت داد:

```
#include <stdio.h>
void main()
{
  int count;
  printf("This is a %n test statement" , &count);
  printf("\n count = %d ",count);
}
```

خروجی

```
This is a test statement
count = 10
```

- آرگومان های تابع printf() می تواند عبارت محاسباتی یا هر عبارت دیگری باشد.
- در این صورت مقدار عددی عبارت محاسبه شده، و با فرمت خواسته شده در رشته کنترلی چاپ می شود.

به برنامه زیر توجه کنید :

```
#include<stdio.h>
main ( )
 {
  double  x=50.0 , y=0.25 ;
  printf(" %f ,%f ,%f ,%f\n ", x , y , x*y , x / y) ;
  printf(" %e ,%e ,%e ,%e ", x , y , x*y , x / y) ;
 }
```

خروجی

```
50.000000  ,0.250000  ,12.500000  ,200.000000
5.000000e+01  ,2.500000e-01  ,1.250000e+01  ,2.000000e+02
```

- ملاحظه کردید که هر مقدار floating point تا ۶ رقم دقت، پس از نقطه اعشار نمایش داده شده است. این تعداد ارقام را می‌توان، با قرار دادن میزان دقت (طول میدان)در رشته کنترلی تغییر داد.

- حداقل طول میدان را می‌توان با قرار دادن یک عدد صحیح (به عنوان مشخص کننده حداقل فضای لازم) بین علامت% و کد فرمت مشخص کرد.

```
printf("%5d , %06d",125,85);
```

خروجی

```
125 , 000085
```

در حالت عادی اطلاعات رشته‌ای از طرف چپ و مقادیر عددی از سمت راست میدان تراز می‌شوند.

در مورد مقادیر floating point برای مشخص ساختن تعداد ارقام بعد از ممیز، باید پس از عدد مشخص‌کننده طول میدان، علامت ممیز " . " پس از آن نیز یک عدد که معرف تعداد ارقام اعشار خواهد بود، قرار داد:

```
#include<stdio.h>
main ( )
  {
    float  x =123.456 ;
    printf("%7f %7.3f %7.1f\n", x , x , x) ;
    printf("%12e %12.5e %12.3e", x , x , x) ;
   }
```

خروجی

```
123.456001 123.456   123.5
1.234560e+02  1.23456e+02    1.235e+02
```

تابع scanf( )

فرم کلی تابع scanf() :

```
scanf ("control string", arguments list) ;
```

و یا :

```
scanf ("control string", argl , arg2 ,..., arg n) ;
```

- این تابع همانند printf() یک تابع کتابخانه ای فرمت دار C است.
- این تابع داده ها را از یک دستگاه ورودی (معمولا صفحه کلید) دریافت کرده و در حافظه کامپیوتر قرار می دهد.
- به کمک این تابع می‌توان داده‌های عددی، کاراکترها، رشته‌ها و یا ترکیبی از آنها را وارد کامپیوتر کرد.

- این تابع می تواند هر تعداد آرگومان داشته باشد.
- این تابع هم همانند printf() در رشته کنترلی اش از کاراکتر های فرمت استفاده می کند.
- این تابع هم خروجی دارد ، اگر تابع با موفقیت اجرا شود تعداد متغیر هایی را که از ورودی خوانده بر می گرداند وگرنه کاراکتر EOF را برمی گرداند.
- تفاوت مهم بین این تابع و printf() آن است که در جلوی آرگومانها ، اپراتور آدرس یعنی"&" نیز قرار می‌گیرد .

مثال : قطعه برنامه زیر یک مقدار صحیح از ورودی خوانده و آن را چاپ می کند:

```
int x ;
scanf(“%d”,&x);
printf(“x = %d”);
```

خروجی

```
10 ↵
x = 10
```

برای خواندن متغیرهای رشته ای از ورودی نیازی به کاراکتر آدرس “&” نیست؛ علت این تفاوت در فصل ۸ بررسی خواهد شد.

```
#include<stdio.h>
main ( )
  {
    int  x ;
    char name[6] ;
    scanf("%d" , &x ) ;
    scanf("%s", name) ;
    printf("%s %d", name, x ) ;
  }
```

خروجی

```
10 ↵
hello ↵
hello 10
```

کاراکترهای فرمت در تابع scanf ( )

کد فرمت	شـــــرح
%c	داده ورودی به صورت تک‌کاراکتر تعبیر می‌شود.
%d	داده ورودی به صورت عدد صحیح علامت‌دار (در مبنای ۱۰) تعبیر می‌شود.
%i	داده ورودی به صورت عدد صحیح علامت‌دار تعبیر می‌شود.
%u	داده ورودی به صورت عدد صحیح بدون علامت دهدهی تعبیر می‌شود.
%f , %e, %g	داده ورودی به صورت عدد صحیح اعشاری با ممیز شناور (floating_point) تعبیر می‌شود.
%h	داده ورودی به صورت عدد صحیح کوتاه (short integer) تعبیر می‌شود.
%s	داده به صورت یک رشته تعبیر می‌گردد. ورودی به‌وسیلهٔ یک کاراکتر non_white_space آغاز می‌گردد و با اولین کاراکتر white_spaceخاتمه می‌پذیرد (به پایان رشته بطور خودکار کاراکتر "\0" افزوده خواهد شد.)
%0	داده ورودی به صورت عدد صحیح در مبنای ۸ تعبیر می‌گردد.
%x , %X	داده ورودی به صورت عدد صحیح در مبنای ۱۶ تعبیر می‌گردد.
%p	داده ورودی به عنوان یک اشاره‌گر تعبیر می‌گردد.

- هنگام خواندن رشته به وسیله تابع scanf() می توانیم کاراکترهای مجاز و غیرمجاز رشته ورودی را تعیین کنیم. به مثال زیر توجه کنید:

```
#include<stdio.h>
main ( )
  {
    char  line[80] ;
    ..........
    scanf("%[ABCDEFGHIJKLMNOPQRSTUVWXYZ]", line) ;
    ..........
  }
```

- عملگر کروشه در رشته کنترلی scanf() مشخص کننده کاراکترهای مجاز است.
- برنامه بالا تا زمانی به خواندن اطلاعات ادامه خواهد داد که کاربر کاراکتر مجاز (در اینجا حروف بزرگ) وارد کند و رشته نیز پر نشده باشد.

- عملگر "^" (circumflex) در رشته کنترلی scanf() مشخص کننده کاراکترهای غیر مجاز است.

```
#include<stdio.h>
main ( )
  {
     char  line[80] ;
     ..........
     ..........
     scanf("%[^0]", line) ;
     ..........
     ..........
  }
```

- برنامه بالا به خواندن اطلاعات زمانی خاتمه خواهد داد که کاربر کاراکتر 0 را وارد کند. یعنی اگر رشته ورودی computer sci0ense باشد فقط رشته computer sci در رشته line وارد خواهد شد و از بقیه رشته ورودی صرفنظر خواهد شد.

تابع getchar( )

- این تابع در فایل سر آیند stdio.h قرار دارد.
- این تابع برای خواندن یک کاراکتر از طریق دستگاه ورودی استفاده می شود.
- این تابع دارای آرگومان نیست و کاراکتری که از ورودی دریافت می کند خروجی آن را تشکیل می دهد.
- تابع getchar() بطور متعارف در یک دستور انتساب یا جایگذاری بکار برده می‌شود.

```
character_variable = getchar( ) ;
```

تابع putchar( )

- این تابع یک ورودی از نوع char می پذیرد و آن را در خروجی استاندارد (صفحه نمایش) نمایش می دهد.
- این تابع به دو صورت زیر قابل استفاده است:

putchar (character_variable) ;

putchar (‘character’)

مثال :

```
char ch='D' ;
putchar(ch) ;
putchar('E') ;
```

خروجی

```
DE
```

مثال : برنامه زیر یک خط متن را از ورودی با حروف کوچک دریافت کرده، آن را به حروف بزرگ تبدیل می‌کند.

```
#include<stdio.h>
#include<ctype.h>
void main ( )
  {
    char  line[80] ;
    int  count , k ;
                                /* read in the line */
    for (k=0 ; (line[k]=getchar( ))!='\n' ;++k) ;
    count = k ;
                                /* write out the line in upper-case */
    for( k=0 ; k<count ; ++k )
          putchar( toupper(line[k]) ) ;
  }
```

خروجی

```
c is a powerful programming language ↵
C IS A POWERFUL PROGRAMMING LANGUAGE
```

تابع getche( )

- این تابع در فایل سر آیند conio.h قرار دارد.
- این تابع برای خواندن یک کاراکتر از طریق دستگاه ورودی استفاده می شود.
- این تابع دارای آرگومان نیست و کاراکتری که از ورودی دریافت می کند خروجی آن را تشکیل می دهد.

- تفاوت این تابع با scanf() و getchar() در این است که وقتی تابع منتظر داده ورودی کاربر است به محض فشرده شدن یک کلید از صفحه کلید کاراکتر فشرده شده دریافت شده و در اختیار برنامه قرار میگیرد.

- از معایب این تابع آن است که کاربر نمی تواند بعد از فشردن کاراکتر اشتباه، آنرا تصحیح کند.

- این تابع کاراکتر فشرده شده را در خروجی نیز نمایش می دهد و از این لحاظ شبیه scanf() و getchar() است.

تابع getch( )

- این تابع در فایل سر آیند conio.h قرار دارد.
- این تابع همانند getche() عمل می کند با این تفاوت که کاراکتر فشرده شده در خروجی نمایش داده نمی شود.

تابع putch( )

- این تابع در فایل سر آیند conio.h قرار دارد.
- این تابع مثل putchar() عمل می کند . تنها تفاوت آنها در فایل سرآیندشان است.

توابع ( )gets و ( ) puts

- این توابع در فایل سر آیند stdio.h قرار دارند.
- این توابع بترتیب برای نوشتن و خواندن رشته از طریق دستگاه ورودی استفاده می شود.
- آرگومان این توابع یک رشته یا اشاره گر کاراکتر خواهد بود.
- ()getsرشته خوانده شده از ورودی را به آرگومان خود نسبت می دهد و ()puts رشته ای را که در آرگومانش قرار می گیرد، در خروجی چاپ می کند.

اصول کامپیوتر ۲

رشته علوم کامپیوتر

۴ واحد درسی

فصل پنجم

نام منبع و مؤلف :

- ✓ اصول کامپیوتر ۲
- ✓ دکتر داود کریم زادگان مقدم
- ✓ انتشارات دانشگاه پیام نور ۱۳۸۳

فصل پنجم: ساختارهای کنترلی و شرطی

- دستور کنترلی switch
- دستور break
- دستور کنترلی while
- دستور کنترلی do-while
- دستور کنترلی for
- دستور exit()
- دستورهای شرطی if و if-else

هدف کلی

آشنایی با ساختار های کنترلی و دستورهای شرطی و کاربرد آنها در برنامه نویسی

هدف های رفتاری

- شکل کلی و کاربرد دستورهای کنترلی if و if-else
- شکل کلی و کاربرد دستور switch
- شکل کلی و کاربرد دستور while و do-while و تفاوت آن با while
- شکل کلی و کاربرد دستور for و continue
- شکل کلی و کاربرد دستور goto و تابع exit()

مقدمه Introduction

- استفاده از دستورات و ساختارهای کنترلی، یکی از ویژگی های زبان های برنامه‌سازی پیشرفته است.
- این دستورات و ساختارها امکان ایجاد حلقه های تکرار و ساختارهای تصمیم گیری را در برنامه ها فراهم می آورند.
- دستور های for , while , do-while به عنوان ساختار حلقه های تکرار در زبان C شناخته می شوند.

فصل پنجم : ساختارهای کنترلی و شرطی – مقدمه

- دستور های if و switch به عنوان دستورهای شرطی یا ساختارهای تصمیم گیری شناخته می شوند.
- و دستورهای exit , break , continue , goto دیگر دستورات کنترلی هستند.
- اغلب دستورهای کنترلی برنامه در زبان C روی نتیجه ی وجود شرطی تکیه می‌کنند تا برحسب برقراری آن شرط عملی انجام گیرد و یا انجام نگیرد.
- در زبان C نتیجه شرط ها یک مقدار تولید می کنند که نتیجه بر اساس آن مقدار مشخص می شود، بطوری که برای مقدار صفر ارزش نادرست و برای تمامی مقادیر غیر صفر ارزش درست درنظر گرفته می شود.

دستور کنترلی while

- فرم کلی دستور while :

```
while (condition)
    statement ;
```

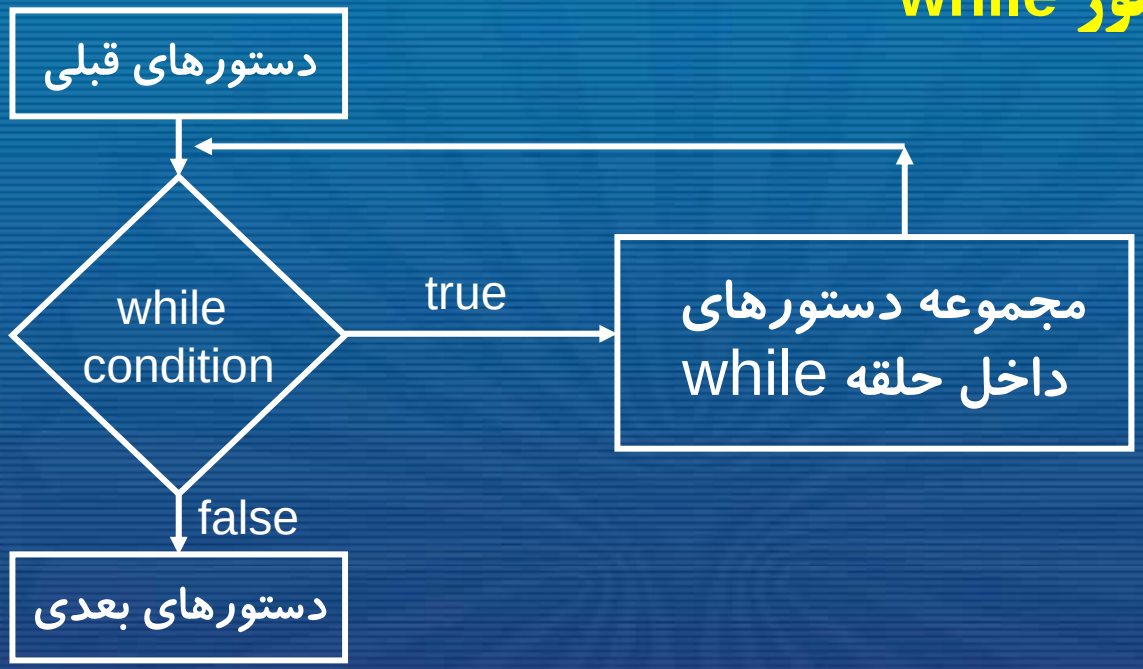
و یا

```
while (condition)
{
    statement ;
}
```

- این دستور برای ایجاد حلقه های تکرار مورد استفاده قرار می گیرد.

- با استفاده از این دستور ، یک حلقه تا موقعی که شرط معینی برقرار باشد اجرا می‌گردد.

نمودار کلی دستور while



- تا موقعی که شرطی که پس از کلمه کلیدی while در داخل پرانتزها نوشته شده، برقرار باشد، مجموعه دستورهای داخل حلقه while به صورت تکراری اجرا خواهد شد، و به محض نقض شرط کنترل از حلقه خارج خواهد شد.

مثال : برنامه زیر اعداد ۰ تا ۵ را در خروجی استاندارد چاپ می کند:

```
#include<stdio.h>
main( )
{
   int  number = 0 ;
   while (number<=5)
      printf ("%d\n", number ++);
  return 0;
}
```

خروجی

```
0
1
2
3
4
5
```

توجه کنید که بعد از خروج کنترل از حلقه while، مقدار متغیر number برابر ۶ خواهد بود.(چرا؟)

مثال : برنامه زیر عدد صحیح n را از ورودی گرفته و فاکتوریل آن را چاپ می کند:

```
#include<stdio.h>
void main ( )
 {
    int  n , i =1 , fact =1 ;
    scanf ("%d",&n) ;
    while (++ i <= n )
        fact *=  i ;
    printf ("factorial of %d is %d", n , fact ) ;
 }
```

خروجی

```
5 ↵
factorial of 5 is 120
```

مثال : برنامه زیر عدد صحیح n را از ورودی گرفته و مجموع ارقام آن را محاسبه و چاپ می کند.

```
#include<stdio.h>
#include<conio.h>
void main ( )
{
    int  n ,temp , sum =0 ;
    scanf ("%d",&n) ;
    temp=n;
    while (n>0 )
    {
        sum+=n%10;
        n/=10;
    }
    printf (“sum of the digits of the %d is %d", temp , sum ) ;
    getch();
}
```

خروجی

```
345 ↵
sum of the digits of the 345 is 12
```

دستور کنترلی do while

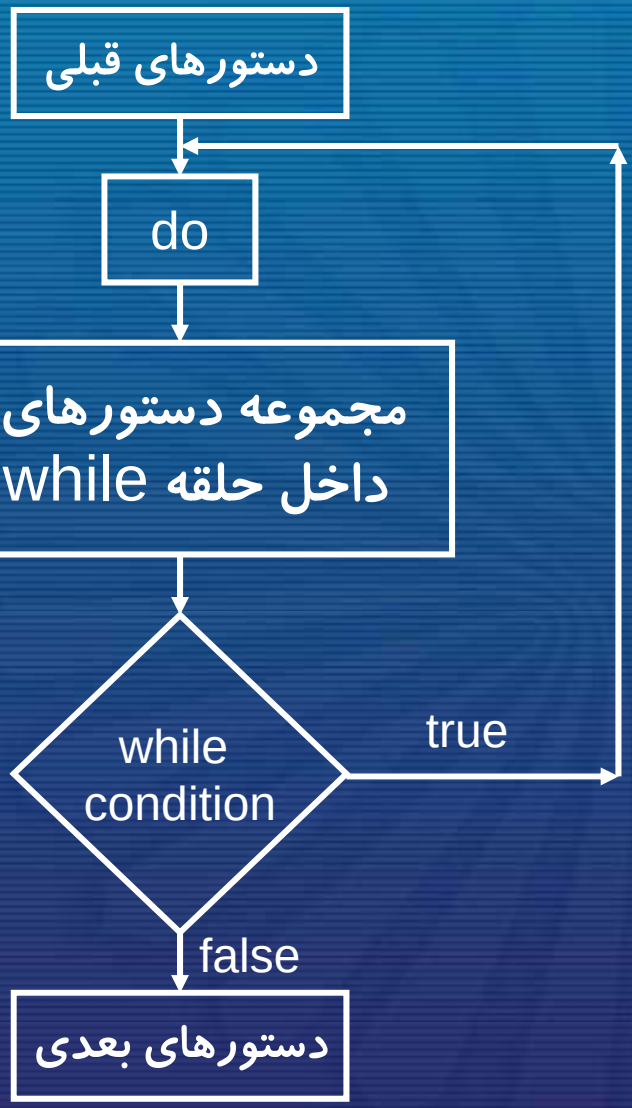
- فرم کلی دستور do-while :

```
do
{
        statement ;

} while(condition)
```

- این دستور مشابه دستور while است با این تفاوت که در دستور do-while شرط حلقه در آخر هر تکرار بررسی می شود.

- در این دستور، بدنه حلقه همیشه حداقل یک بار اجرا میشود و بعد شرط حلقه بررسی می شود ، در صورت درست بودن شرط اجرای حلقه ادامه پیدا میکند.

نمودار کلی دستور do-while



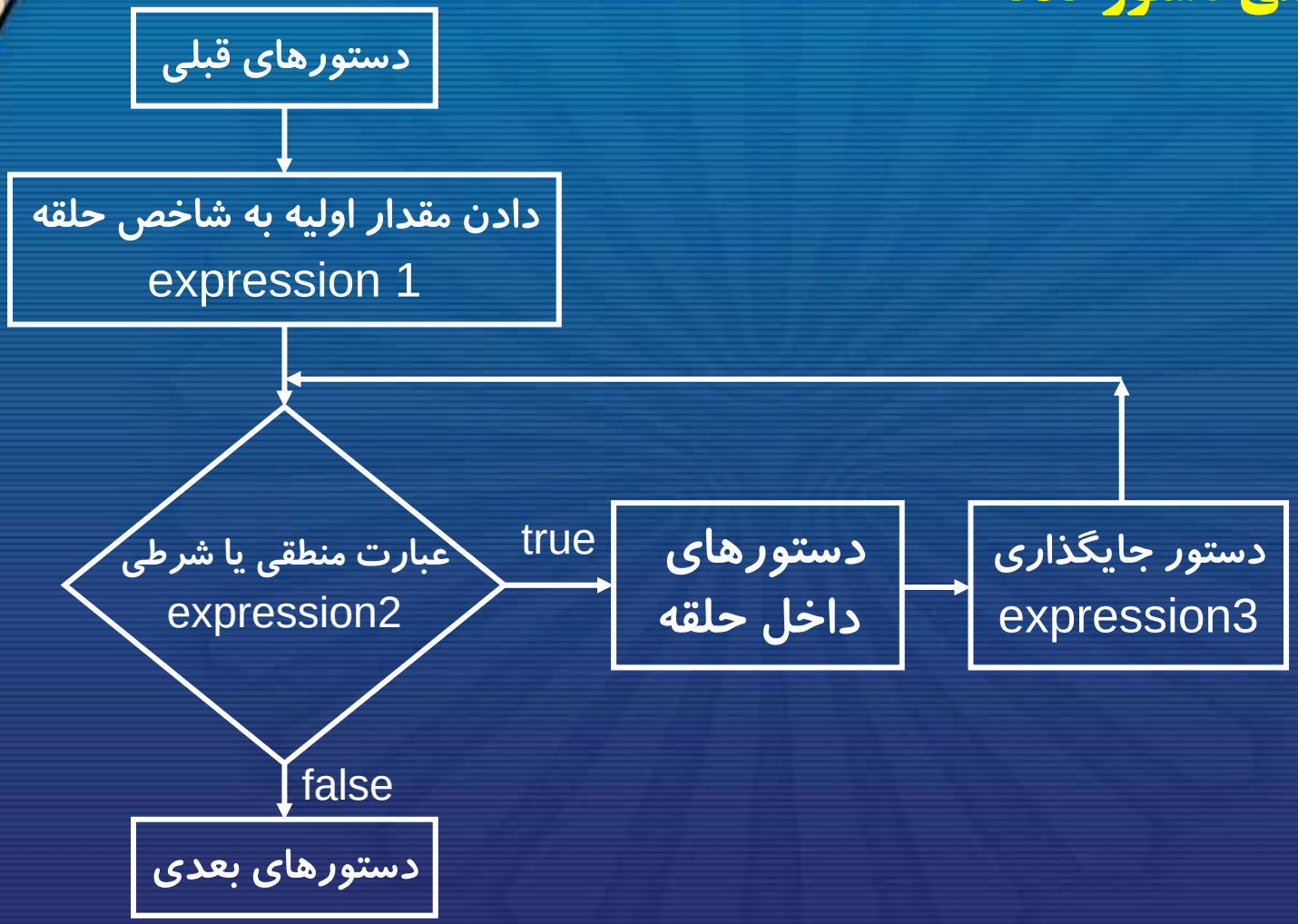
دستور کنترلی for

- فرم کلی این دستور به صورت زیر است:

```
for (expression1; expression2 ; expression3)
{
  statement ;
}
```

- دستورfor متداولترین دستور حلقه در زبان C است.
- این دستور شامل ۳ عبارت و یک شاخص است:
  - عبارت اول مقدار نخستین شاخص را مشخص میسازد.
  - عبارت دوم چگونگی ادامه یا پایان حلقه را تعیین می کند.
  - و سومین عبارت، شاخص را در پایان هر حلقه تغییر می دهد.

نمودار کلی دستور for



- معمولاً expression1 یک عبارت جایگذاری ، expression2 یک عبارت منطقی یا رابطه‌ای و expression3 یک unary expression و یا یک عبارت جایگذاری می‌باشد.

- دستور for معادل قطعه برنامه زیر است:

```
expression1 ;
while (expression2)
  {
    statements
    expression3 ;
  }
```

- در دستور for سه عبارت به کار می رود که بین آنها از دو سمیکلون استفاده می شود؛ حذف هر یک از عبارتها مجاز است ولی حذف سمیکلونها مجاز نمی باشد.

```
for ( ; ; ) ;
```

- در صورت حذف هر یک از عبارتهای ۱ یا ۳ یا هر دو، باید اعمال مربوط به آنها را به طرق دیگری انجام داد ولی حذف عبارت ۲ باعث میشود، حلقه for بینهایت بار اجرا شود.

- معمولا حلقه for بیشتر از دیگر حلقه ها مورد استفاده قرار می گیرد.

مثال: برنامه ی زیر n عدد از ورودی دریافت کرده میانگین آنها را محاسبه و چاپ می کند.

```
#include<stdio.h>
#include<conio.h>
main ( )
{
   int n,i,temp,sum=0;
   float average;
   printf("Enter number of numbers:");
   scanf("%d",&n);
   for ( i=0 ; i<n ; i++ )
   {
      printf("\nEnter number:");
      scanf("%d",&temp);
      sum+=temp;
   }
   average = (float)sum / n;
   printf("\n\nAverage is : %5.2f",average);
   getch();
   return 0;
}
```

خروجی

```
Enter number of numbers: 5
Enter number: 12
Enter number: -2
Enter number: 5
Enter number: 1
Enter number: 3

Average is : 3.80
```

- حلقه های تکرار می توانند تودرتو تعریف شوند در این صورت حلقه داخلی تر برای هر تکرار حلقه خارجی ، یک بار بطور کامل اجرا خواهد شد.

- در صورت تودرتو بودن حلقه های for، باید برای هر حلقه از اندیس متفاوت استفاده گردد.

- حلقه for اجازه پردازش چندین اندیس را بطور یکجا دارد.

مثال : برنامه زیر با استفاده از حلقه های for تودرتو جدول ضرب n×n چاپ می کند:

```
#include<stdio.h>
main()
{
  int n,i,j;
  scanf("%d",&n);
  for( i=1 ; i<=n ; i++)
  {
    for(j=1 ; j<=n ; j++)
        printf("%4d",i*j);
    printf("\n");
  }
  return 0;
}
```

خروجی

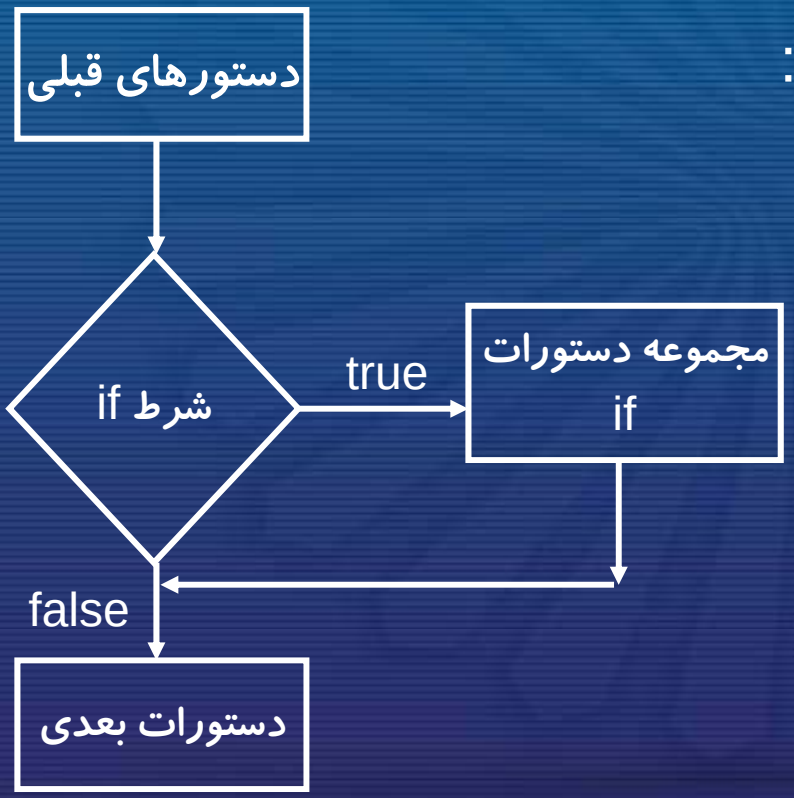
```
5

1   2   3   4   5
2   4   6   8   10
3   6   9   12  15
4   8   12  16  20
5   10  15  20  25
```

- در صورت استفاده از چند اندیس برای حلقه for تمام اندیسها در محل عبارت ۱ مقداردهی شده و با عملگر کاما از یکدیگر جدا می شوند و همینطور عملیات تغییر تمام اندیسها در محل عبارت ۳ انجام شده و با عملگر کاما از یکدیگر جدا می شوند:

```
for ( i=0 , j=3 ; i<n ; i++, j+=3 )
{
  statements;
}
```

دستورهای شرطی if و if-else

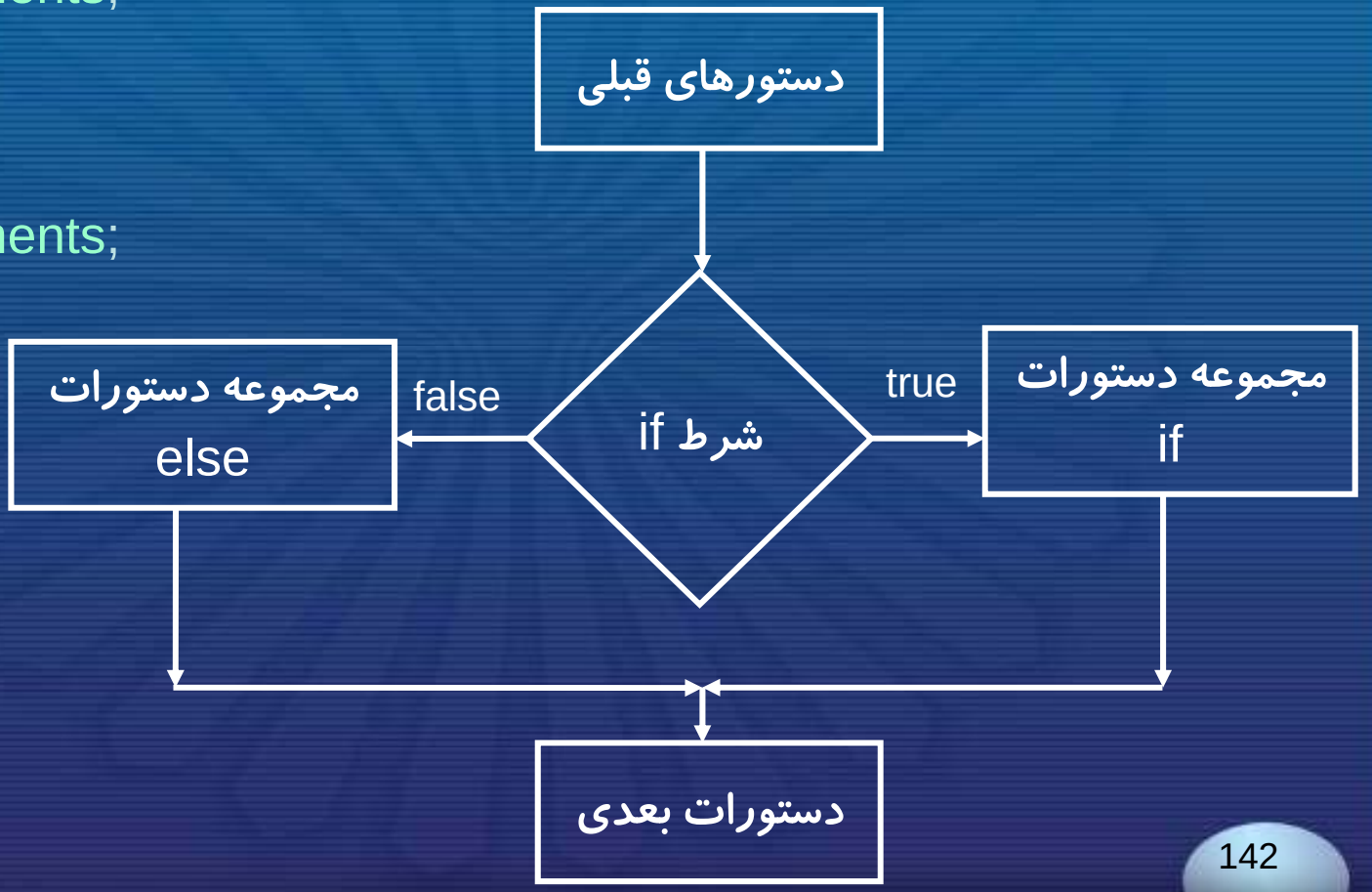
- دستورهای شرطی(conditional statements) برای انجام یک آزمون منطقی و برگزیدن یکی از دو حالت ممکن که به نتیجه آزمون بستگی دارد استفاده می شود.
- فرم کلی دستور if بصورت زیر است:




```
if (condition)
{
    statements;
}
```

شکل کلی دستور if-else به صورت زیر است:

```
if (condition)
{
   statements;
}
else
{
   statements;
}
```

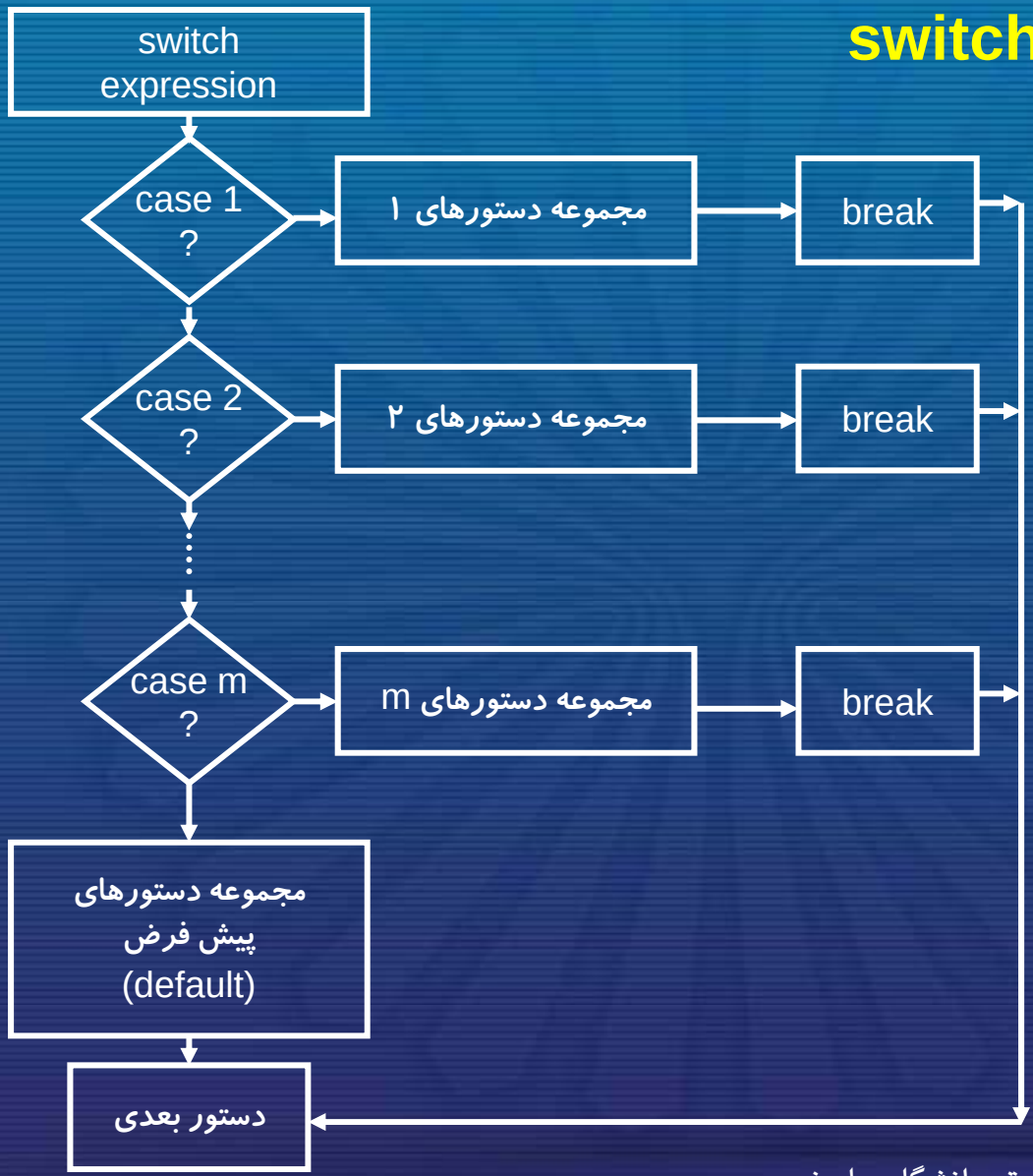


دستورشرطی switch

فرم کلی ساختارswitch به صورت زیر است :

```
switch(expression)
{
   case   exp1 :  statements 1
                  break ;
   case   exp2 :  statements 2
                  break ;
   . . . .
   . . . .
   case   exp n :  statements n
                  break ;
       default   :  statements
                  break ;
}
```

نمودار کلی دستور switch



- وقتی بخواهیم بر اساس مقدار یک عبارت ، عمل یا اعمالی را از بین گروهی از اعمال انتخاب و اجرا کنیم ، از دستور switch استفاده می کنیم.

- این ساختار بر اساس مقادیر گسسته کار می کند نه مقادیر پیوسته.

- وقتی مقدار عبارت داخل پرانتز مقابل switch با یکی از مقادیر مقابل caseها برابر شد دستورات بعد از آن تا رسیدن به اولین دستور break اجرا شده و کنترل به اولین دستور بعد از switch منتقل می شود.

- آخرین عبارت دستور switch ؛ default است، تعریف این عبارت اختیاری است و زمانی که عبارت switch با عبارت هیچیک از caseها مطابقت نکند، دستورات مقابل default اجرا خواهد شد.

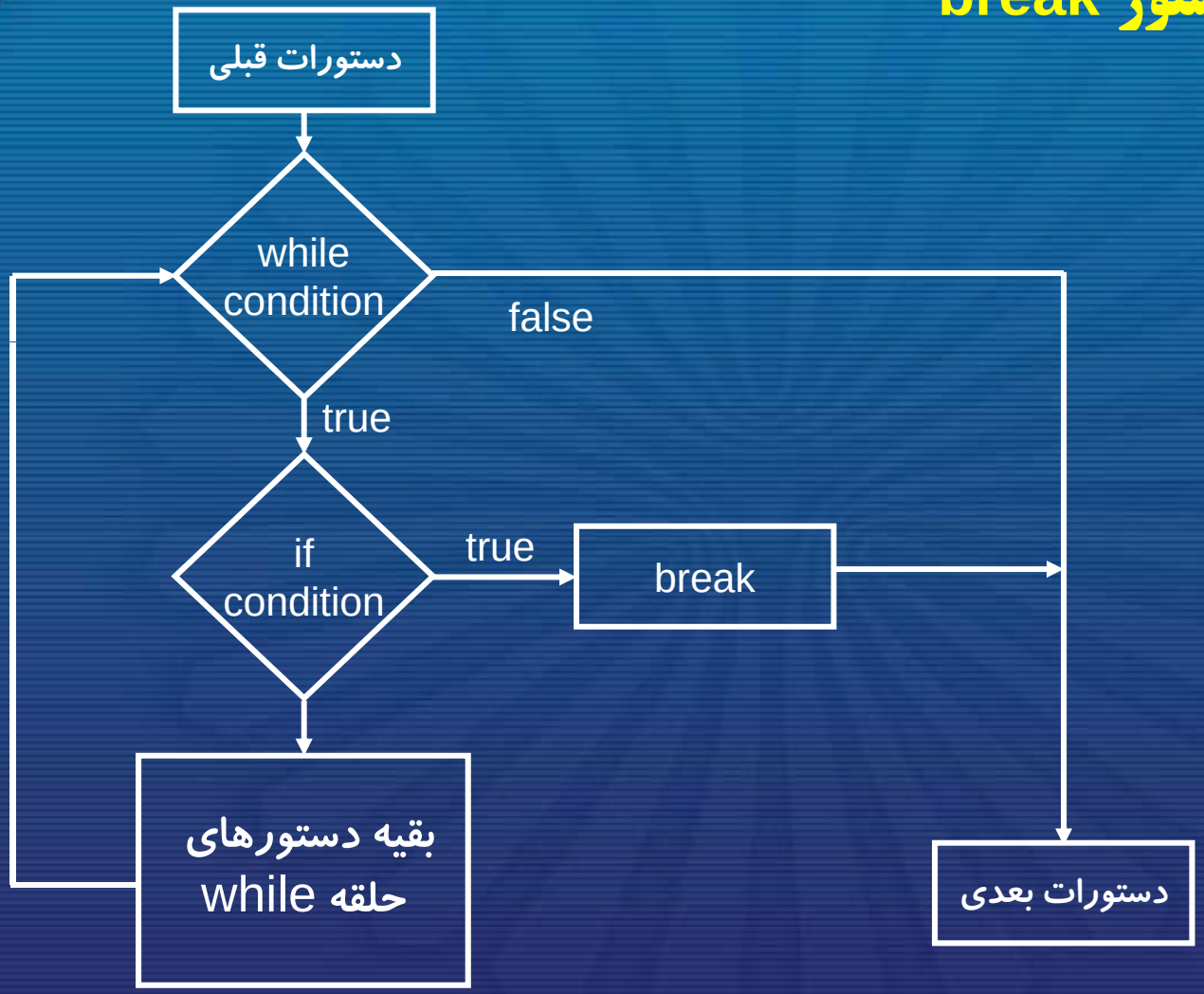
- اگر برای دستور switch ، default تعریف نکنیم و عبارت switch با عبارت هیچیک از caseها مطابقت نکند ، بدون اینکه دستوری اجرا شود کنترل از دستور switch خارج خواهد شد.

- اگر برای هریک از caseها break ننویسیم، با case بعدی OR میشود.

دستور break

- این دستور مسیر معمولی و ثابت کنترل را تغییر داده و یا قطع می کند.
- دستورbreak برای خارج شدن از یک دستورswitch و یا پایان دادن به اجرای حلقه ها استفاده می شود.
- دستور break باعث خروج زود هنگام کنترل از داخل حلقه های for وwhile و do-while می شود.
- اگر دستور break داخل حلقه های تودرتو استفاده شود فقط باعث خروج از حلقه جاری می شود.

نمودار کلی دستور break



مثال : برنامه مقابل اعداد صحیح را گرفته و با هم جمع می کند، تا وقتی که عدد 0 وارد شود و در آخر حاصل را نمایش می دهد.

```
#include<stdio.h>
#include<conio.h>
void main()
{
  int n,sum=0;
  while(1)
  {
    scanf("%d",&n);
    sum+=n;
    if(!n)
      break;
  }
  printf("\nSum=%d",sum);
  getch();
}
```

خروجی

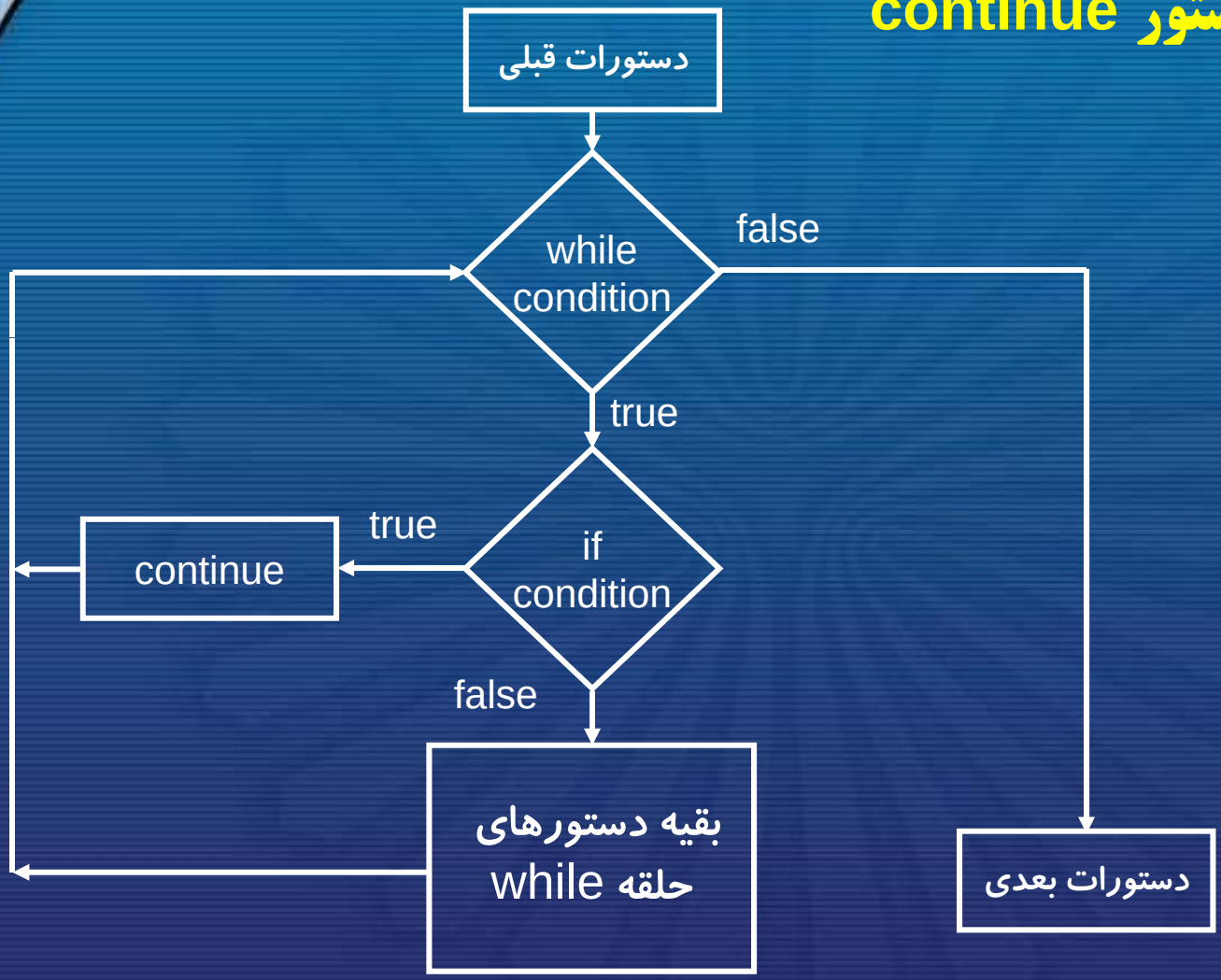
```
1
12
541
-4
0

Sum=550
```

دستور continue

- این دستور نیز مسیر معمولی و ثابت کنترل را تغییر داده و یا قطع می کند.
- دستور continue باعث می شود باقیمانده تکرار جاری یک حلقه نادیده گرفته شده و بلافاصله تکرار بعدی حلقه آغاز گردد.
- این دستور می‌تواند با ساختارهای for , do-while , while بکار برده شود و فرم کلی آن به صورت ساده زیر است :

```
continue ;
```

نمودار کلی دستور continue



دستور goto

- این دستور ترتیب طبیعی اجرای برنامه را تغییر داده و کنترل را به قسمت دیگری از برنامه منتقل میکند .
- در واقع یک انتقال کنترل بدون شرط بوده و اصول برنامه‌سازی ساخت‌یافته در زبان C را نقض می‌نماید .
- استفاده از دستور goto باعث می شود برنامه خوانایی خود را از دست بدهد؛ پس استفاده بی مورد از این دستور توصیه نمی شود.

دستور exit()

- الگوی این تابع در فایل stdlib.h قرار دارد.
- این دستور یک تابع است و اجرای برنامه را بطور کامل قطع کرده و موجب خاتمه پذیرفتن کل برنامه می‌گردد.
- این تابع که در کتابخانه استاندارد وجود دارد معمولاً با آرگومان صفر فراخوانی می‌شود و دلالت بر خاتمه پذیرفتن طبیعی و نرمال برنامه می‌کند.
- کاربرد متداول exit() زمانی است که یک شرط خاص برای اجرای برنامه، برقرار نباشد و اجرای برنامه قطع شود.

کدهای توسعه یافته

- صفحه کلید برای حروف ، اعداد و علائم مخصوص، کدی یک بایتی تولید می کند .
- برای بسیاری از کلیدهای دیگر و یا ترکیبی از چند کلید ، کدی دو بایتی تولید می گرددکه به آن کد توسعه یافته گفته می شود .
- بایت اول کد توسعه یافته برابر با صفر و بایت دوم آن کد کلید است.

برخی کدهای توسعه یافته

بایت دوم کد توسعه یافته	نام کلید
59	F1
60	F2
61	F3
62	F4
63	F5
64	F6
65	F7
66	F8
67	F9
68	F10

مثال : برنامه مقابل کد اسکی کلید های فشرده شده از صفحه کلید را در خروجی نمایش می دهد، اگر کاراکتر فشرده شده '@' باشد برنامه خاتمه پیدا می کند.

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
void main()
{
  char ch;
  ch=getch();
  while(1)
   {
      if(ch=='@')
         exit(0);
       if(ch==0)
       {
          ch=getch();
          printf("\n00 %d",ch);
       }
       else
          printf("\n%d",ch);
       ch=getch();
   }
}
```

خروجی

```
105
48
32
00 72
00 64
97
65
```

کلیدهای فشرده شده بترتیب از چپ به راست:

i,0,space,up arrow,F7,a,A,@

فصل ششم : برنامه سازی پیمانه ای

اهداف کلی فصل

پارامترهای خط فرمان

تعریف تابع ها

کلاستوهای static

فراخوانی تابع

توابع بازگشتی

هدف کلی

آشنایی با توابع و ماکروها و کاربرد آنها در برنامه نویسی

هدف های رفتاری

- [illegible] آنها
- [illegible] از آن
- [illegible] برنامه
- [illegible] تابع
- [illegible] return

مقدمه Introduction

- تابع یک قطعه برنامه کامل است که کار معینی را انجام می دهد و موجب جلوگیری از برنامه نویسی تکراری در بین برنامه ها می شود .
- هر برنامه در زبان C، مجموعه‌ای از یک یا چندین تابع است که فقط یکی از آنها تابع اصلی(main) است و بقیه تابع فرعی هستند .
- توابع در C به دو دسته تقسیم می شوند:

- به تقسیم بندی برنامه به قسمتهای کوچک و نوشتن توابع برای هر قسمت ، برنامه نویسی ماژولار گفته می شود.

- مزایای طراحی برنامه به صورت پیمانه‌ای یا ماژولار:
  - امکان استفاده مجدد از تابع
  - امکان انجام تغییرات در برنامه و اصلاح آن
  - ساده شدن کنترل و خطایابی
  - امکان همکاری برنامه‌نویسان متعدد در نوشتن یک برنامه

تعریف تابع

بطور کلی هر تابع شامل عناصر زیر است:

عنوان تابع - شامل نام تابع ، نوع تابع و آرگومانهای تابع است .

بدنه تابع - شامل موارد زیر است:

- اعلان متغیرهای محلی
- سایر دستورهای تابع

```
< نوع تابع >< نام تابع >( اسامی پارامترها)
{
  اعلان متغیرهای محلی ;
  سایر دستورهای تابع ;
}
```

عناصر یک تابع

دستور return

- برای خروج از تابع از دستورreturn استفاده می شود.
- اگر نیاز باشد که تابع ، مقداری را برگرداند دستورreturn دارای آرگومان خواهد بود.
- دستور return تنها می تواند شامل یک آرگومان باشد .
- چنانچه نیاز باشد که بیش از یک مقدار به تابع فراخواننده بازگردانده شود ، باید از اشاره گرها استفاده کرد.

مثال: تابع زیرکاراکتری را از ورودی خوانده و اگر از حروف بزرگ باشد به حرف کوچک آن برمی گرداند در غیر اینصورت خود کاراکتر دریافت شده برگردانده می شود.

```
char  UptoLow (void)
{
  char  ch ;
  ch = getchar( ) ;
  if ((ch > 64) && (ch < 91))
      return (ch + 32) ;
  else
      return (ch) ;
 }
```

توابعی که دارای انشعاب های چندگانه اند اغلب به چند return احتیاج پیدا می کنند .

مثال: برنامه زیر عددی صحیح از ورودی خوانده ، سپس با فراخواندن تابعی به نام fact ، فاکتوریل عدد مزبور محاسبه شده و در خروجی چاپ می شود.

تابع fact بدون مقدار برگشتی

```
#include<stdio.h>
void fact(int);
main ( )
}
 printf("n=");
  scanf("%d , &n) ;
  fact(n) ;
}
void  fact( int  n )
{
  int  f = 1 , i ;
  for ( i =2 ; i<=n ; ++i )
     f = f * i ;
  printf ("fact (n)= %d" , f) ;
}
```

تابع fact با مقدار برگشتی

```
#include<stdio.h>
int fact(int);
main ( )
{
  int   n ;
  long  int  factorial ;
  printf("n = " ) ;
  scanf ("%d" , &n) ;
  factorial = fact(n) ;
  printf ("\n fact (n) = %d"  , factorial ) ;
}
int  fact ( int  n)
{
  int  i ;
 long int f = 1 ;
  if  ( n >1 )
     for ( i = 2 ; i<=n ; ++i )
          f = f * i ;
  return( f ) ;
}
```

- اگر نوع خروجی تابع در اعلان آن مشخص نشود، نوع مقدار خروجی بطور پیش فرض int درنظر گرفته میشود.

- پس اگر نوع خروجی تابع غیر از int باشد باید در اعلان تابع مشخص شود ویا قبل از هر فراخوانی تابع ، باید نوع آن مانند متغیر معمولی در تابع فراخوانده شده توصیف گردد .

مثال : در برنامه زیر تابعی تعریف شده که حاصل جمع دو عدد را به تابع اصلی باز می گرداند

```
main ( )
{
  float  sum (float , float) ;
  float  x , y ;
  scanf("%f %f " , &x ,& y) ;
  printf ("%f " , sum (x , y)) ;
}
```

```
float sum (float a , float b)
{
   return (a + b) ;
}
```

فراخوانی تابع

- بطور کلی در زبان C فراخوانی یک تابع با نوشتن نام تابع و پارامترهای آن به صورت یک تک دستور و یا در یک دستور جایگذاری ، انجام می‌شود.
- در صورت نداشتن پارامتر زوج پرانتز خالی بکار می رود :

```
ch = getchar();
putchar(ch);
```

- ممکن است نام تابع در یک عبارت محاسباتی و یا یک دستور خروجی فراخوانی شود.

- تقسیم بندی توابع، ازنظر نحوه انتقال آرگومانها :
  - فراخوانی با مقدار یا فراخوانی توسط ارزش،
  - فراخوانی توسط آدرس یا فراخوانی توسط ارجاع؛
- در روش اول خود متغیرها یا آرگومانها به تابع فراخوانده شده (تابع فرعی) انتقال نمی‌یابد ، بلکه فقط مقدار آنها به تابع فراخوانده شده انتقال می‌یابد.
  یعنی مقدار آن آرگومانها ، در تابع فراخواننده تغییر نخواهد کرد.

مثال: به برنامه زیر توجه کنید:

```
#include<stdio.h>
main ( )
{
   int  x = 2  ;
   printf (“before calling the function x=%d" , x )
   calc (x ) ;
   printf ("\nafter calling the function x=%d" , x )
}
void  calc (int  x )
{
   printf ("\nin the function before increase x=%d " , x );
   x ++ ;
   printf ("\ nin the function after increase x=% d " , x ) ;
}
```

خروجی

```
before calling the function x=2
in the function before increase x=2
in the function after increase x=3
after calling the function x=2
```

تغییر مقدار x در تابع فرعی calc هیچ تاثیری بر مقدار آن در تابع اصلی نداشته است.

- آرگومانهایی که به تابع فرعی فرستاده می‌شوند، متغیرهای مجازی یا متغیرهای مستعار نامیده می‌شوند و اسامی آنها در تابع فرعی می‌تواند متفاوت از نام آنها در تابع اصلی باشد.
- بنابراین برنامه اسلاید قبلی را می‌توان به صورت زیر نیز نوشت :

```
#include<stdio.h>
main ( )
{   int  x = 2 ;
     printf (" %d" , x ) ;
     calc (x ) ;
     printf ("\n d" , x ) ;
}
void  calc (int  k)
{
   printf ("\n%d" , k );
   k ++ ;
    printf ("\n %d " , k) ;
}
```

تابع بازگشتی

- به توابعی که در حین اجرا به صورت مستقیم یا غیر مستقیم خود را فراخوانی می کنند توابع بازگشتی می گویند.
- استفاده از این توابع جهت انجام محاسبات تکراری که در آن، منطقی وجود دارد مناسب است.
- توابع بازگشتی برای مسائلی مطرح می شود که در منطق آنها حالت توقف وجود دارد ، و وقتی برنامه بعد از چندین فراخوانی به آن حالت رسید، عمل فراخوانی متوقف شده و تابع یک مقدار بر می گرداند.

- به عنوان مثال پیدا کردن فاکتوریل یک عدد می تواند به صورت بازگشتی تعریف شود:

می دانیم :

$$m! = m * \underbrace{(m-1) * (m-2) * \ldots * 3 * 2 * 1}_{(m-1)!}$$

- پس می توانیم بنویسیم:

$$m! = m * (m-1)!$$

$$(m-1)! = (m-1)*(m-2)!$$

$$(m-2)! = (m-2)*(m-3)!$$

$$3! = 3 * 2!$$

$$2! = 2 * 1!$$

حالت توقف

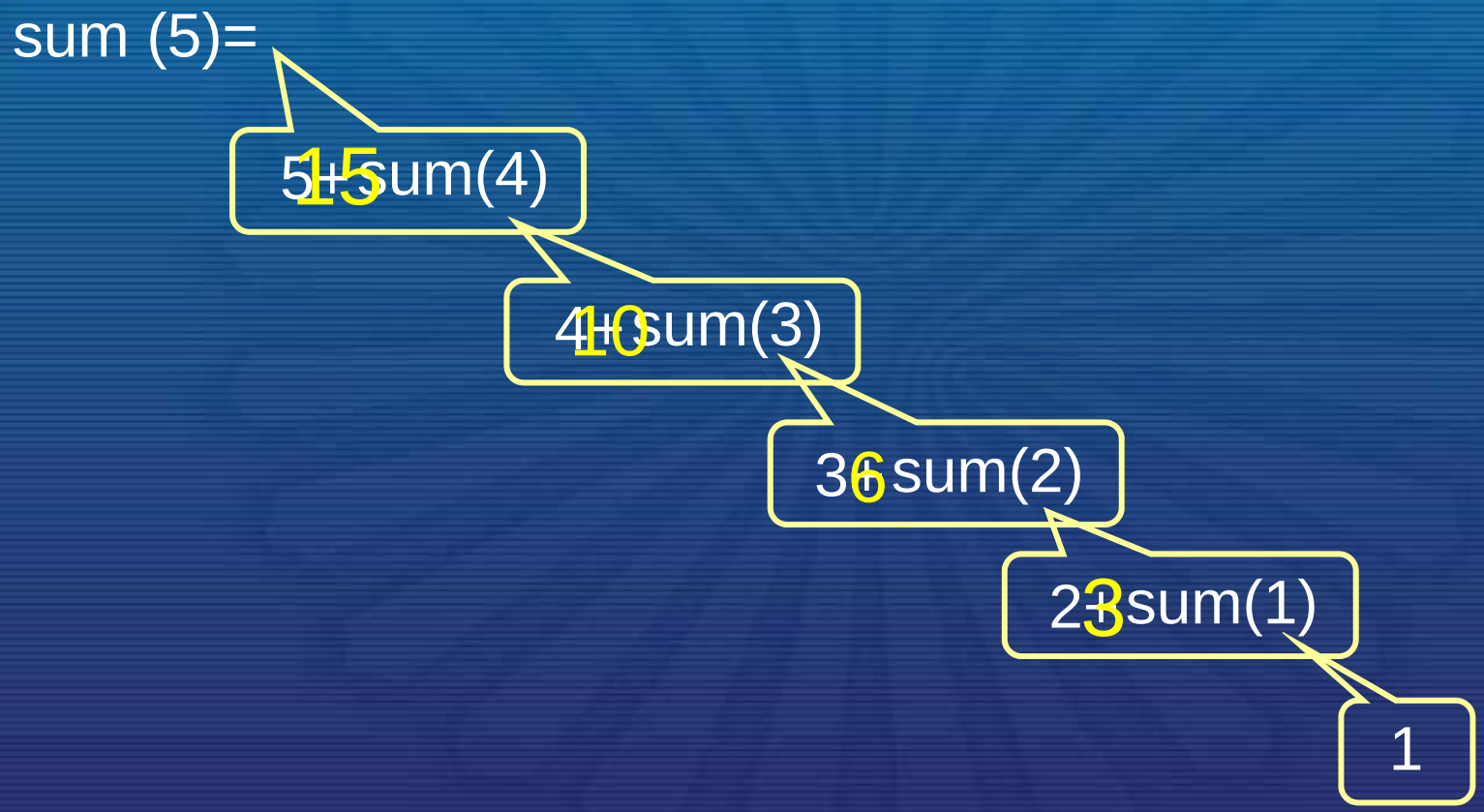
1

- توابع بازگشتی حتما دارای یک ساختار شرطی برای تشخیص حالت توقف هستند.

مثال: تابع زیر مجموع اعداد طبیعی ۱ تا n را به صورت بازگشتی محاسبه می‌کند:

```
int  sum(int  n)
 {
    if (n<=1)
        return (1) ;
    else
        return ( n + sum (n-1)) ;
 }
```

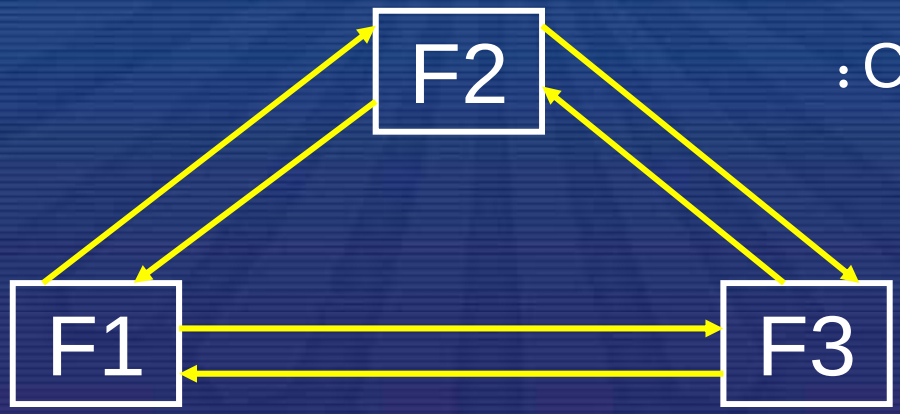
نحوه عملکرد تابع اسلاید قبل برای ورودی 5 :



استفاده از چند تابع

- زبان C این امکان را به برنامه نویس می دهد که هر تعداد تابع مورد نیاز باشد تعریف کند.
- در زبان C توابع نمی توانند تودرتو تعریف شوند؛ همه توابع در یک سطح تعریف میشوند.
- همه توابع بهم دسترسی دارند و می توانند همدیگر را فراخوانی کنند.

ارتباط توابع در C :



پارامترهای خط فرمان

- اگر بخواهیم پارامترهایی را به عنوان پارامترهای تابع اصلی(main) به برنامه منتقل کنیم آنها را هنگام اجرای برنامه در خط فرمان سیستم عامل بعد از نام برنامه درج می کنیم.

- برای دریافت آرگومان خط فرمان، دو آرگومان مخصوص از پیش تعریف شده به اسامی argc و argv بکار برده می‌شود.

- argc از نوع int و argv از نوع آرایه ای از اشاره گرهای کاراکتری (رشته ها) می باشد.
- argc تعداد پارامتر های ورودی، و argv خود پارامترها را نگه می دارد.
- نام برنامه همیشه به عنوان اولین پارامتر و در نتیجه اولین عنصر آرایه ی argv خواهد بود.
- argc و argv درداخل برنامه های C شناخته شده اند و بر حسب نیاز می توان از آنها استفاده کرد.

مثال: برنامه زیر آرگومانهای ورودی از خط فرمان را داخل برنامه نمایش می دهد:

```
#include<stdio.h>
void main (int  argc , char *argv[ ])
{
   int  count ;
   printf("argc = %d\n" , argc) ;
   for (count = 0 ; count<argc ; + +count)
   printf ("argv[%d] = %s\n" , count , argv[count]) ;
 }
```

خط فرمان

```
C:\TC\Bin>prog605.exe  hello  14  21no  …
```

خروجی

```
argc = 5
argv[0] = C:\TC\Bin\PROG605.exe
argv[1] = hello
argv[2] = 14
argv[3] = 21no
argv[4] = …
```

قلمرو متغیرها

- متغیرهایی که در داخل یک تابع تعریف گردند(متغیرهای محلی) فقط در درون همان تابع شناخته می‌شوند و سایر توابع یا تابع اصلیmain ، آنها را نمی‌شناسد.

- این متغیرها با وارد شدن کنترل برنامه به تابع (یا بلاک) ، از سیستم حافظه می گیرند و با خارج شدن کنترل از تابع (یا بلاک)، حافظه را آزاد می کنند، (از بین می روند).

مثال: به تابع زیر توجه کنید :

```
func1( )
{
    int  x ;
   scanf ("%d" , &x) ;
   if ( x <=1)
      {
         char str[20] ;
         printf ("Enter City : ") ;
         gets(str) ;
         puts(str) ;
      }
}
```

در اینجا متغیر محلی str هنگام ورود به بلوک مربوط به if ایجاد می‌گردد و هنگام خروج از آن تخریب می‌گردد . بعلاوه str فقط در درون بلوک if شناخته شده است و در جای دیگر نمی‌توان به آن مراجعه کرد . حتی در درون همان تابع ، ولی خارج از بلوکی که توصیف شده است .

- متغیرهای عمومی یا global variables متغیرهایی هستند که در طول تمام برنامه شناخته شده می‌باشند و می‌توانند به‌وسیلهٔ هر قسمت از برنامه بکار برده شوند.
- متغیرهای عمومی در بالای تابع main تعریف می شوند.

```
int  count ;        /*  global  variable */
main( )
 {
   count = 100 ;
   printf("\ncount =%d",count);
   func1( ) ;
   printf("\ncount =%d",count);
 }
func1( )
  {
    int  temp ;
    temp = count ;
    count = 20;
    printf("\ntemp=%d" , temp) ;
  }
```

خروجی

```
count = 100
temp = 100
count = 20
```

کلاسهای حافظه

کلاس حافظه ، قلمرو متغیر و نیز زمان حیات یک متغیر را در یک برنامه مشخص می‌کند.

انواع کلاسهای حافظه در C

- اتوماتیک (automatic)
- استاتیک (static)
- ثبات (register)
- خارجی (external)

کلاس حافظه اتوماتیک

- متغیرهای تعریف شده داخل توابع (متغیر های محلی) بطور پیش فرض از این نوع کلاس هستند.
- این متغیرها با فراخوانی تابع ایجاد می شوند و با خاتمه اجرای تابع از بین می روند.
- برای تعیین کلاس حافظه اتوماتیک ، از کلمه کلیدی auto استفاده می شود.
- ویژگیهای متغیرهای تعریف شده با کلاس حافظه اتوماتیک :

۱- فقط در همان تابعی که تعریف می شوند قابل استفاده اند.

۲- طول عمر این متغیرها محدود به زمان اجرای تابعی است که در آن تعریف شده اند.

کلاس حافظه ثبات

- متغیرهای تعریف شده با این کلاس حافظه از نوع اتوماتیک هستند پس حوزه و طول عمر مشابه با کلاس حافظه اتوماتیک دارند.

- کامپایلر برای اختصاص حافظه به این متغیر ها از ثبات CPU درخواست حافظه می کند ؛ اگر امکان آن و جود داشته باشد حافظه تخصیص داده می شود و گرنه متغیر ازحافظه اصلی حافظه می گیرد.

- اگر متغیر موفق شود از ثبات CPU حافظه بگیرد ، از سرعت بالایی برخوردار خواهد بود؛ پس بیشتر برای شمارنده های حلقه ها از این کلاس حافظه استفاده می شود.

کلاس حافظه ثبات محدودیتهایی دارد که عبارتند از:

۱- فقط برای متغیر های محلی قابل استفاده اند؛

۲- فقط انواع کاراکتر، صحیح و اشاره گر می توانند با این کلاس تعریف شوند. در مورد سایر انواع بستگی به کامپایلرمورد استفاده دارد؛

۳- چون تعداد ثبات پردازنده محدود است، تعداد کمی از متغیرها می توانند با این کلاس تعریف شوند.

۴- آدرس برای متغیرهایی با کلاس حافظه ثبات، معنی ندارد.

کلاس حافظه استاتیک

به دو دسته تقسیم می شوند:

مقدار اولیه متغیر های استاتیک، صفر است.

ویژگیهای متغیرهای استاتیک محلی :

۱- فقط در همان تابعی که تعریف می شوند قابل استفاده اند.

۲- هنگام فراخوانی تابع ایجاد می شوند و هنگام خروج از تابع نه تنها از بین نمی روند بلکه آخرین مقدار خودشان را حفظ می کنند.

۳- فقط یک بار مقدار اولیه می گیرند.

مثال : با برنامه زیر تفاوتهای متغیرهای کلاس حافظه اتوماتیک و استاتیک بخوبی مشاهده می شود:

```
#include<stdio.h>
#include<conio.h>
void  func ( void ) ;
void main ( )
{
   register int  i;
   clrscr();
   for(i=0;i<5;i++)
      func();
   getch();
}
void  func ()
{
   int x=0 ; //automatic variable
   static int y=0;
   printf("\nauto x= %d, static y= %d",x,y);
   x++;
   y++;
}
```

خروجی

```
auto x= 0, static y= 0
auto x= 0, static y= 1
auto x= 0, static y= 2
auto x= 0, static y= 3
auto x= 0, static y= 4
```

ماکرو

- یک ماکرو شناسه ای است که معادل با یک عبارت ، یک دستور و یا گروهی از دستورات تعریف شده باشد .
- به صورت متداول ، تعریف ماکروها در ابتدای برنامه و پیش از تعریف اولین تابع قرار داده می شود.
- در تعریف یک ماکرو می توان آرگومان هایی نیز قرار داد.

مثال: برنامه زیر ماکرویی دارد که جای دو پارامتر اولش را به کمک پارامتر سوم، عوض می کند.

```
#include <stdio.h>
#define swap(x,y,t) ( (t) = (x) , (x) = (y) , (y) = (t) )
void main()
{
  int a=10, b=5, temp;
  printf("\nBefor swapping; a= %d ,b= %d",a,b);
  swap(a,b,temp);
  printf("\nAfter swapping; a= %d ,b= %d",a,b);
}
```

خروجی

```
Before swapping; a=10 ,b=5
After swapping; a=5 ,b=10
```

مثال: برنامه زیر ماکرویی دارد که در چند خط تعریف شده، و همچنین یک آرگومان دارد:

```
#include<stdio.h>
#define  multiply(n)    for( i = 1 ; i < n ; i++ )        \
                              for( j = 1 ; j < n ; j++ )    \
                                 printf("%d" , i*j )  ;

main( )
{
   int  i , j , n ;
   scanf("%d" , n ) ;
   multiply(n)
}
```

برنامه بالا هنگلم کامپایل به صورت مقابل در می آید:

```
#include<stdio.h>
main( )
{
  int  i , j , n ;
  scanf("%d" , n ) ;
  for( i = 1 ; i < n ; i++ )
     for( j = 1 ; j < n ; j++ )
          printf("%d" , i*j )  ;
}
```

اصول کامپیوتر ۲

رشته علوم کامپیوتر

۴ واحد درسی

فصل هفتم

نام منبع و مؤلف :

- ✓ اصول کامپیوتر ۲
- ✓ دکتر داود کریم زادگان مقدم
- ✓ انتشارات دانشگاه پیام نور ۱۳۸۳

فصل هفتم : آرایه

- مفهوم آرایه و عناصر تشکیل دهنده آن
- آرایه‌های یک بعدی و روش‌های مقداردهی
- توابع کار با رشته‌ها و آرایه‌های رشته‌ای
- مرتب‌سازی حبابی
- مرتب‌سازی درجی
- مقداردهی تصادفی به آرایه
- آرایه‌های چند بعدی

هدف کلی

آشنایی با آرایه‌ها و کاربردهای مختلف آن در برنامه نویسی

هدف های رفتاری

- تعریف آرایه و انواع آن و روابط حاکم بر آن
- انواع مختلف برنامه ها با استفاده از آرایه
- روشهای مقایسه ای به منظور جستجو
- انواع روشهای مرتب سازی آرایه
- توابع کتابخانه ای مربوط به رشته ها

مقدمه Introduction

- آرایه، مجموعه عناصری است که دارای ویژگیها و صفات یکسان هستند.
- همه عناصر یک آرایه از یک نوع بوده و بوسیله اندیس مشخص می‌شوند.
- در زبان C می‌توان آرایه‌های چندبعدی نیز تعریف کرد.

تعریف آرایه

آرایهٔ یک بعدی به صورت زیر تعریف می‌گردد:

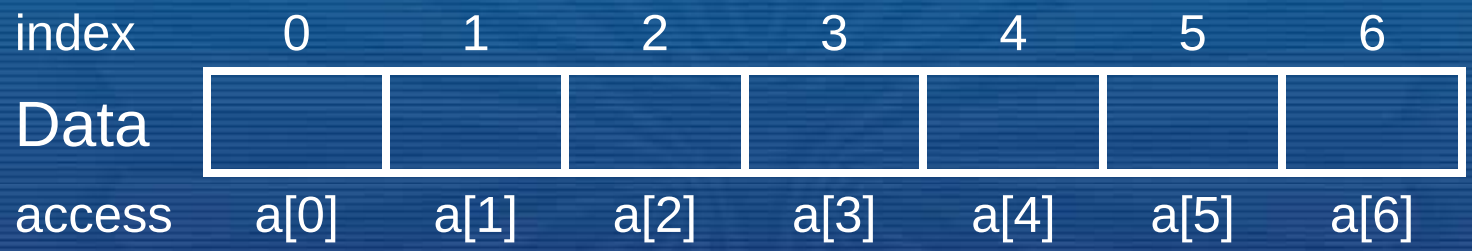
```
type  array_name  [array_size] ;
```

نمونه‌هایی از تعریف چند آرایه یک بعدی

```
int  A[5] ;
float  B[25] ;
static  float  C[15] ;
double  x1[10] ;
char  str[80] ;
```

اندیس آرایه

- اندیس آرایه ها در C از صفر شروع می شود.



- مراجعه به عناصر یک آرایه با شماره ی اندیس آن عنصر انجام می شود.

دستور مقابل عنصر دهم آرایه را چاپ می کند:

```
printf("%d",a[10]);
```

مقداردهی اولیه آرایه

روش تعریف آرایه با مقداردهی اولیه :

storage_class data_type array_name [size] = {value1 , value2 , ... value*m*} ;

که در آن value *1*، value *2*،....value *m* به ترتیب مقدار اولیه اولین ، دومین ، ... و m اُمین عنصر آرایه هستند .

به نمونه های زیر توجه کنید:

```
int  a[7] = {1 , 2 , 3 , 4 , 5 , 6 , 7} ;
float  b[5] = {2.5 ,  -3.5 ,  1.25 ,  12.5  , 3.14} ;
char  c[3] = {'a' , 'b' ,'c'} ;
```

اگر هنگام تعریف آرایه طول آن را تعیین نکنیم و به آن مقدار اولیه دهیم ، سیستم به تعداد مقادیر اولیه، به آرایه حافظه اختصاص می دهد:

طول آرایه ی digit، 5 خواهد بود.

```
int  digit[ ] = {1 , 2 , 3 , 4 , 5} ;
```

مثال : برنامه زیر n (n<=20)عدد صحیح دریافت کرده و بعد از محاسبه ی میانگین آنها عناصر بزرگتر از میانگین و عناصر کوچکتر یا مساوی میانگین را در سطرهای جداگانه نمایش می دهد:

```
#include<stdio.h>
#include<conio.h>
void main()
{
  int n, a[20]={0}, i, sum;
  float average;
  do{
     printf("\nNumber of numbers(n<21):");
     scanf("%d",&n);
     }while(n>20);
  printf(\nEnter %d numbers:\n",n);
  for(i=0; i<n ; i++)
     scanf("%d",&a[i]);
```

ادامه در اسلاید بعد ...

```
    //compute average
sum=0;
for(i=0;i<n;i++)
    sum+=a[i];
average=(float)sum/n;
printf("\nAverage= %5.2f ",
              average);
printf("\nElements of greater
            than average:\n");
for(i=0;i<n;i++)
   if(a[i]>average)
       printf("%6d,",a[i]);
printf("\nElements of less than
              average:\n");
for(i=0;i<n;i++)
   if(a[i]<=average)
       printf("%6d,",a[i]);
getch();
}
```

خروجی

```
Number of numbers(n<21): 10
Enter 10 numbers:
52
12
85
-6
51
9
1
45
121
80
Average= 42.00
Elements of greater than average:
    52,    85,     45,   121,    80,
Elements of less than average:
   121,    -6,    21,       9,      1,
```

آرایه های چند بعدی

- آرایه‌های چندبعدی نیز مشابه آرایه‌های یک‌بعدی تعریف می‌گردند.
- در اینجا به تعداد اَبعاد کروشه داریم و طول هربُعد بین کروشه های آن بُعد آمده است:

storage_class data_type array_name[sizeD*1*][sizeD*2*]... [sizeD*n*] ;

برای مثال تعریف یک آرایه ی دو بعدی m×n به صورت زیر خواهد بود:

```
int a[m][n];
```

آرایه دوبعدی m×n (شامل m سطر و n ستون) را می‌توان به صورت شکل زیر نشان داد:

	ستون ۱	ستون ۲	ستون ۳	...	ستون n
سطر ۱	a[0][0]	a[0][1]	a[0][2]	...	a[0][n -1]
سطر ۲	a[1][0]	a[1][1]	a[1][2]	...	a[1][n -1]
.	.	.	.		.
.	.	.	.		.
.	.	.	.		.
.	.	.	.		
سطر m	a[m -1][0]	a[m -1][1]	a[m - 1][2]	...	a[m -1][n -1]

دادن مقدار اولیه به آرایه های چند بعدی هنگام تعریف:

```
int  array[3][4] = {
                     {1 , 2 , 3 , 4} ,
                     {5 , 6 , 7 , 8} ,
                     {9 , 10 , 11 , 12}
                   } ;
```

1	2	3	4
5	6	7	8
9	10	11	12

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

1	2	3	4
5	6	7	0
8	9	10	0
0	0	0	0

انتقال آرایه به یک تابع

- برای گذر دادن یک آرایه به یک تابع باید فقط نام آن بدون کروشه و بدون اندیس ، به عنوان آرگومان واقعی در فراخوانی تابع ظاهر گردد.

```
main( )
  {
     int  n ;                          /* variable declaration */
     float  avg ;                      /* variable declaration */
     float  list [100] ;                          /* array definition */
     float  average ( ) ;          /* function declaration */
     ....
     avg = average (n , list) ;
     ....
  }
 float average (int a ,float x[ ])          /* function definition */
 {
        ....
 }
```

فصل هفتم : آرایه – انتقال آرایه به یک تابع

- برای ارسال آرایه به تابع ، باید نام تابع به عنوان آرگومان ذکر شود. در این صورت پارامتر معادل آن می تواند به سه صورت زیر تعریف شود:
  - آرایه ای با طول مشخص.
  - آرایه با طول نامشخص که در این صورت بهتر است طول آرایه به عنوان آرگومان دیگری منتقل شود.
  - اشاره گر (این روش در فصل بعد بررسی خواهد شد).

مثال: به برنامه مقابل توجه کنید:

f1 با روش ۱ و f2 با روش ۲ تعریف شده اند.

```
void f1(int x[ ]);
void f2(int x[ ] , int len);
void main()
{
  int x[5];
  …
  f1(x);
  f2(x,5);
}
void f1(int x[5])
{
  …
}
void f2(int x[ ], int len)
{
  …
}
```

مثال : چون ارسال آرایه به داخل تابع از نوع ارسال با ارجاع (فراخوانی با آدرس) است پس هر تغییری درآرایه ایجاد شود نتیجه به تابع فراخواننده نیز منعکس خواهد شد.

```
#include<stdio.h>
```

خروجی

```
Print array data in main before calling the func:
   0,   0,   0,   0,   0,
Print array data in the func:
   0,   1,   2,   3,   4,
Print array data in main after calling the func:
   0,   1,   2,   3,   4,
```

```
{
   printf("Print array data in the func:\n");
   for(int i=0 ; i<5 ; i++)
    printf("%4d,",x[ i ]=i);
}
```

آرایه ها و رشته ها

- در زبان C رشته‌ها ، به عنوان آرایه‌ای از کاراکترها تعریف می شوند.
- در C برای تعیین انتهای رشته از کاراکتر خاصی بنام تهی (‘\0’) استفاده می شود. این کاراکتر در آخر تمام ر شته وارد خواهد شد ، پس هنگام تعریف باید طول رشته را یک کاراکتر بیشتر از اندازه مورد نیاز تعریف کنید.

```
char str[10] = “abc”;
```

S[0]	S[1]	S[2]	S[3]	S[4]	S[5]	S[6]	S[7]	S[8]	S[9]
a	b	c	\0	?	?	?	?	?	?

مثال: قطعه برنامه زیر اسامی ۵ نفر را به آرایه رشته‌ایname می‌خواند و آنها را در یک سطر چاپ می‌کند .

```
# include<stdio.h>
main ( )
{
   char  name[5][12] ;
   int  i ;
   for ( i = 0 ;  i<5 ; ++i )
       scanf ("%s" , &name [i]) ;
   for ( i = 0 ; i<5 ; ++i )
       printf (“ %s," , name [i]);
   }
```

خروجی

```
ali
hosein
behrooz
mohammad
saeed

ali, hosein, behrooz, mohammad, saeed,
```

روشهای مرتب سازی

- یکی از کاربردهای متداول آرایه ها ، استفاده از آنها در روشهای مرتب سازی است.
- تعداد مقایسه ها و نیز تعداد جابجایی عناصر، از عوامل اساسی در مورد سرعت مرتب سازیها می باشند .

مرتب سازی حبابی (Bubble Sort)

تابع زیر آرایه n عنصری A را به روش حبابی ، مرتب می نماید :

```
void  Bubble_Sort (int A[ ] , int  n)
{
   int  i , j , temp ;
   for ( i =1 ; i<n ; + +i )
      for ( j =0 ; j<n-i ; + +j )
         if ( A[ j] > A[ j+1] )
         {
            temp = A [ j] ;
            A[ j] = A[ j+1] ;
            A[ j+1] = temp ;
          }
 }
```

مثال: برنامه زیر آرایه ی ۱۰ عنصری از اعداد صحیح را بطور تصادفی پر کرده و آن را مرتب می کند:

```
#include<stdio.h>
#include<stdlib.h>
void buble_sort(int [ ], int);
void main()
{
  int length=10,array[length],i;
  randomize();
  printf("Array:\n");
  for(i=0;i<length;i++)
  {
    array[i]=random(100);
    printf("%4d, ",array[i]);
  }
  buble_sort(array,length);
  printf("\nSorted Array:\n");
  for(i=0;i<length;i++)
    printf("%4d, ",array[i]);
}
```

```
void buble_sort(int a[ ], int len)
{
  int i,j,temp;
  for ( i =1 ; i<len ; ++i )
    for ( j =0 ; j<len-i ; ++j )
      if ( a[ j] > a[ j+1] )
      {
        temp = a [ j] ;
        a[ j] = a[ j+1] ;
        a[ j+1] = temp ;
      }
}
```

خروجی

```
Array:
 81,  37,  64,  63,  52,  12,  89,  62,  80,  58,
Sorted Array:
 12,  37,  52,  58,  62,  63,  64,  80,  81,  89,
```

پیاده سازی مرتب سازی حبابی:

12 > 2 do change

12 > 8 do change

12 > 6 do change

12 > 1 do change

```
for ( i =1 ; i<n ; + +i )
      for ( j =0 ; j<n-i ; + +j )
        if ( A[ j] > A[ j+1] )
        {
           temp = A [ j] ;
           A[ j] = A[ j+1] ;
           A[ j+1] = temp ;
         }
```

temp

مرحله ۱

پیاده سازی مرتب سازی حبابی:

A[0]	A[1]	A[2]	A[3]	A[4]
2	8	6	1	12

2 < 8 don't change 8 > 6 do change 8 > 1 do change

```
for ( i =1 ; i<n ; + +i )
    for ( j =0 ; j<n-i ; + +j )
      if ( A[ j] > A[ j+1] )
      {
         temp = A [ j] ;
         A[ j] = A[ j+1] ;
         A[ j+1] = temp ;
       }
```

temp

مرحله ۲

پیاده سازی مرتب سازی حبابی:

2 < 6 6 > 1

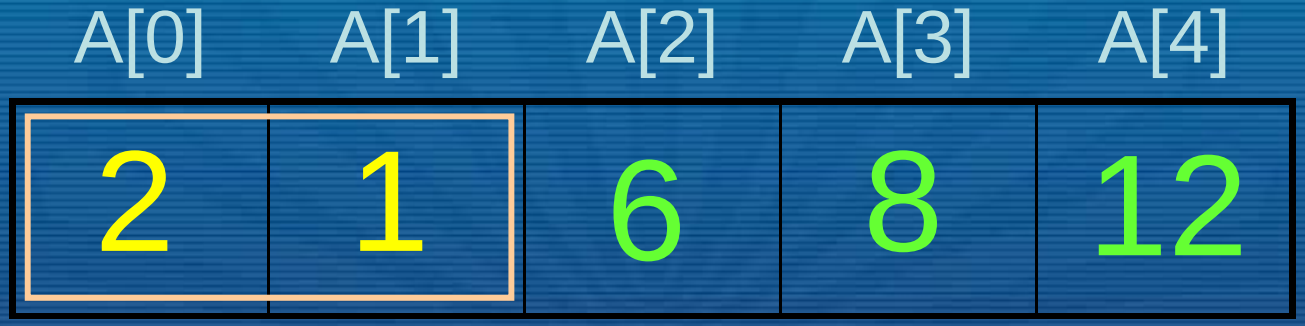
don't change do change

```
for ( i =1 ; i<n ; + +i )
      for ( j =0 ; j<n-i ; + +j )
        if ( A[ j] > A[ j+1] )
        {
           temp = A [ j] ;
           A[ j] = A[ j+1] ;
           A[ j+1] = temp ;
         }
```

temp

مرحله ۳

پیاده سازی مرتب سازی حبابی:



2 > 1

do change

آرایه ی مرتب شده

temp

مرحله ۴

```
for ( i =1 ; i<n ; + +i )
      for ( j =0 ; j<n-i ; + +j )
        if ( A[ j] > A[ j+1] )
        {
           temp = A [ j] ;
           A[ j] = A[ j+1] ;
           A[ j+1] = temp ;
         }
```

روش مرتب سازی انتخابی (Selection Sort)

پیچیدگی زمانی این روش بهتر از مرتب سازی حبابی است، چون در هر مرحلـه حد اکثر یک جابجایی صورت می گیرد.

```
void selection_sort (int a[ ], int len)
{
   int i,j,temp,max,exchange;
   for(i=0 ;i< len-1; i++)
   {
      exchange=0;
      max=i;
      for(j=i+1 ; j<len ; j++)
        if(a[j] < a[max] )
        {
            max=j;
            exchange=1;
        }
      if (exchange)
      {
          temp = a[i] ;
          a[i] = a[max] ;
          a[max] = temp ;
      }
   }
}
```

مثال: برنامه زیر آرایه ی ۱۰ عنصری از اعداد صحیح را بطور تصادفی پر کرده و آن را بروش انتخابی مرتب می کند:

```
#include<stdio.h>
#include<stdlib.h>
#define length 10
void selection_sort(int [ ], int);
void main()
{
  int array[length],i;
  randomize();
  printf("Array:\n");
  for(i=0;i<length;i++)
  {
    array[i]=random(100);
    printf("%4d, ",array[i]);
  }
  selection_sort(array,length);
  printf("\nSorted Array:\n");
  for(i=0;i<length;i++)
    printf("%4d, ",array[i]);
}
```

```
void selection_sort (int a[ ], int len)
{
  int i,j,temp,max,exchange;
  for(i=0 ;i< len-1; i++) {
     exchange=0;
     max=i;
     for(j=i+1 ; j<len ; j++)
          if(a[j] < a[max] ) {
               max=j;
               exchange=1; }
      if (exchange){
          temp = a[i] ;
          a[i] = a[max] ;
          a[max] = temp ; }
     }
}
```

خروجی

```
Array:
   93,  95,   8,  98,  77,  22,   0,  11,    7,   96,
Sorted Array:
    0,   7,   8, 11,   22, 77,  93,  95,   96,   98,
```

روشهای جستجو

- هر گاه در داخل مجموعه‌ای از عناصر، دنبال عنصر خاصی بگردیم، این عمل را جستجو کردن یا searching می نامند .

جستجوی خطی

- در این روش کلید مورد جستجو با اولین عنصر آرایه مقایسه می شود ، اگربرابر باشند اندیس عنصر را برمی گردانیم و جستجو با موفقیت به پایان می رسد ، وگر نه جستجو را با عنصر بعدی آرایه ادامه می دهیم تا وقتی که کلید را پیدا کنیم و یا به آخر آرایه برسیم.

- تابع جستجوی خطی :

```
int linear_search(int a[ ], int len, int key)
{
  for(int i=0 ; i<len ; i++)
      if(a[i] == key)
          return i ;
  return -1 ;
}
```

مثال : برنامه زیر آرایه ی ۱۰ عنصری از اعداد صحیح را بطور تصادفی پر کرده و آن را نمایش می دهد، سپس یک عدد از ورودی گرفته و در آرایه جستجو می کند:

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#define length 10
int linear_search(int [ ], int, int);
void main()
{
  int array[length],i,target;
  randomize();
  printf("Array:\n");
  for(i=0;i<length;i++)
  {
    array[i]=random(100);
    printf("%4d, ",array[i]);
  }
  printf("\nEnter a number to search for:");
  scanf("%d",&target);
  i=linear_search(array,length,target);
  if(i==-1)
      printf("\nNumber not found.");
  else
      printf("\nNumber find in position %d.",i+1);
  getch();
}

int linear_search(int a[ ], int len, int key)
{
  for(int i=0 ; i<len ; i++)
      if(a[i] == key)
          return i ;
  return -1 ;
}
```

خروجی

```
Array:
   35,  1,  5, 48,  41, 42, 48, 34,  6,  1,
Enter a number to search for: 1
Number find in position 2.
```

ملاحظه می کنید که تابع نوشته شده فقط اولین وقوع کلید مورد جستجو را گزارش می کند؛ اگر بخواهیم نتیجه جستجو شامل همه وقوع های کلید (در صورت تکراری بودن کلید داخل آرایه) باشد، باید تغییراتی در بدنه تابع انجام دهیم.

جستجو به روش دودویی

- این روش جستجو به مراتب از جستجوی خطی بهتر و سریع تر عمل می کند.
- تنها عیب این روش این است که آرایه قبل از جستجو باید مرتب شده باشد وگرنه این جستجو ممکن است نتیجه منطقی ندهد.
- می دانیم که اگر فقط یک بار آرایه رامرتب کنیم ، می توانیم چندین بار با این روش روی آن جستجو انجام دهیم.

تابع جستجوی دودویی :

```
int  BinarySearch (int  a[ ] , int  len , int  key)
{
   int   middle , Low , High  ;
   Low = 0 ;
   High = len-1 ;
   while ( Low <= High)
   {
      middle = (Low+High)/2 ;
      if ( key == a[middle] )  // indicate that result is found
         return (middle +1) ; // return the index of the key
      if ( key > a[middle] )    // key doesn't exist in left hand
         Low = middle +1 ;   // go and search right hand
     else                          // key doesn't exist in right hand
         High = middle -1 ;   // go and search left hand
   }
   return(0) ;                    // indicate that result isn't found
}
```

مثال : مراحل الگوریتم جستجو به روش دودویی برای ده عدد زیر، در جدول نمایش داده شده است:

65 , 141 , 192 , 205 , 218 , 389 , 424 , 500 , 538 , 567

جستجوی عدد ۵۶۷				جستجوی عدد ۲۰۵			
X	l	h	m	X	l	h	m
218	1	10	5	218	1	10	5
500	6	10	8	141	1	4	2
538	9	10	9	192	3	4	3
567	10	10	10	205	4	4	4

توابع کتابخانه ای رشته ها

- توابع کتابخانه‌ای مربوط به رشته‌ها در فایل string.h قرار دارند .

جدول توابع کتابخانه ای رشته ها

نام تابع	عمل تابع
strcpy (s1 , s2)	رشته s2 را روی رشته s1 کپی می‌کند .
strcat (s1 , s2)	رشته s2 را به دنبال رشته s1 ملحق (ضمیمه) می‌کند.
strlen (s)	طول رشته s را برمی‌گرداند .
strcmp (s1, s2)	رشته s1 را با رشته s2مقایسه می‌کند .

مثال : برنامه زیر اسامی 5 دانشجو را از ورودی خوانده و آنها را مرتب کرده و چاپ می کند:

```
# include<stdio.h>
# include<string.h>
void main ( )
{
  int  i , j ;
  char  name [5][12] , temp [12] ;
  for ( i=0 ; a<5 ; ++i )
     scanf ("%s" , &name[i] ) ;
  printf("\n\nSorted list:\n");
  for ( i=1 ; i<5 ; ++i )
     for ( j=0 ; j<4 ; ++j )
        if (strcmp (name [j] , name [j+1]) > 0 )
           {
             strcpy (temp , name [j] ) ;
             strcpy (name[j] , name [j+1] ) ;
             strcpy (name[j+1] , temp) ;
           }
  for ( i=0 ; i<5 ; ++i )
     printf ("\n %s" , name [i] ) ;
}
```

خروجی

```
saeed
behrooz
ali
mohammd
pirooz

Sorted list:
 ali , behrooz , mohammad ,
 pirooz , saeed ,
```

اصول کامپیوتر ۲

رشته علوم کامپیوتر

۴ واحد درسی

فصل هشتم

نام منبع و مؤلف :

- ✓ اصول کامپیوتر ۲
- ✓ دکتر داود کریم زادگان مقدم
- ✓ انتشارات دانشگاه پیام نور ۱۳۸۳

فصل هشتم : اشاره گر

- تعریف اشاره گرها و متغیرهای اشاره گر
- ارتباط اشاره گرها و آرایه ها
- اشاره گرها و رشته ها
- آرایه ای از اشاره گرها
- اشاره گر به اشاره گر
- اشاره گرها و آرگومان های توابع
- برگرداندن اشاره گر از توابع
- اشاره گر به توابع
- آرایه ای از اشاره گر به توابع

هدف کلی

آشنایی با اشاره گرها و کاربردهای مختلف اشاره‌گرها در برنامه سازی با زبان C

هدف های رفتاری

- تعریف اشاره گر و نحوه اعلان و مقداردهی اشاره گر در برنامه
- آشنایی با رابطه بین اشاره گر و آرایه و متغیر در برنامه
- مفهوم و کاربرد توابع با آرگومان اشاره‌گر
- مزیت‌ها و محدودیت‌های استفاده از اشاره گرها
- آشنایی با انواع عملیات بر روی اشاره‌گرها

مقدمه Introduction

- اشاره‌گر ، متغیری است که آدرس متغیر دیگری را در خود نگه می‌دارد.
- اشاره گر روش غیر مستقیم دسترسی به داده هاست.
- اغلب قابلیتهای C به نقش اشاره گرها در این زبان برمی گردد، و جود اشاره گرها باعث شده که بتوان عملیاتی نظیر عملیات زبان اسمبلی را در C انجام داد.

- استفاده از اشاره گرها در C ، قابلیتهای زیر را فراهم می کند:
  - انتقال آدرس متغیرها به تابع فرعی
  - برگرداندن چندین مقدار از تابع فرعی
  - دستیابی به عناصر آرایه‌ها
  - تشکیل ساختارهای پیچیده‌تر مانند : لیستهای پیوندی ، درختها و گرافها
  - عمل تخصیص حافظه به صورت پویا

تعریف اشاره‌گر

روش کلی تعریف متغیری از نوع اشاره‌گر به صورت زیر است :

```
data_type  * pointer_variable ;
```

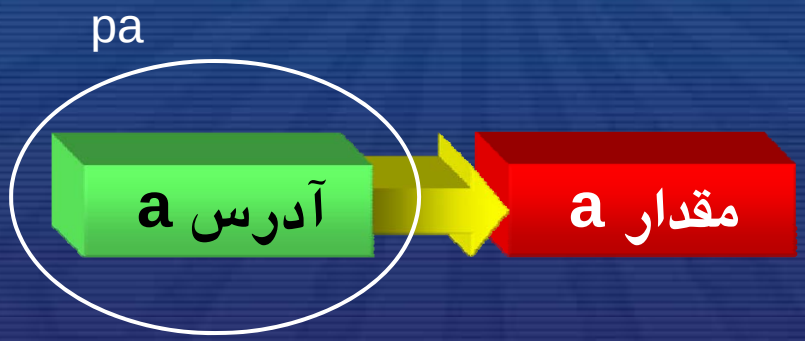
- pointer_variable متغیری است که میتواند آدرس متغیر دیگری از نوع data_type را نگه دارد.
- آدرس محل متغیر a در حافظه بوسیلهٔ عبارت &a تعیین می‌گردد؛ عملگر & یک عملگر تک عملوندی است که آدرس عملوند خود را بر می گرداند.

به تعاریف مقابل توجه نمایید :

```
int  a , *pa ;
```

بعد از تعاریف بالا عبارت مقابل آدرس متغیر a را در pa قرار می دهد و در نتیجه pa به متغیر a اشاره می کند:

```
pa = &a ;
```



- عملگر دیگر مربوط به اشاره گرها عملگر * است که فقط با متغیرهای از نوع اشاره گر به کاربرده می شود، و مقدار محلی که اشاره گر به آنجا اشاره می کند را بر می گرداند.

- اگر تعاریف مقابل را داشته باشیم :

```
int a=2 , k , *p ;
p = &a ;
k = *p ;
```

مقادیر متغیرهای k و a برابر خواهد بود.

آدرس متغیر

- هر متغیر دارای آدرس منحصر به فردی است که محل آن متغیر را در حافظه مشخص می‌کند.
- گاهی بهتر است برای دسترسی به یک متغیر از آدرس آن به جای نام آن استفاده کنیم.

مثال: برنامه ساده زیر مقدار و آدرس متغیر A را چاپ می‌کند:

```
# include <stdio.h>
void main ( )
 {
    int  A = 5 ;
    printf (" The  value of  A  is : %d\n" , A) ;
    printf (" The  address of  A is : %p\n" , &A) ;
  }
```

خروجی

```
The  value of  A is : 5
The  address of  A  is : FFF4
```

مقداردهی اولیه اشاره‌گر

- یک اشاره گر می تواند مثل هر متغیر دیگری هنگام تعریف مقدار اولیه بگیرد ولی مقدار اولیه آن باید یک آدرس باشد.
- یک اشاره گر می تواند با NULL نیز مقداردهی شود ، در این صورت اشاره گر به هیچ جایی اشاره نمی کند.
- NULL در فایلهای سرآیند ,alloc.h ,stdio.h, mem.h, stddef.h, و stdlib.h تعریف شده است.

اشاره‌گر تهی

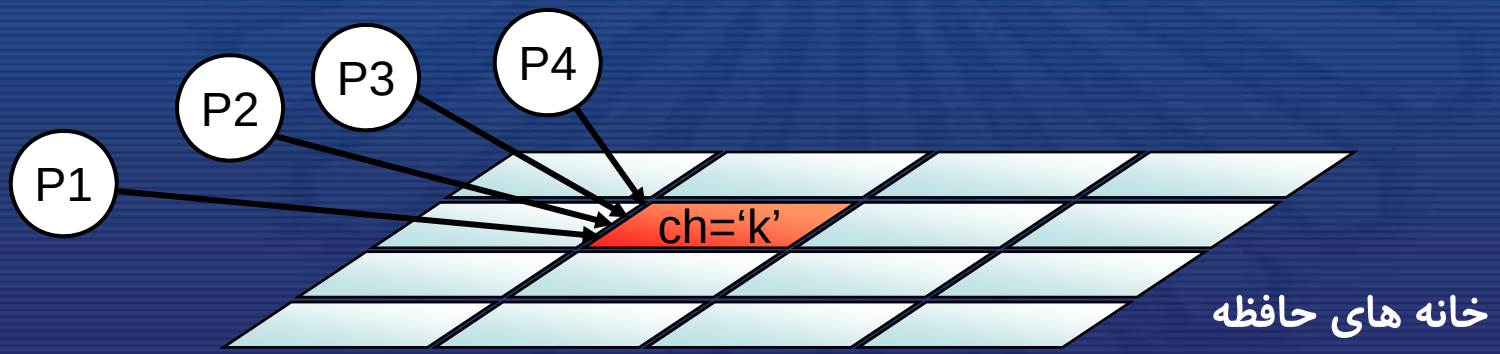
- قبلا اشاره شد که زبان C مفهوم اشاره‌گرNULL را پشتیبانی می‌کند و آن اشاره‌گری است که به هیچ شئی قابل قبول یا معتبر اشاره نمی‌کند.
- اشاره‌گرهایی با مقدار صفر یا NULL بعنوان false درنظر گرفته می‌شوند.
- اشاره‌گرهایNULL به ویژه در دستورهای مربوط به کنترل جریان مفید هستند .زیرا اشاره‌گرهایی با مقدار صفر بعنوان false درنظر گرفته می‌شوند ، درحالی که متغیرهای اشاره‌گر با سایر مقادیر ، true منظور می‌گردند.

عملیات روی اشاره‌گرها

انتساب اشاره گرها

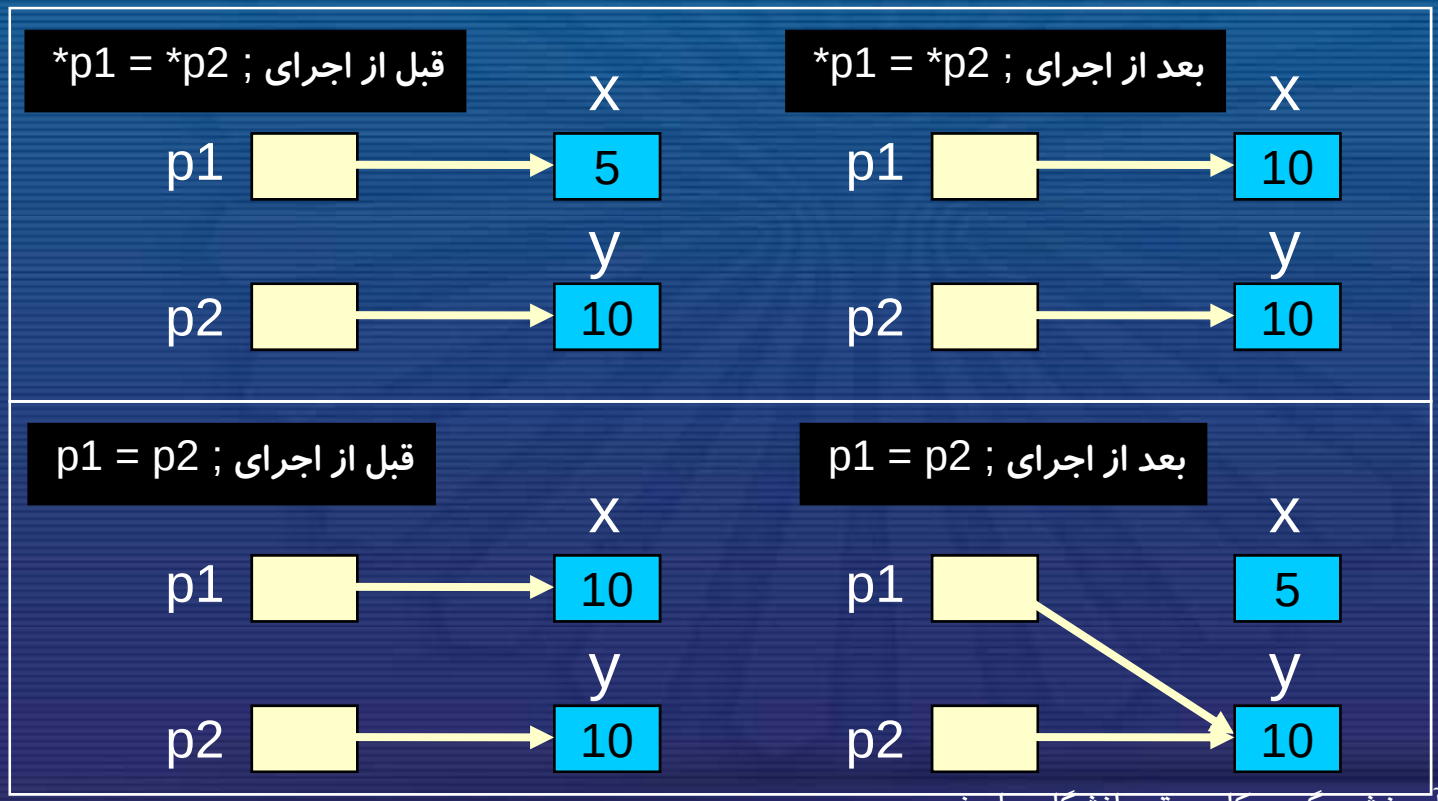
اگر مقدار چند اشاره گر برابر باشند، همه ی آنها آدرس یک خانه از حافظه را دارند، به عبارت دیگر همگی به یک محل خاص از حافظه اشاره خواهند کرد.

```
char ch='k',*P1,*P2,*P3,*P4 ;
P1=P2=P3=P4=&ch;
```



نتیجه دستورات و انتساب های مقابل در شکل نشان داده شده است : `int x, y, *p1,*p2 ;`

```
x = 5 ;
y = 10 ;
p1 = &x ;
p2 = &y ;
*p1 = *p2 ;
p1 = p2 ;
```



اعمل محاسباتی بر روی اشاره گرها

- زبان C ، اجازه می‌دهد که یک مقدار صحیح را به یک اشاره‌گر اضافه یا کم کنید:

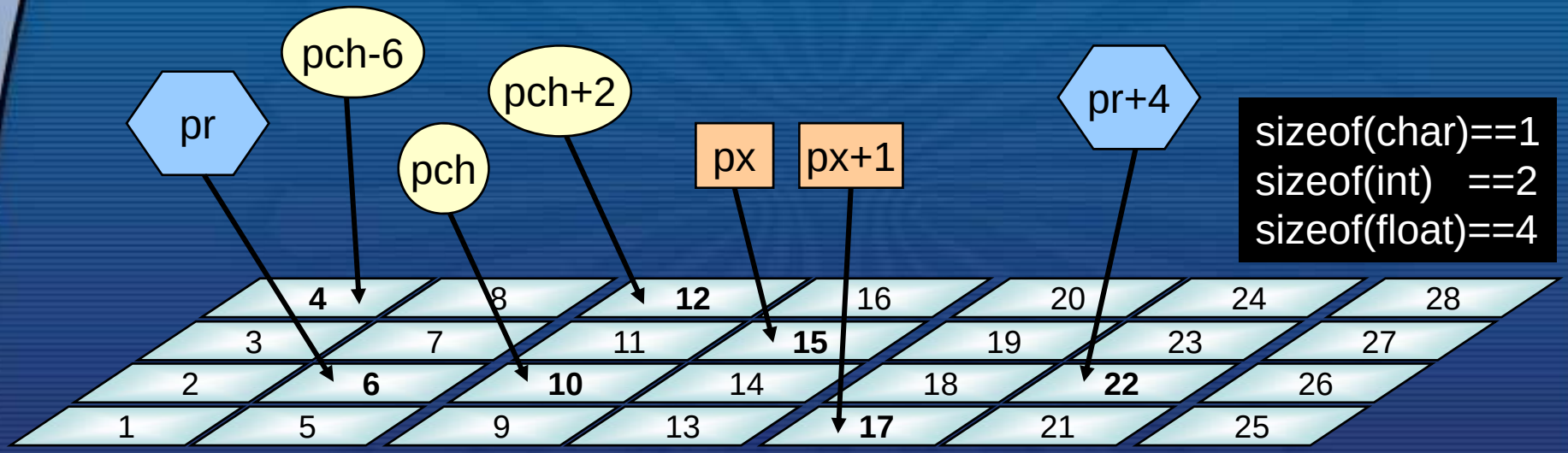
$$p = p + 2$$

- در این صورت، اشاره گر p به ۲ محل بعد از آدرس کنونی اشاره خواهد کرد.

- منظور از ۲ محل، ۲ شیئ از نوعی که p به آن اشاره می کند است.

فرض کنید شکل زیر قسمتی از حافظه سیستم باشد که به خانه های (سلول های) یک بایتی تقسیم شده و شماره های روی خانه ها نشان دهنده ترتیب پر شدن آنها باشد؛ در این صورت داریم :

```
char  *pch=10;
int  *px=15;
float  *pr =6;
```



pr = 6
pr + 4 = 6 + (sizeof(float) * 4) = 6+16 = 22

pch = 10
pch-6=10 – (sizeof(char)*6) = 10 - 6 =4

مقایسه اشاره گرها

- دو اشاره‌گر را می‌توان در یک عبارت رابطه‌ای با یکدیگر مقایسه کرد، به شرطی که هر دو، اشاره گر به یک نوع داده باشند.
- برای مثال دستور زیر معتبر است به شرطی که px وpy هردو اشاره گر به یک نوع داده (مثلا int) باشند.

```
if  (px<py)
   printf ("px  points to lower memory than  py") ;
else
   printf ("px  points to upper memory than  py") ;
```

انتقال اشاره‌گر به تابع

- وقتی که یک اشاره‌گر به یک تابع گذر داده می‌شود، در واقع آدرس آن قلم از داده به تابع فرستاده می‌شود .
- هر گونه تغییراتی که در محتوای آدرس مورد نظر انجام گیرد، هم در تابع فراخوانده شده و هم در برنامه یا تابع فراخواننده تأثیر می‌گذارد .
- پارامتر متناظر تابعی که یک آدرس را به عنوان آرگومان دریافت می‌کند ، باید یک اشاره‌گر باشد.

مثال : افزایش مقدار یک متغیر به وسیلهٔ فراخوانی با آدرس:

خروجی

```
the original value of count is 7
the new value of count is 8
```

```
# include <stdio.h>
void add (int *) ;
void main ( )
 {
   int  count = 7 ;
   printf("the original value of count is %d\n" , count) ;
   add (& count) ;
   printf ("the new value of count is %d\n" , count) ;
 }
void  add (int  *countptr)
 {
    ++ (*countptr) ;       /* increments count in main */
  }
```

مثال: برنامه زیر، تفاوت بین آرگومانهای معمولی که بوسیله مقدارعبور داده می شوند و آرگومانهای اشاره گر را که بوسیله مرجع عبور داده می شوند روشن می سازد .

```
#include<stdio.h>
#include<conio.h>
void  func1 ( int  u , int  v ) ;
void  func2 ( int  *pu , int  *pv ) ;
void main ( )
{
   int  u = 1 ;
   int  v = 3 ;
   printf (“\n Before  calling  func1  : u=%d   v=%d " , u , v )  ;
   func1(u , v ) ;
   printf (“\n After  calling  func1  : u=%d   v=%d " , u , v )  ;
   printf (“\n Before  calling  func2  : u=%d   v=%d " , u , v )  ;
   func2(&u , &v ) ;
   printf (“\n After  calling  func2  : u=%d   v=%d " , u , v )  ;
   getch();
}
```

```
void  func1 ( int  u , int  v )
{
   u = 0 ;
   v = 0 ;
   printf (“\n Within  func1  : u=%d   v=%d " , u , v )  ;
   return ;
}
void  func2 ( int  *pu , int  *pv )
{
   *pu = 0 ;
   *pv = 0 ;
   printf (“\n Within  func2  : *pu=%d   *pv=%d " , *pu , *pv )  ;
   return ;
}
```

خروجی

```
Before  calling  func1  :   u = 1    v = 3
Within  func1  :              u = 0    v = 0
After  calling  func1  :     u = 1    v = 3
Before  calling  func2  :   u = 1    v = 3
Within  func2  :           *pu = 0   *pv = 0
After  calling  func2  :     u = 0    v = 0
```

انتقال دوطرفه اطلاعات

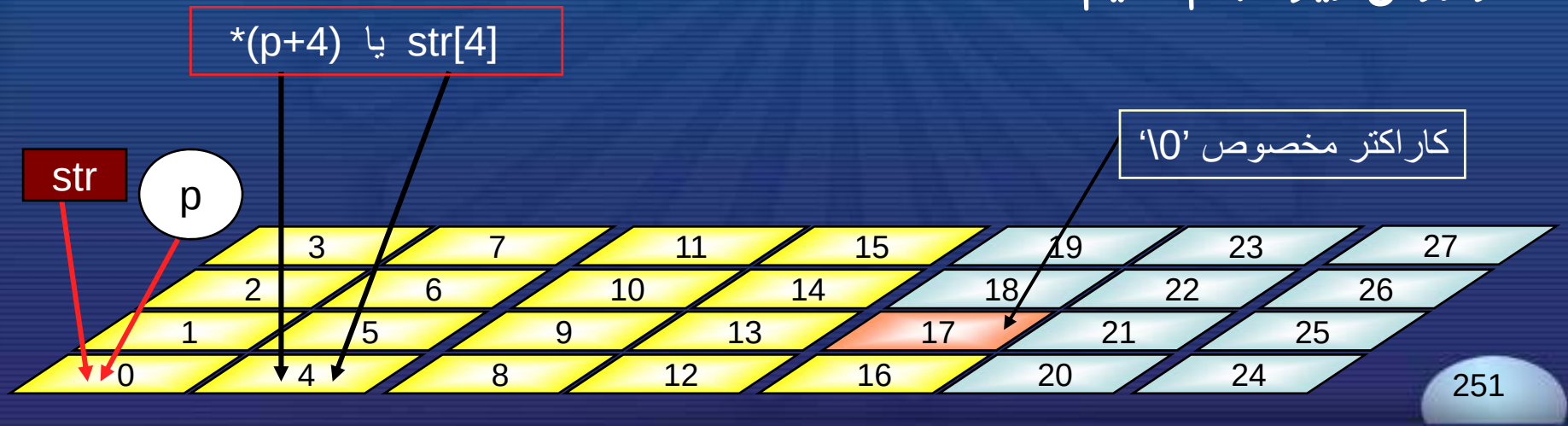
به کمک اشاره‌گرها می‌توان مقادیر را هم از برنامه فراخواننده به تابع فراخوانده شده و هم از تابع فراخوانده شده به برنامه فراخواننده آن (درواقع در هر دو جهت) گذر داد.

اشاره‌گر و آرایه

- بین اشاره‌گرها و آرایه‌ها رابطه نزدیکی وجود دارد . نام یک آرایه در واقع اشاره گری به اولین عنصر آرایه است.

```
char str[18] , *p ;
p = str;
```

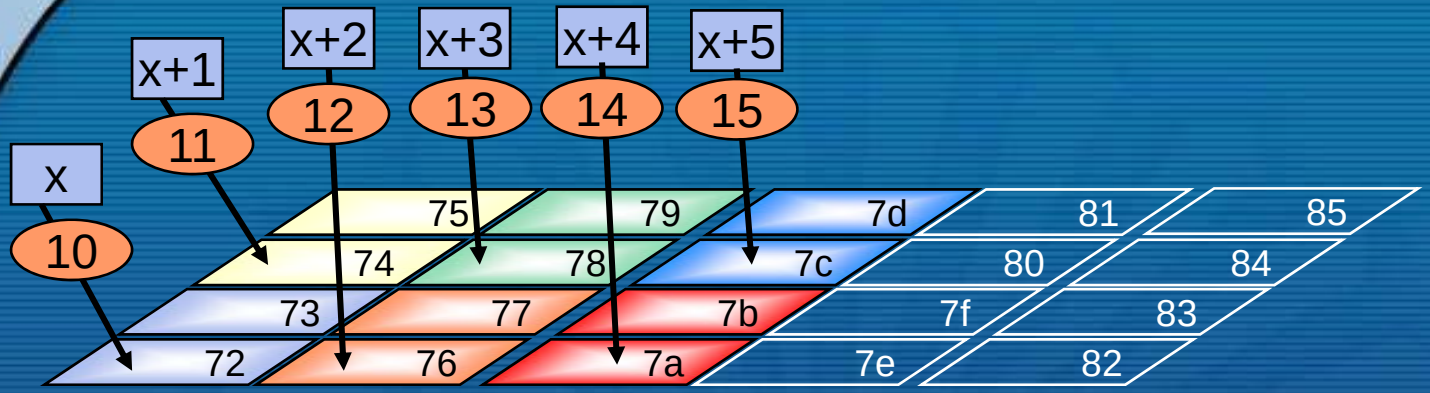
- اگر بخواهیم به پنجمین عنصر در str دسترسی داشته باشیم ، می‌توانیم این کار را به دو روش زیر انجام دهیم :



مثال: برنامه زیر را درنظر بگیرید

```
#include <stdio.h>
void main ( )
{
  static int x[6] = {10 , 11 , 12 , 13 , 14 , 15};
  int  i ;
  for ( i=0 ; i<6 ; + +i )
     printf("\n i =%d  x[i] = %d   *(x+i) = %d  &x[i] = %x  x+i =%x", i , x[i] , *(x+i) , &x[i] , x+i) ;
}
```

با فرض بر این که آدرس شروع آرایه ، ۷۲ در مبنای ۱۶ است، خروجی بصورت جدول اسلاید بعد خواهد بود.



خروجی

i = 0	x[i] = 10	*(x+i) = 10	&x[i] = 72	x+i = 72
i = 1	x[i] = 11	*(x+i) = 11	&x[i] = 74	x+i = 74
i = 2	x[i] = 12	*(x+i) = 12	&x[i] = 76	x+i = 76
i = 3	x[i] = 13	*(x+i) = 13	&x[i] = 78	x+i = 78
i = 4	x[i] = 14	*(x+i) = 14	&x[i] = 7a	x+i = 7a
i = 5	x[i] = 15	*(x+i) = 15	&x[i] = 7c	x+i = 7c

تخصیص حافظه پویا

- در C این امکان هست که آرایه را به صورت یک متغیر اشاره گر تعریف کرد و در جریان اجرای برنامه بطور پویا (هر اندازه که نیاز باشد) از سیستم حافظه درخواست کرد.
- برای این کار از تابع malloc() استفاده می کنیم:

```
pointer_variable = malloc (size)
```

مثال : در قطعه برنامه مقابل به اندازه آرایه ی ۵ عنصری از نوع int حافظه تخصیص یافته است :

```
int *a ;

a = (int *) malloc (sizeof(int) * 5) ;
```

وقتی آرایه ای به روش تخصیص حافظه پویا ایجاد شود ، امکان دادن مقدار اولیه به آن وجود ندارد.

مثال: برنامه زیر یک مجموعه از اعداد را از ورودی خوانده و با استفاده از اشاره گرها مرتب می کند.

```
#include<stdio.h>
#include<stdlib.h>
void main ( )
 {
   int  k , m , *a ;
   void  sort (int k , int  *a) ;
   scanf ("%d" , &m) ;                    /* read in a value for m */
   a = (int*) malloc (m * sizeof(int)) ;  /* allocate memory */
   for (k = 0 ; k<m ; ++k)                /* read in the list of numbers */
      scanf ("%d" , a + k) ;
   sort (m , a) ;
   for (k=0 ; k<m ; ++k)                  /* display sorted list , elements */
      printf ("\n k=%d  a =%d" , k+1 , *(a+k)) ;
  }
```

```
void  sort (int m , int *a)    /* sort array in ascending order */
{
  int  i , j , temp ;
  for ( i=1 ; i<m ; ++i )
     for ( j=0 ; j<m-i ; ++j )
          if (*(a+j) < *(a+j+1))
          {
              temp = *(a+j) ;
              *(a+j) = *(a+j+1) ;
              *(a+j+1) = temp;
          }
   return ;
}
```

خروجی

```
5
4
1
12
8
6
k=1  a=12
k=2  a=8
k=3  a=6
k=4  a=4
k=5  a=1
```

در برنامه بالا از تخصیص حافظه پویا ، فرستادن اشاره گر به تابع ، استفاده از اشاره گر در دستورات ورودی و خروجی و دسترسی به خانه های متوالی در حافظه با استفاده از اشاره گر و... استفاده شده است.

اشاره‌گر و آرایه‌های چند بعدی

- عناصر یک آرایهٔ یک بعدی می‌تواند برحسب یک اشاره‌گر (نام آرایه) و یک مقدار به عنوان آفست به منظور جبران کردن مقدار اندیس عنصر مورد نظر آرایه ، نمایش داده شود.

- می‌توان یک آرایه دو بعدی را به صورت یک اشاره‌گر به یک گروه پیوسته و مجاور هم از آرایه‌های یک بعدی تعریف کرد.

درنتیجه می‌توان توصیف یک آرایه دو بعدی را بجای:

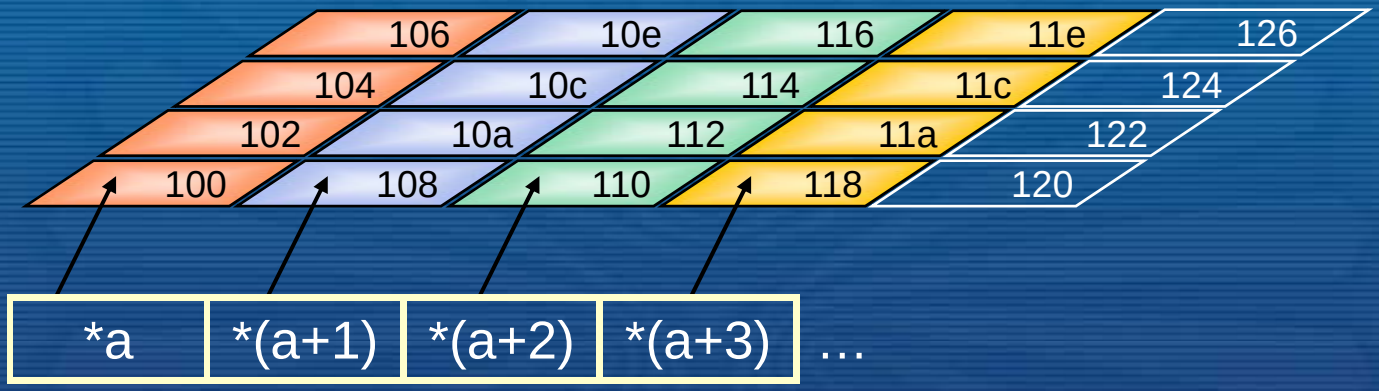
```
data_type    array[d1][d2] ;
```

به صورت زیر نوشت:

```
data_type   (*ptvar)[d2] ;
```

این ایده را می‌توان به آرایه‌های n بعدی هم تعمیم داد.

```
int (*a)[4] ;
```



آرایه ای از اشاره گرها که به طول دلخواه می تواند انتخاب شود.

انتقال آرایه به تابع

- قبلا گفته شد که نام آرایه اشاره گر به اولین خانه (و بطور کلی به خود آرایه) است، پس می توان بجای فرستادن آرایه به تابع ، اشاره گری به آرایه را به تابع گذر دهیم. (تا بحال هم همین کار را کرده ایم)
- برای انجام این کار در تابع فرعی نیاز است که ما آرگومان را بعنوان اشاره‌گر به اولین عنصر آرایه توصیف کنیم . برای این کار ، دو راه به صورت زیر وجود دارد :

روش اول:

```
func(float *ar)
 {
    .......
    .......
 }
```

روش دوم:

```
func(float ar[ ])
 {
    .......
    .......
 }
```

آرایه‌ایی از اشاره‌گرها

- در زبان C می توان آرایه‌ای از اشاره‌گرها تعریف کرد. یعنی آرایه‌ای که عناصر آن اشاره‌گر باشند :

```
int  *x[10] ;
```

- برای قراردادن آدرس یک متغیر int مثلا k در خانه ای از آرایه از دستور مقابل استفاده می کنیم :

```
x[2] = &k ;
```

- برای بدست آوردن مقدار k از دستور *x[2] و یا **(x+2) استفاده می کنیم.

مثال: تابع Func می‌تواند آرایه x اسلاید قبل را به صورت زیر دریافت نماید:

```
void  Funk (int  *a[ ] )
{
   int  k ;
   for ( k=0 ; k<10 ; k+ +)
      printf (" %p -> %d" , *a[k] ,**a[k] ) ;
}
```

توجه داشته باشید که در مثال بالا ، a یک اشاره‌گر به مقادیر صحیح نیست بلکه یک اشاره‌گر به آرایه‌ای از اشاره‌گرهایی به مقادیر صحیح است.

اشاره‌گر به اشاره‌گر

- قبلا گفتیم اگر متغیری آدرس متغیر دیگری را در خود نگه دارد آن را اشاره‌گر نامند ؛
- حال اگر متغیر دوم نیز از نوع اشاره‌گر باشد در این صورت متغیر اول یک اشاره‌گر به اشاره‌گر است.

شکلهای زیر مفهوم اشاره‌گر به متغیر عادی و اشاره‌گر به اشاره‌گر را روشن می‌کند :

اشاره گر به متغیر عادی

اشاره گر به اشاره گر به متغیر عادی

- این روش می‌تواند (به‌صورت تودرتو) تا هرچند بار که نیاز باشد ، تکرار گردد.

- برای توصیف متغیرهایی از نوع اشاره‌گر به اشاره‌گر باید دو ستاره در جلوی آن قرار داد.

z اشاره گری به اشارگر به یک مقدار float است: float **z ;

- برای دستیابی به مقدار متغیر هدف که آدرس آن در اشاره‌گر دوم است ، باید اپراتور ستاره را دوباره بکار ببرید .

```
float a ;
a = **z ;
```

```
int  a , *k , **z ;
a = 5;
k = &a ;
z = & k ;
```

در شکل بالا بعد از مجموعه تعاریف، خواهیم داشت:

&z : آدرس یک اشاره گر به اشاره گر به مقدار صحیح

z : آدرس یک اشاره گر به مقدار صحیح

&k : آدرس یک اشاره گر به مقدار صحیح

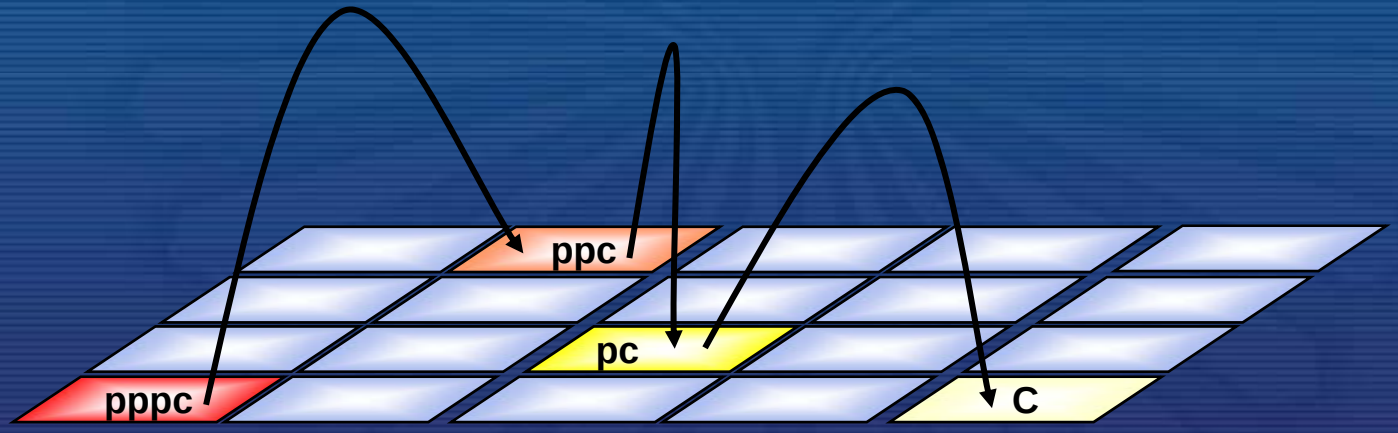
k : آدرس یک مقدار صحیح

&a : آدرس یک مقدار صحیح

a : یک مقدار صحیح

نمایش مفهوم اشاره گر به اشاره گر بر روی حافظه:

```
char c , *pc , **ppc , ***pppc ;
pc = &c ;
ppc = &pc ;
pppc = &ppc ;
```



اصول کامپیوتر ۲

رشته علوم کامپیوتر

۴ واحد درسی

فصل نهم

نام منبع و مؤلف :

- ✓ اصول کامپیوتر ۲
- ✓ دکتر داود کریم زادگان مقدم
- ✓ انتشارات دانشگاه پیام نور ۱۳۸۳

فصل نهم : نوع داده کاربر

- اشتقاق کلی خواص و رفتارها
- مقدمه تعریف شده توسط کاربر
- ساختار داده ها و اشاره گر ها
- آخرین سطر از مقدار خروجی
- چند نمونه از ساختار
- نوع های شمارشی

هدف کلی

آشنایی با انواع داده هایی که توسط کاربر تعریف می شوند و کاربرد آنها در برنامه سازی

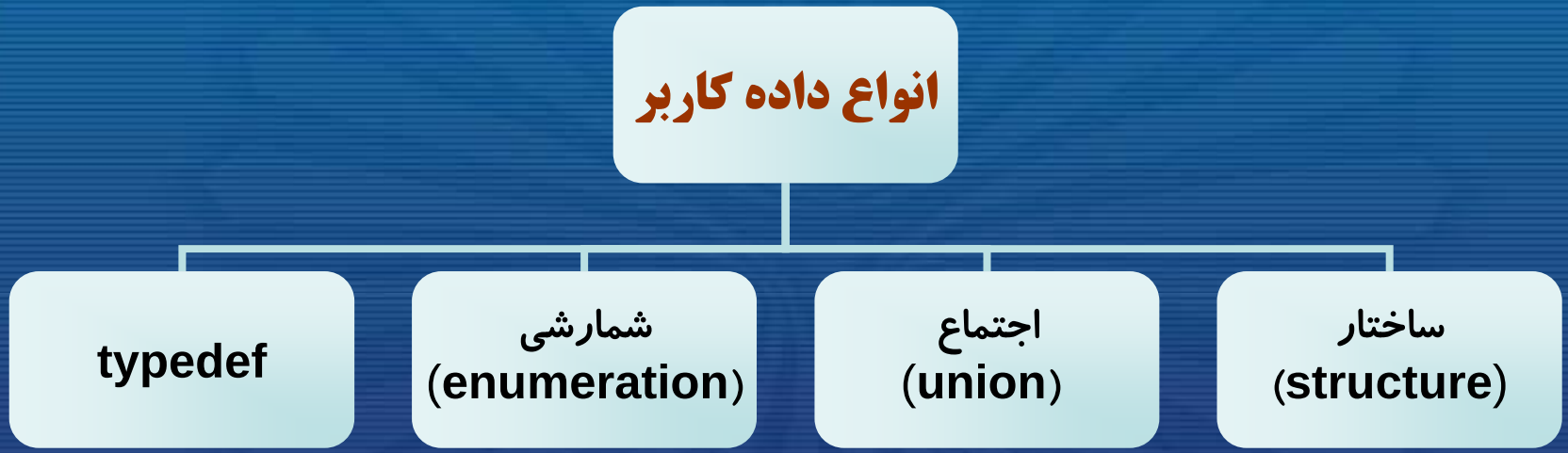
هدف های رفتاری

- تعریف ساختار و نحوه دسترسی به عناصر آن
- نوع شمارشی و کاربرد آن
- روشهای انتقال یک ساختار به تابع
- ارتباط بین ساختار و اشاره گر
- کلمه کلیدی typedef و کاربرد آن

مقدمه Introduction

- قبلا گفتیم که آرایه ها مجموعه داده های هم نوع را پشتیبانی می کنند.
- اگر با مجموعه‌ای از داده‌ها که ویژگی های یکسان ندارند ، مواجه باشیم، در این صورت از انواع داده کاربر استفاده خواهیم کرد.
- انواع داده کاربر اجازه می‌دهد که کاربر به چند روش نوع داده‌هایی با توجه به نیاز خود ایجاد کند.

انواع داده های کاربر در زبان C عبارتند از :



ساختار struct

- ساختار، داده جدیدی است که هر عنصر آن می تواند از نوع داده متفاوتی باشد.
- اگر بخواهیم داده های مختلف مربوط به یک پدیده را در یک مجموعه تعریف و ذخیره کنیم، از ساختار استفاده می کنیم.
- مثلا داده های مربوط به یک دانشجو، شامل: نام دانشجو (string)، شماره دانشجویی (long int) و نمره (float) می توانند براحتی در یک مجموعه داده از نوع ساختار تعریف شده و با یک نام مشخص در دسترس باشند.

فرم کلی ترکیب یک ساختار

```
struct   structure_name
{
  type  variable1_name ;
  type  variable2_name ;
   . . . .
   . . . .
  type  variablem_name ;
} ;
```

برای تعریف یک ساختار در زبان C از کلمه کلیدی struct استفاده می کنیم:

```
struct   student
{
   int  st_no ;
   char  name[15] ;
   float  grade ;
} ;
```

توجه : تعریف بالا یک تعریف نوع است! نه تعریف متغیر.

بعد از تعریف نوع یک ساختار، برای استفاده در برنامه باید متغیر هایی از آن نوع داده تعریف شوند:

```
struct  student  x , z ;
```

می‌توان x و z را به یکی از دو صورت زیر نیز تعریف کرد:

روش اول

```
struct  student
{
   int  st_no ;
   char  name[15] ;
   float  grade ;
} x , z ;
```

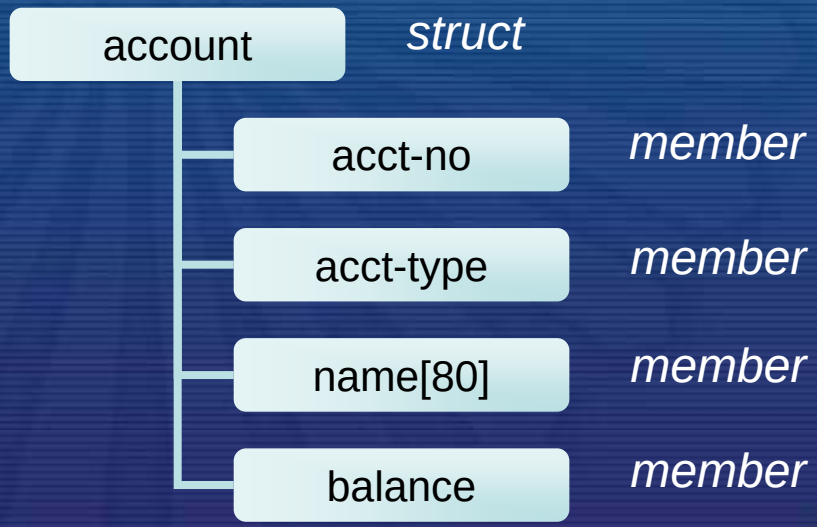
روش دوم

```
struct
{
   int  st_no ;
   char  name[15] ;
   float  grade ;
} x , z ;
```

در هر دو روش بالا x و z بلافاصله بعد از تعریف struct تعریف شده اند ، ولی در روش اول نامی برای نوع داده ساختار تعیین نشده و این باعث می شود نتوانیم جای دیگری در برنامه، متغیر دیگری از این نوع ساختار تعریف کنیم.

- هریک از اعضای ساختار می‌تواند متغیر معمولی، اشاره‌گر، آرایه، یا حتی ساختار دیگری باشد.
- به اعضای یک ساختار نمی‌توان کلاس حافظه اختصاص داد و همچنین نمی‌توان هنگام تعریف یک ساختار ، به اعضای آن مقدار اولیه نسبت داد.
- نمونه ای از تعریف ساختار ، همراه با نمای ظاهری آن:

```
struct  account
{
      int  acct_no ;
      char  acct_type ;
      char name[80] ;
      float  balance ;
};
```

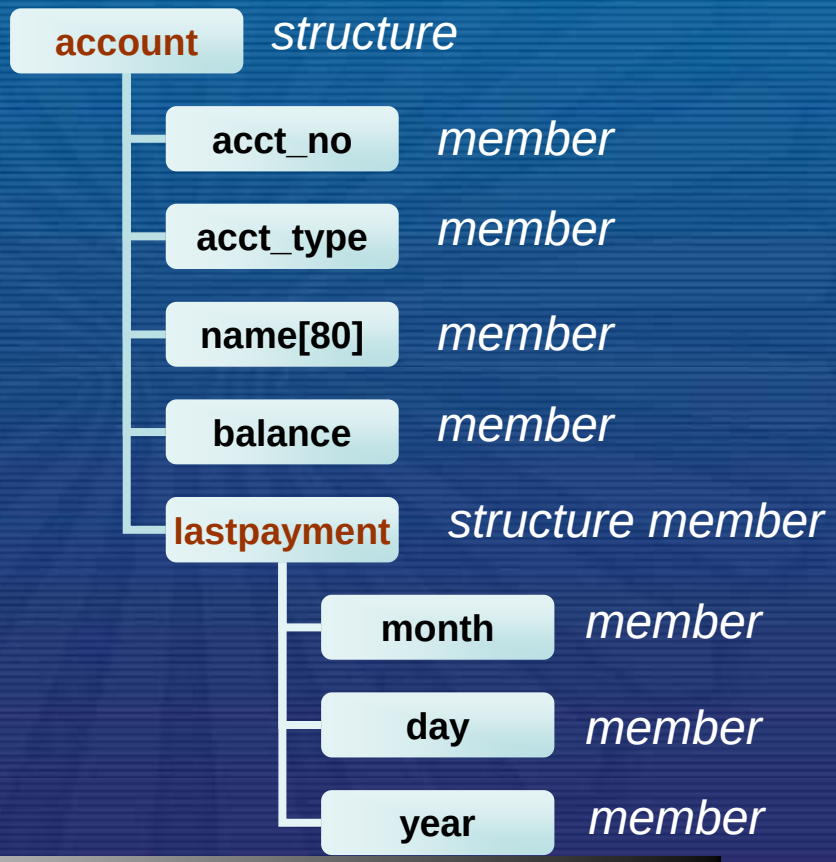


ساختارهای تو در تو

nested structs

تعریف یک ساختار تودرتو با نمای ظاهری آن:

```
struct date
{
    int month ;
    int day ;
    int year ;
} ;


struct account {
    int  acct_no ;
    char  acct_type ;
    char  name[80] ;
    float  balance ;
    struct  date  lastpayment ;
} oldcustomer , newcustomer;
```



در چنین موقعیتی باید تعریف date قبل از تعریف account بیاید.

اختصاص مقادیر اولیه initialization

- به اعضا یا فیلدهای متغیر در ساختار ، می‌توان مقادیر اولیه نسبت داد.
- مقادیر اولیهٔ مورد نظر، باید در داخل زوج آکولاد قرار گیرند و با کاما از یکدیگر مجزا گردند.
- فرم کلی آن به صورت زیر است:

storage_class struct tag variable = {value1 , value2 , … , value *m*} ;

- که در آن value1 , value2 , ...value *m* معرف مقادیری است که باید به ترتیب به عناصر اول، دوم ، ... و m اُم متغیر ساختار اختصاص یابد .

مثال: به متغیر oldcustomer که از نوع ساختار accont تعریف میشود مقدار اولیه می دهیم:

```
struct  date {
                int  month ;
                int  day ;
                int year ;
              } ;
struct  account {
                    int  acct-no ;
                    char  acct_type ;
                    char  name[80] ;
                    float  balance ;
                    struct  date  lastpayment ;
                  } ;
static  struct  account  oldcustomer = {12746 , `R´ , “Mahmood Hesabi" , 12986.50 , 5 , 24 , 70} ;
```

آرایه‌ای از ساختارها an array of structs

- می توان آرایه ای تعریف کرد که هر عنصر آن یک ساختار باشد.
- این کاربرد متداولترین کاربرد ساختار است و اساس پایگاههای داده را تشکیل می دهد.
- برای ایجاد این ساختمان داده ، ابتدا باید ساختار را تعریف کنیم ، سپس آرایه ای از نوع ساختار تعریف شده ، تعریف می کنیم:

```
struct account
{
  …
};
struct account customer[50];
```

به هر عضو آرایه ای از ساختارها ، نیز می توان مقدار اولیه نسبت داد؛ این کار بصورت ترکیبی از مقداردهی آرایه ها و ساختارها انجام می شود:

```
struct  date
{
                    char  name[80] ;
                    int  month ;
                    int  day ;
                    int  year ;
} ;
static  struct date birthday[ 7] =  { "Sara" , 12 , 30 , 73 ,
                                      "Hassan" , 5 , 13 , 66 ,
                                      "Dara" , 7 , 15 , 72 ,
                                      "Iraj" , 11 , 29 , 70 ,
                                      "Arash" , 2 , 4 , 77 ,
                                      "Susan" , 12 , 29 , 63 ,
                                      "Ahmad" , 4 , 12 , 69 } ;
```

پردازش یک ساختار processing a struct

- اعضای ساختار معمولا به صورت مستقل و جدا از هم پردازش می شوند.
- به هر عضو یا عنصر ساختار با استفاده از عملگر ´.` یا عملگر عضویت (که گاهی آن را عملگر period یا"dot" نیز نامند) دستیابی یا رجوع می‌شود.
- فرم کلی رجوع به عنصری از یک ساختار :

variable . member ویا structure_name . element_name

- اپراتور نقطه نسبت به سایر اپراتورها ، حتی اپراتورهای یکانی ، تقدم بالاتری دارد.

برای مثال عبارات زیر معادل هم هستند:

++ variable . member و ++ (variable . member)

&(variable . member) و &variable . member

انتقال ساختار به تابع

- می‌توان اعضای ساختار و یا تمامی ساختار را به یک تابع گذر داد.
- برای فرستادن یک عنصر از یک ساختار، به تابع یا برگشت دادن عنصری از یک ساختار ازتابع به تابع فراخواننده با هریک از اعضای ساختار مشابه یک متغیر معمولی تک مقدار ، برخورد می‌شود .

مثال: برنامه مقابل نحوه انتقال عنصری از ساختاررا به تابع و برگشت عنصری از ساختار را نمایش می دهد:

```
main ( )
{
typedef  struct {                /* structure declaration */
              int  month ;
              int  day ;
              int  year ;
             }date ;
```

بقیه در اسلاید بعد...

```
suruct   {                                   /* structure declaration */
      int  acct_no ;
      char  acct_type ;
      char  name[80] ;
      float  balance ;
      date  lastpayment ;
   }customer ;

float  adjust (char  name[ ] , int  acct_no , float  balance) ;  // function declaration
   . . .
   customer.balance = adjust (customer.name , customer.acct_no , customer.balance) ;
   . . .
}
float  adjust (char name[ ] , int acct_no , float balance)  // function defination
{
   float  newbalance ;  /* local variable declaration */
   . . .
   newbalance = ... ;   /* adjust value of balance */
   . . .
   return (newbalance) ;
}
```

فصل نهم : نوع داده کاربر – انتقال ساختار به تابع

- به کمک یک اشاره‌گر به نوع ساختار ، می‌توان تمامی یک ساختار را به عنوان آرگومان به یک تابع انتقال داد و یا از تابع به عنوان خروجی برگشت داد.

- چون انتقال ساختار به تابع با اشاره گرصورت می گیرد، نتیجهٔ هر عضو ساختار که در درون تابع فراخوانده شده تغییر یابد ، در تابع فراخواننده نیز منعکس خواهد شد .

مثال: برنامه ساده زیر را در نظر بگیرید:

```
typedef  struct
{
  char  *name ;
  int  acct_no ;
  char  acct_type ;
  float  balance ;
} record ;
void  adjust (record  *pt) ;
```

بقیه در اسلاید بعد...

فصل نهم : نوع داده کاربر – انتقال ساختار به تابع

```
main ( )   / * transfer a structure-type pointer to a function */
{
      static  record customer = {"Nader , 3333 , `c´ , 33.33} ;
      printf (" %s %d %c %.2f \n" , customer.name , customer.acct_no ,
               customer.acct_type , customer.balance) ;
      adjust (&customer) ;
      printf (" %s %d %c %.2f \n" , customer.name , customer.acct_no ,
               customer.acct_type , customer.balance) ;
}

void  adjust (record  *pt)
{
               pt-> name = "Payam" ;
               pt-> acct-no = 9999 ;
               pt-> acct-type = `r´ ;
               pt-> balance = 99.99
               return ;
}
```

این برنامه، نحوهٔ انتقال یک ساختار به یک تابع را با گذر دادن آدرس آن به تابع ، نمایش می‌دهد.

- در برنامه قبل از عملگر <- (پیکان راست) استفاده شده است.

- این عملگر در فصل اول به عنوان یک عملگر حافظه معرفی شد؛ این عملگر فقط با اشاره گر های ساختار و به صورت زیربه کار می رود:

با فرض تعاریف مقابل :

```
struct rec
{
   int data;
   char type;
}
struct rec *rec_pointer ;
```

عبارتهای زیر معادل هم اند :

(* rec_pointer) . data و rec_pointer -> data

(* rec_pointer). type و rec_pointer -> type

داده تعریف شده توسط کاربر

- برنامه نویس C می تواند بکمک دستور typedef نام جدیدی برای نوع داده تعریف کند.
- فرم کلی دستور typedef به صورت زیر است :

```
typedef   type  name ;
```

و یا به این صورت خواهد بود :

```
typedef   type   new_type ;
```

که در آن ، type می‌تواند هریک از نوع داده‌های مجاز باشد وname یاtype- new نیز نام جدید برای این نوع است .

دو تعریف زیر معادلند :

```
typedef  float  height ;
height  a[100] , b[100] ;
```

و

```
typedef  float  height[100] ;
height  a , b ;
```

بیشتر کاربرد typedef در تعریف ساختارهاست:

```
typedef  struct
{
  member 1 ;
  member 2 ;
  . . .
  member m ;
} new_type ;

new_type   x,y;
```

ساختار داده‌ها و اشاره‌گرها

- عملگر<- نیز مانند عملگر `.` دارای بالاترین تقدم است .
- عملگر<- می‌تواند با عملگر نقطه ترکیب شود و بدین طریق عمل دستیابی به یک زیرعضو یا submember را در درون یک ساختار فراهم سازد:

```
struct_ptvar ->member . submember
```

عضو ساختار

- اعضای یک ساختار می توانند اشاره گر به انواع داده ها باشند حتی اشاره گر به ساختار.
- عضوی از یک ساختار حتی می تواند اشاره گری از جنس خود ساختاری که در آن تعریف شده، باشد.

```
struct  tag
{
        member 1 ;
        member 2 ;
        ....
        ....
        ....
        struct  tag *name ;
};
```

- یکی از مهمترین کاربردهای روش بالا، در ایجاد ساختارهایی به نام لیست پیوندی و انجام عملیات یا پردازش روی آن می‌باشد.

اجتماع

union

- union ،ساختار داده هایی هستند که از چند عضو تشکیل یافته و هر عضو آنها می تواند دارای نوع داده ای منحصر بخود باشد .
- union ، متغیری است که امکان ذخیره کردن انواع مختلف داده ، در مکان مشترکی از حافظه را فراهم می‌کند.
- این گونه ساختمان داده‌ها در کاربردهایی مفید می‌باشند که دارای اعضای چندگانه از نوع مختلف هستند ، ولی در هر زمان باید فقط به یکی از این اعضا مقداری نسبت داده شود.

ترکیب یک union می‌تواند به صورت زیر تعریف گردد:

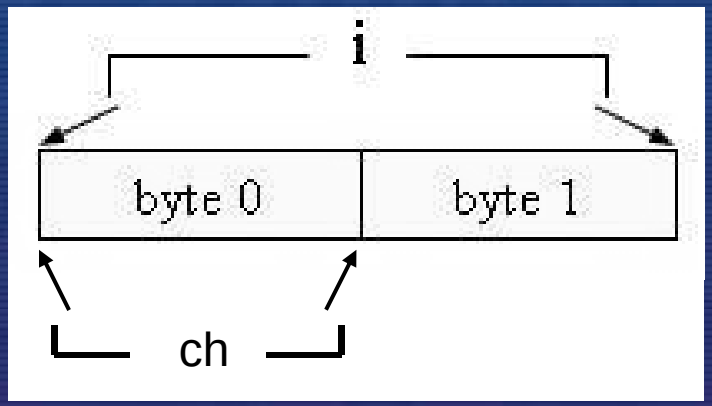
```
union  tag {
    member 1 ;
    member 2 ;
    ...
    ...
    ...
    member m ;
} ;
```

این تعریف نیز مشابه تعریف یک ساختار ، موجب توصیف یا اعلان متغیر نمی‌گردد ، بلکه فقط تعریف یک نوع داده است .

تعریف مقابل را در نظر بگیرید:

```
union  tag
{
    int  i ;
    char ch ;
} ;
union tag x;
```

متغیرهای i وch که به ترتیب از نوع int وchar می‌باشند ، دارای حافظه مشترک هستند که البته متغیر i ، دو بایت و متغیر ch ، یک بایت حافظه را اشغال می‌کند ، ولی آدرس شروع آنها ، یکسان می‌باشد:



- توجه داشته باشید که کامپایلر برای یک union به اندازه ی حافظه در نظر می گیرد که بزرگترین عنصر union اشغال می کند.

- یک union می‌تواند عضو یک ساختار باشد. همچنین یک ساختار می‌تواند عضو یک unionباشد .

نوع شمارشی

- یکی از انواع داده‌های اسکالر نوع شمارشی است.
- در حالت کلی یک نوع شمارشی به صورت زیر تعریف می‌شود:

enum tag {member 1 , member 2 , ... , member *m*} ;

- اسامی اعضا باید متفاوت و متمایز از یکدیگر باشند.

مثال: قطعه برنامه زیر ، یک نوع شمارشی به نام coin تعریف می‌کند و نوع متغیر moneyرا از این نوع اعلان می‌کند.

```
enum coin {penny , nickel , dime , quarter , half_dollar , dollar} ;
enum coin money ;
```

با داشتن این تعریف و اعلان ، دستورهای زیر کاملاً درست و معتبر است:

```
money = dime ;
if (money = = quarter)  prinf ("is a quarter") ;
```

- هر سمبول معرف یک مقدار صحیح است و می‌تواند در هر عبارت از نوع مقادیر صحیح بکار برده شود:

```
printf ("the number of nickels in a quarter is %d" , quarter + 2) ;
```

- مقدار اولین سمبول شمارشی برابر صفر ، مقدار دومین سمبول شمارشی برابر ۱ ، و بالاخره مقدار nاُمین سمبول شمارشی برابر n – 1 می‌باشد.

- می‌توان به هریک از سمبولها ، مقدار اولیه نسبت داد .

- وقتی که به یکی از سمبولها به طریق مزبور مقدار اولیه نسبت داده شد ، سمبول بعدی ، مقدار بعدی را خواهد داشت

```
enum coin {penny , nickel , dime , quarter = 100 , half_dollar , dollar} ;
```

- پس از تعریف بالا مقادیر سمبولها به صورت زیر خواهند بود:

penny	0
nickel	1
dime	2
quarter	100
half_dollar	101
dollar	102

توجه : برخلاف تصور سمبولهای یک نوع داده شمارشی نمی توانند بطور متعارف از طریق دستورات ورودی و خروجی استاندارد خوانده یا نوشته شوند؛ در ادامه روشهایی برای چاپ سمبلها در خروجی بحث خواهد شد.

۱– می توان برای نمایش سمبولها از دستور switch استفاده کرد؛ مثلا بصورت زیر :

```
switch  money {
   case penny : printf ("penny") ;
     break ;
   case nickel : printf ("nickel") ;
     break ;
   case dime : printf ("dime") ;
     break ;
   case quarter : printf ("quarter") ;
     break ;
   case half_dollar : printf ("half_dollar") ;
     break ;
   case dollar : printf ("dollar") ;
   }
```

۲- می توان آرایه‌ای از رشته‌ها تعریف کرد و مقدار متغیر شمارشی را به عنوان شاخص یا index آن آرایه بکار برد تا یک مقدار شمارشی را به رشته متناظر آن ترجمه کند .

```
char name[ ] = {
            "penny" ,
            "nickel" ,
            "dime" ,
            "quarter" ,
            "half_dollar" ,
            "dollar"
                  } ;
printf ("%c" , name [(int) money] ) ;
```

توجه: درصورتی که به سمبولها مقدار اولیه مغایر با ترتیب معمول داده شود مسلما این روش کارایی خود را از دست خواهد داد.

اصول کامپیوتر ۲

رشته علوم کامپیوتر
۴ واحد درسی
فصل دهم

نام منبع و مؤلف :

- ✓ اصول کامپیوتر ۲
- ✓ دکتر داود کریم زادگان مقدم
- ✓ انتشارات دانشگاه پیام نور ۱۳۸۳

فصل دهم : پشته و صف

اهداف کلی و رفتاری

مقدمه

نوع داده مجرد پشته

پشته چندگانه

ارزشیابی عبارات infix، postfix، prefix

نوع داده مجرد صف

صف حلقوی(دایره ای)

هدف کلی

آشنایی با ساختار پشته و صف

هدف های رفتاری

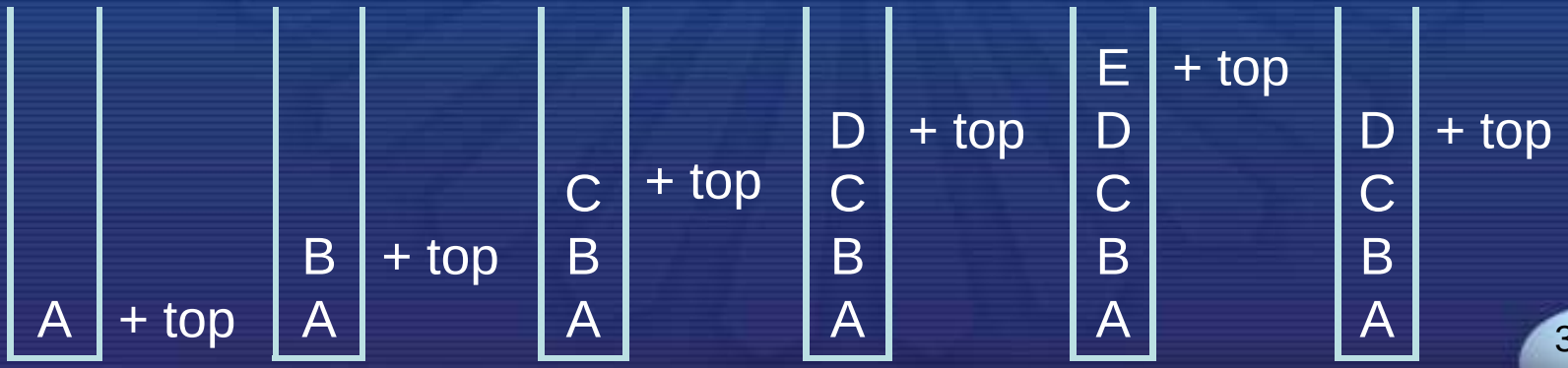
- آشنایی کلی با انواع مجرد پشته
- نحوه درج و حذف عناصر در پشته
- پشته ها با توابع پیاده سازی صف و پشته در زبان C
- آشنایی با نوع دادهٔ مجرد صف
- نحوه درج و حذف عناصر در صف

مقدمه Introduction

- پشته و صف، حالت های خاصی از نوع داده عمومی یعنی لیست های مرتب شده، هستند.
- مفاهیم صف و پشته، منجر به تشکیل ساختمان داده جدیدی به نام لیست های پیوندی می شوند.
- این دو نوع داده، در دانش کامپیوتر کاربرد وسیع و متنوعی دارند.

نوع دادهٔ مجرد پشته

- پشته لیستی است که هم جایگذاری و هم حذف از یک سمت آن که top نامیده می‌شود، صورت می‌گیرد.
- محدودیت کار با پشته ما را ملزم می‌سازد که اگر عناصر A, B, C, D, E را به ترتیب به پشته اضافه کنیم، E اولین عنصری خواهد بود که از پشته حذف می‌گردد.



- پشته یک لیست LIFO است، به این معنی که آخرین ورودی به پشته، اولین خروجی از پشته است.
- اضافه کردن یک عنصر به هر پشته ای امکان پذیر است ولی حذف عنصر از پشته خالی ممکن نیست.
- قبل ازحذف عنصری از پشته باید از خالی نبودن پشته اطمینان حاصل کرد.

نوع داده مجرد پشته:

Structure Stack is
objects: a finite ordered list with zero or more elements.
functions:
for all stack ∈ Stack, item ∈ element, MaxStackSize ∈ positive integer
Stack CreateS (MaxStackSize) ::= create an empty stack whose maximum size is MaxStackSize
Boolean Is Full (stack, MaxStackSize) ::= if (number of elements in stack = = MaxStackSize)
return TRUE
else return FALSE
Stack Add (stack , item) ::= if (Is Full (stack)) stack_full
else insert item into top of stack and return
Boolean IsEmpty (stack) ::= if (stack = = CreateS (MaxStackSize))
return TRUE
else return FALSE
Element Delete (stack) ::= if (IsEmpty (stack)) return
else remove and return the item on the top of the stack.

- راحت‌ترین روش پیاده‌سازی Stack ADT، استفاده از یک آرایهٔ یک‌بعدی به نام stack[MaxStackSize] است که MaxStackSize ، حداکثر تعداد عناصر آرایه می‌باشد.

- همراه با آرایه، یک متغیر به نام top وجود دارد که همیشه به عنصر بالایی پشته اشاره می‌کند.

- مقدار اولیه top ، 1- است که نشان‌دهندهٔ یک پشتهٔ خالی است.

تعاریف مربوط به پیاده سازی ADT پشته :

```
Stack CreateS(MaxStackSize)::=
# define MaxStackSize 100     /* maximum stack size*/
typedef struct  {
                        int key;
                                    /* other fields */
                    } element ;
element stack [MaxStackSize];
int top = -1;
Boolean Is Empty (Stack) ::= top < 0;
Boolean Is Full (Stack) ::= top > = MaxStackSize-1;
```

- عملکرد Is Empty بسیار ساده بوده و بطور مستقیم در تابع add پیاده‌سازی شده است:

```
void  add(int *top , element item)
{
  if (*top >= MaxStackSize-1)
    {
      stack_full( ) ;
      return ;
    }
  stack[++top]=item ;
}
```

- عملکرد Is Full نیز ساده بوده و در تابع delete به این صورت پیاده‌سازی شده است:

```
element delete(int *top)
{
  if (*top= = -1)
      return stack_empty( );
  return  stack[(*top)--] ;
}
```

- فراخوانی‌های این دو تابع به صورت زیر خواهد بود :

```
add(&top, item);

item=delete(&top);
```

- توجه داشته باشید که در هر دو فراخوانی، آدرس top را به تابع می فرستیم تا تغییرات ایجاد شده در top بوسیلهٔ add یا delete به برنامهٔ اصلی برگشت داده شود.

- پشته در زبان C می تواند به صورت ساختمانی با دو عضو تعریف شود :

۱- آرایه ای که عناصر پشته را نگه می دارد؛

۲- متغیر صحیحی که top را نگه می دارد.

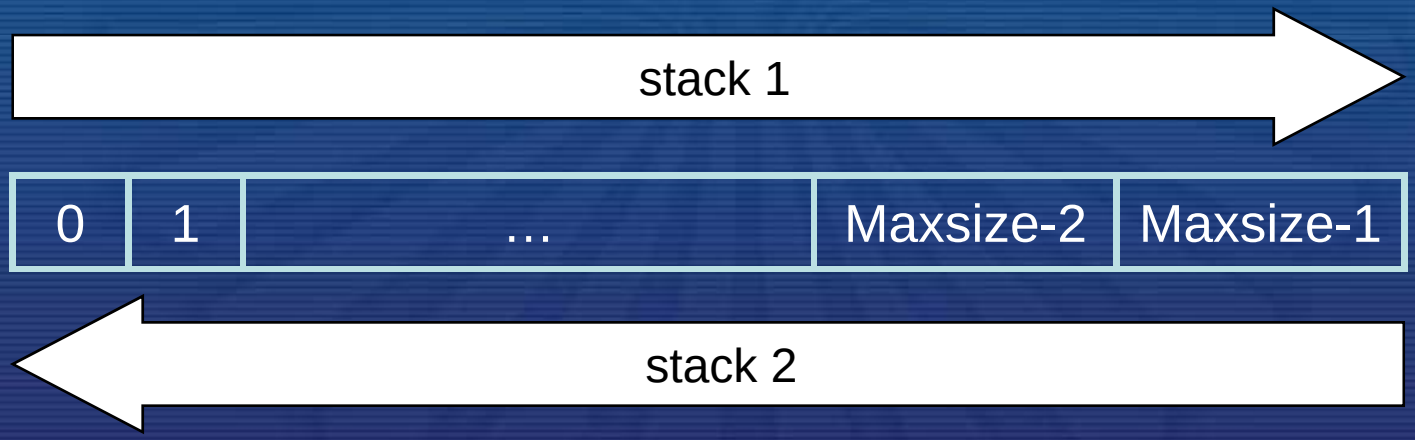
تعریف پشته ای با عناصر صحیح :

```
#define  size  50
struct  stack {
        int  top ;
        int  item[size] ;
};
struct  stack  s ;
```

پشته چندگانه

پشته های چندگانه پشته هایی هستند که برای ذخیره اطلاعات خود از یک آرایه استفاده می کنند و هر یک از پشته ها سهمی از آرایه داشته و در محل خود رشد می کنند.

برای مثال دو پشته می توانند از آرایه زیر بطور مشترک استفاده کنند؛ پشته ۱ از طرف چپ به راست و پشته ۲ از طرف راست به چپ گسترش می یابد.



هر دو پشته تا جایی که جای خالی داشته باشند می توانند عنصر جدید قبول کنند.

- راه حل برای بیشتر از دو پشته به این سادگی نیست؛ برای استفاده ی n پشته از یک آرایه ، اگر طول مورد نیاز هر یک از پشته ها را ندانیم، مقدار کل حافظه موجود (آرایه) را به n قسمت مساوی تقسیم می کنیم؛
- در این صورت stackno به یکی از n پشته اشاره می‌کند؛
- برای ایجاد این پشته باید برای هر دو موقعیت انتهایی و ابتدایی این پشته ها اندیس ایجاد کنیم.

boundary[stackno] (0 ≤ stackno < MaxStackS)

به عنصر تحتانی stackS اشاره می کند ؛

top[stackno] (0 ≤ stackno < MaxStackS)

به عنصر بالایی stackS اشاره می کند ؛

در یک پشته اگر boundary[stackno]=top[stackno] باشد، آنگاه پشته خالی خواهد بود

- پشتهٔ stackno می‌تواند از boundary[stackno]+1 تا boundary[stackno+1] قبل از اینکه پر شود، گسترش یابد .

- ساختمان این پشته های چندگانه بطوری است که اگر یکی از پشته ها پر شد، و پشته های دیگر محل خالی داشتند می توانیم ترتیبی دهیم که پشته پر شده، از فضای خالی دیگر پشته ها استفاده کند.

- این کار میتواند با حرکت دادن پشته ها بطوری که فضای خالی به پشته ی پر شده برسد، انجام شود.

ارزشیابی عبارات

infix ، postfix، prefix

- روش استاندارد نوشتن عبارات،به‌عنوانinfix معرفی و شناخته می‌شود. در این روش عملگرهای دودویی را در بین دو عملوند قرار می‌دهیم.
- کامپایلرها عموماً از نشانه‌گذاری بدون پرانتز به نامpostfix استفاده می‌کنند، در این روش، هر عملگر بعد از عملوندهای مربوطه ظاهر می‌شود.

چندین عبارت infix و معادل postfix آنها:

infix	Postfix
2+3*4	2 3 4*+
a*b/c	ab*c/
((a/(b-c+d))*(e-a)*c	abc-d+/ea-*c*
a/b-c+d*e-a*c	ab/c-de*+ac*-

ارزیابی یک عبارت postfix با استفاده از پشته :

عبارت: 62/3-42*+

on the process

6	2	/	3	-	4	2	*	+

expression string

is operator

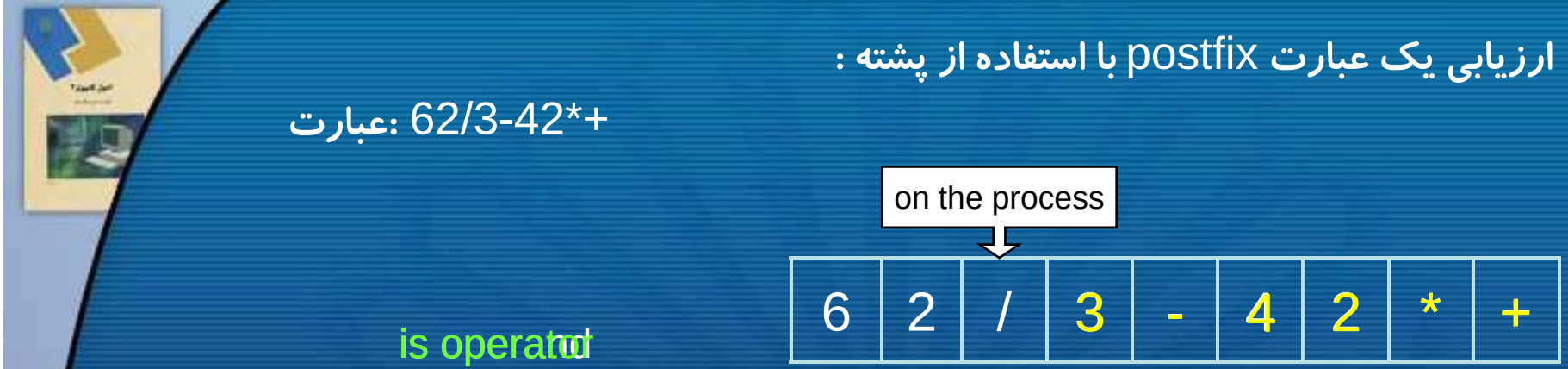
3

top

stack

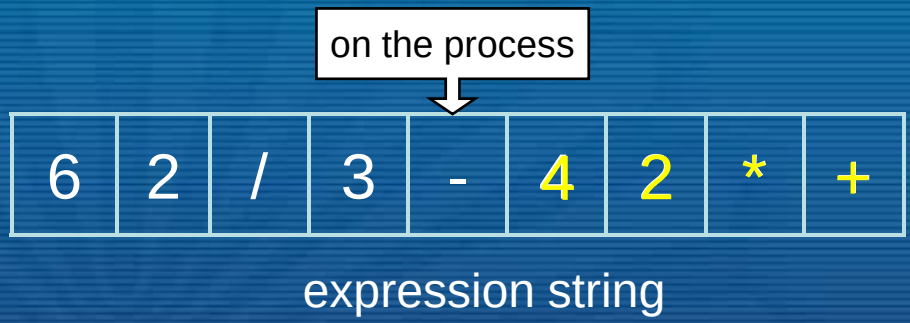
ارزیابی یک عبارت postfix با استفاده از پشته :

عبارت: 62/3-42*+

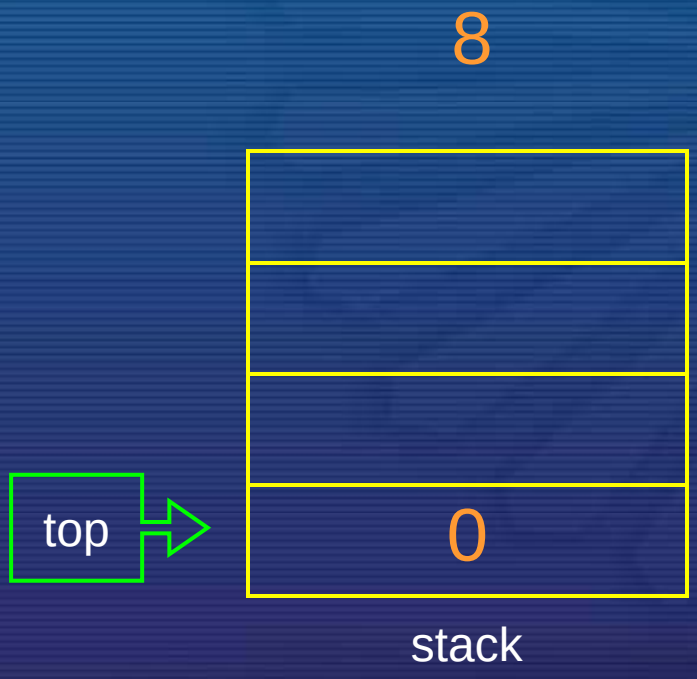


ارزیابی یک عبارت postfix با استفاده از پشته :

عبارت: 62/3-42*+

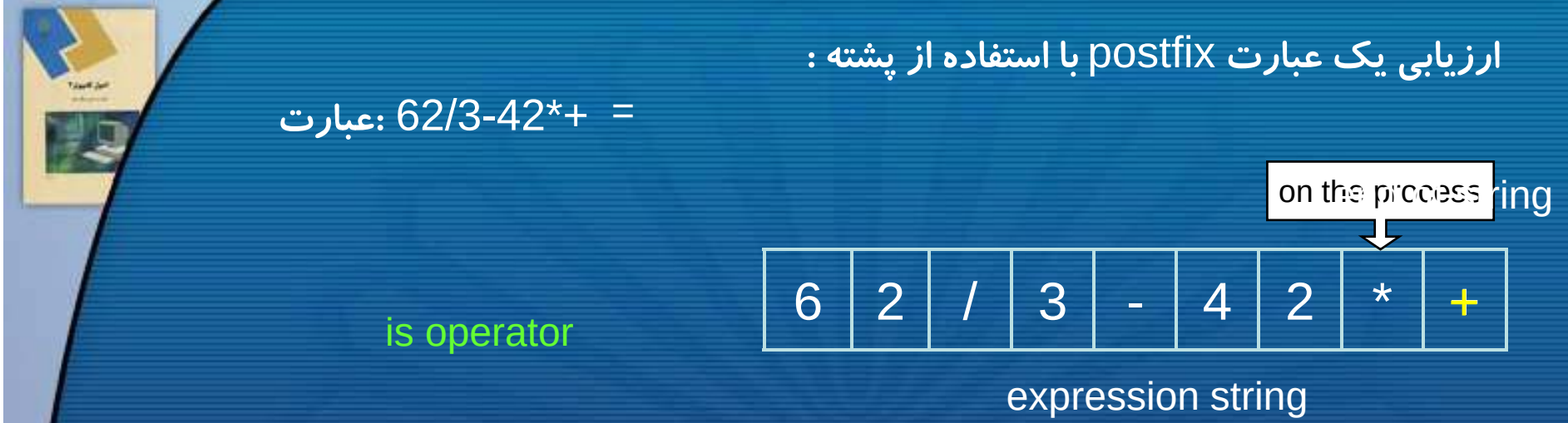


is operator



ارزیابی یک عبارت postfix با استفاده از پشته :

عبارت: 62/3-42*+ =



- برای تبدیل یک عبارتinfix به postfix می‌توان الگوریتمی را به صورت زیر بیان نمود:

۱ـ پرانتزگذاری کامل عبارت،

۲ـ انتقال همه عملگرهای دودوئی به نحوی که با پرانتز بستهٔ مربوطه ی سمت راست آن تعویض شوند؛

۳ـ حذف تمام پرانتزها.

- برای مثال عبارت a/b-c+d*e-a*c زمانی که کاملاً پرانتزگذاری شود، به صورت زیر درمی‌آید:

((((a/b)-c) + (d*e))-a*c))

و بعد از انجام مراحل ۲و۳ داریم :

ab/c-de*+ac*-

- شکل دیگری برای نمایش یک عبارت که ارزیابی آن ساده بوده و نیازی به پرانتزگذاری ندارد، نشانه‌گذاری prefix نامیده می‌شود .

- در این روش نمایش، عملگرها بر عملوندهای مربوطه مقدم است.

چند عبارت infix و معادل prefix آنها:

infix	Prefix
a*b/c	/*abc
a/b-c+d*e-a*c	-+-/abc*de*ac
a*(b+c)/d-g	-/*a+bcdg

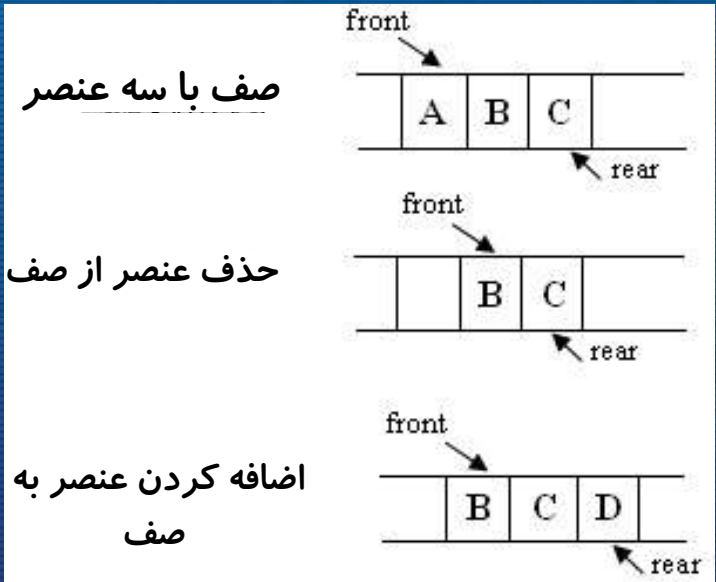
نوع داده مجرد صف

- صف یک لیست دارای ترتیب است که تمام جایگذاری ها درآن از یک سمت و تمام حذف های آن از سمت دیگر انجام می‌گیرد .

- در یک صف $Q=(a_0, a_1, \ldots, a_{n-1})$; a_0عنصر ابتدا (front) و a_{n-1} عنصر انتها (rear) می‌باشد و a_{i+1} در کنار a_i قرار دارد $(0 \le i < n-1)$.

- از آنجا که اولین عنصر وارد شده به یک صف اولین عنصری است که خارج می‌شود، صفها به‌عنوان لیست های FIFO (First In First Out) شناخته می شوند.

یک صف با سه عنصر :



تعریف یک صف:

```
Queue Create Q(MaxQueueSize)::=
# define MaxQueueSize 100      /* Maximum queue size */
typedef struct  {
                        int key;
                        /* other fields */
                    }  element;
element queue [MaxQueueSize];
int rear = -1;
int front = -1;
Boolean  IsEmptyQ(queue)::=front = = rear
Boolean  IsFullQ (queue) ::= rear = = MaxQueueSize-1
```

تابع جایگذاری یا درج در صف:

```
void  addq (int *rear, element item)
{
  if (*rear = = MaxQueueSize-1)
  {
     queue_full ( );
     return;
  }
   queue[++ * rear] = item;
}
```

این تابع عنصری را به انتهای صف اضافه می کند .

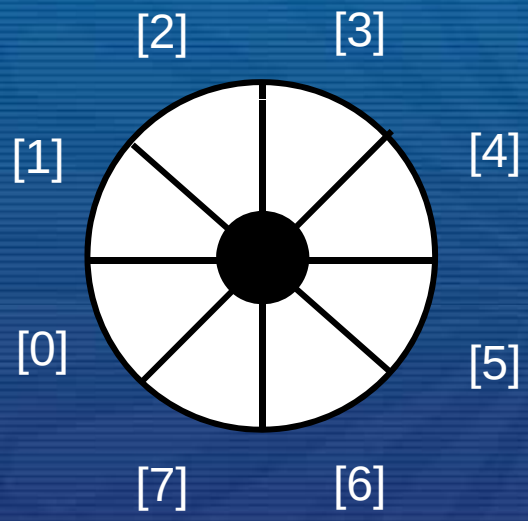
تابع حذف عنصری از یک صف:

```
element  deleteq (int *front , int  rear)
{
   if (* front = = rear)
        return queue_empty ( );     /* return an error key */
   return queue [++ *front];
}
```

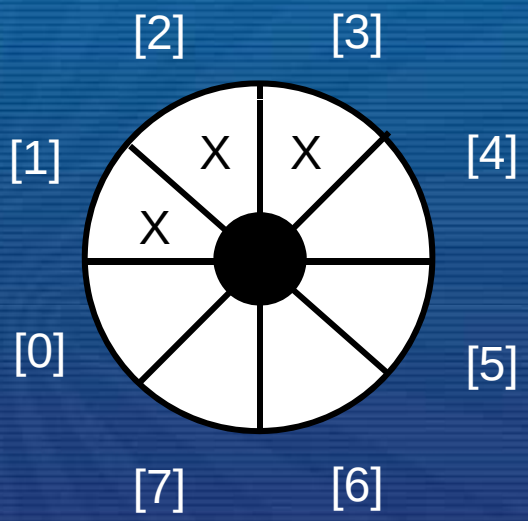
این تابع عنصری را از ابتدای صف حذف می کند .

صف حلقوی(دایره ای)

می‌توان نمایش مؤثرتری برای صف ارائه داد ؛ صف حلقوی :



front = 0
rear = 0

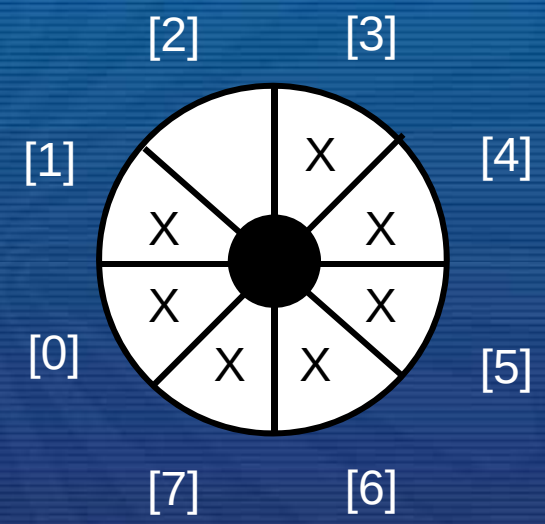


front = 0
rear = 3

در این نمایش هرگاه front=rear باشد، صف خالی خواهد بود.

در صفهای حلقوی با وارد شدن داده به صف rear افزایش پیدا می کند و با حذف داده front افزایش پیدا می کند.

نمونه ای از یک صف حلقوی پر :



front = 2
rear = 1

خانه 2 به این خاطر خالی نگه داشته شده که بتوانیم یک صف خالی را از یک صف پر تشخیص دهیم. چون با ورود داده به این خانه rear مساوی front می شود و این همان شرط خالی بودن صف است.

- گفتیم با ورود داده و با خروج داده rear و front هر دو همواره افزایش پیدا می کنند.
- برای افزایش rear و front در صف های حلقوی از فرمول زیر استفاده می کنیم :

```
*rear = (*rear + 1) % MaxQueueSize;

*front = (*front + 1) % MaxQueueSize;
```

- تابع جایگذاری به داخل یک صف حلقوی :

```
void  addq (int front , int *rear, element item)
{
   *rear = (*real+1) % MaxQueueSize;
   if (front = = *rear)
   {
      queue-_full (rear);    /* reset rear and print error */
      return;
    }
   queue[*rear] = item;
}
```

تابع حذف از یک صف حلقوی :

```
element  deleteq (int *front , int rear)
{
    element  item;
    if (*front = = rear)
         return queue_empty ( );     /* queue_empty returns an error key */
   *front = (*front+1) % MaxQueueSize;
    return queue[*front];
}
```

بعضی از موارد کاربرد صف های دایره ای عبارتند از :

- در سیستم عامل جهت نگهداری اطلاعات خوانده شده از روی دیسک؛
- در سیستم های بلادرنگ (real time system) جهت پردازش اطلاعات بافر؛
- در پردازشگرهای کلمه جهت پاراگراف بندی یا تنظیم خطوط.

اصول کامپیوتر ۲

رشته علوم کامپیوتر

۴ واحد درسی

فصل یازدهم

نام منبع و مؤلف :

- ✓ اصول کامپیوتر ۲
- ✓ دکتر داود کریم زادگان مقدم
- ✓ انتشارات دانشگاه پیام نور ۱۳۸۳

فصل یازدهم : ساختار درختی

پیمایش درخت

عمق درخت جستجوی دودویی

بررسی عنصر در درخت جستجوی دودویی

حذف عنصر از درخت جستجوی دودویی

عمق یک درخت جستجوی دودویی

درخت آوایلی متعادل دودویی

هدف کلی

آشنایی با انواع درخت، ویژگیهای ساختاری درخت و کاربردهای آن در برنامه سازی

هدف های رفتاری

- آشنایی با درخت جستجوی دودویی
- روشهای مختلف نمایش در ریسم درخت دودویی
- آشنایی با ساختمان داده درخت و ویژگیهای آن در زبان برنامه سازی C
- روشهای نمایش درخت دودویی
- روشهای پیمایش درخت دودویی

مقدمه Introduction

- یکی دیگر از ساختمان داده های مهم و پر کاربرد در C ، درخت یا Tree است.
- ساختار درختی یعنی مجموعه داده‌های سازماندهی شده‌ای که عناصر اطلاعاتی‌شان به‌وسیلهٔ انشعاباتی با هم ارتباط داشته باشند.
- درخت، یک نوع داده نیست بلکه یک ساختمان داده است.

درخت Tree

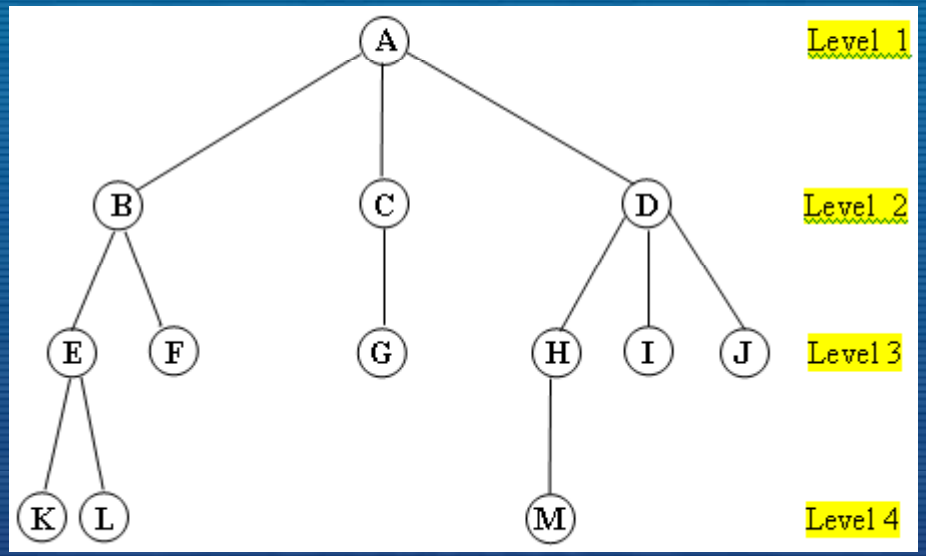
- درخت مجموعهٔ محدودی از یک یا چند گره با شرایط زیر است:

۱- دارای گره خاصی به نام ریشه است.

۲- بقیه گره‌ها به $n \geq 0$ مجموعهٔ مجزا $T_1, \ldots, T_n$ تقسیم شده که هریک از این مجموعه‌ها خود یک درخت هستند و زیردرخت ریشه نامیده می‌شوند.

- اولین گره درخت، ریشه درخت (root) نامیده می شود و در بالاترین نقطه درخت قرار می گیرد .

نمونه ای از یک درخت:



- این درخت سیزده گره دارد؛
- هر عنصر در یک درخت، ریشه زیردرختهای پایین تر از خود است؛
- تعداد زیردرختهای یک گره، درجه آن گره نامیده می‌شود؛

- گره‌هایی که درجه صفر دارند، برگ یا گره‌های پایانی نامیده می‌شوند؛
- درجه یک درخت حداکثر درجه گره‌های آن درخت می‌باشد؛
- گره‌ای که دارای زیردرختانی است، والد (parent) ریشه‌های زیردرختان است و ریشه‌های زیردرختان فرزندان آن گره می‌باشند؛
- فرزندان یک گره، گره‌های همزاد یا هم‌نیا نامیده می‌شوند؛

- اجداد یک گره، گره‌هایی هستند که در مسیر طی شده از ریشه تا آن گره وجود دارند؛
- نسلهای یک گره شامل تمام گره‌هایی است که در زیردرخت آن گره قرار دارند .
- سطح(level) یک گره بدین‌صورت تعریف می‌گردد که ریشه در سطح یک قرار می‌گیرد؛ و برای تمامی گره‌های بعدی، سطح گره برابر است با : سطح والد گره به اضافه یک.
- ارتفاع یا عمق یک درخت به بیشترین سطح گره‌های آن درخت گفته می‌شود ؛

نمایش درخت

نمایش لیست

- درخت را می‌توان به‌صورت زیر نمایش داد به‌نحوی که هر زیردرخت خود یک لیست می‌باشد:

(A (B (E (K, L), F), C(G), D(H (M), I, J)))

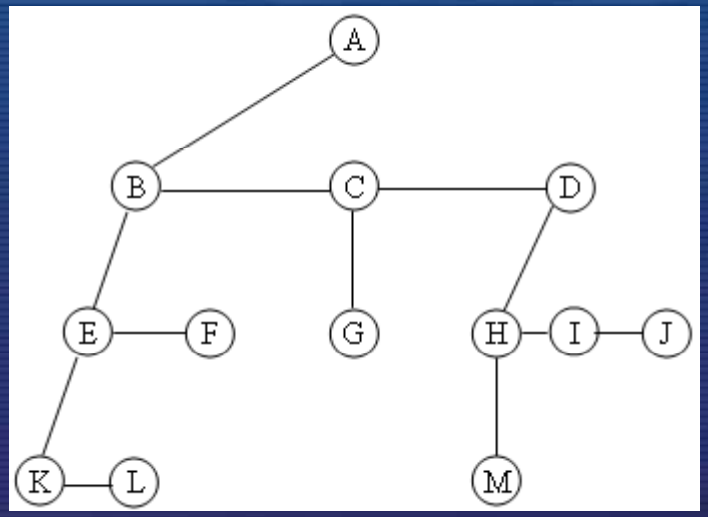
- توجه داشته باشید که ابتدا اطلاعات در گره ریشه ذکر می‌شود و بعد لیست زیردرختهای آن گره می‌آید.

نمایش فرزند چپ ــ همزاد راست

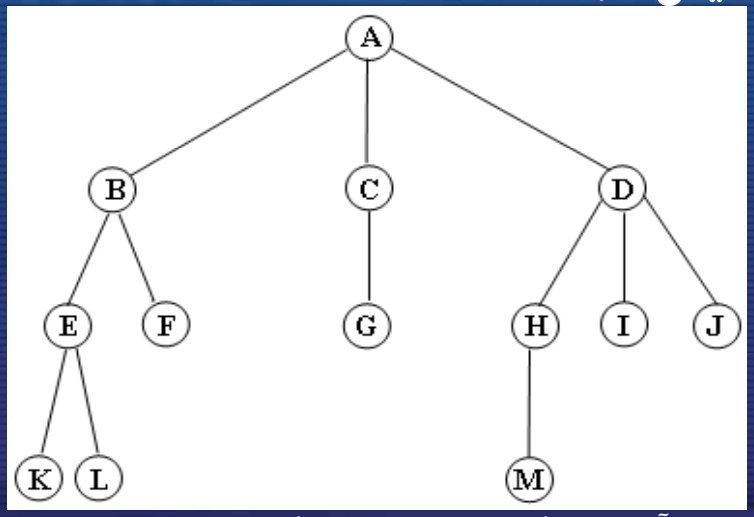
شکل زیر ساختار گره مورد استفاده برای نمایش فرزند چپ ـ همزاد راست را نشان می‌دهد :

data	
Left child	right sibling

نمایش فرزند چپ ـ همزاد راست



نمایش درخت



نمایش دودویی یک درخت

- درخت فرزند چپ ـ همزاد راست را به میزان ۴۵ درجه در جهت عقربه‌های ساعت چرخش می‌دهیم تا در خت دودویی نمایان شود.
- در این حالت ما به دو فرزند یک گره به‌عنوان فرزند سمت چپ و راست اشاره خواهیم نمود
- توجه داشته باشید که فرزند سمت راست گره ریشه تهی می‌باشد زیرا ریشه هر درخت فاقد گره همزاد می‌باشد

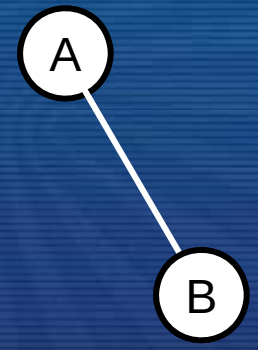
درخت دودویی

- اگر هر گره در یک درخت دارای دو انشعاب باشد به آن درخت، درخت دودویی گفته می شود .
- درخت دودویی یکی از انواع مهم ساختارهای یک درخت بوده که کاربردهای زیادی دارد .
- می‌توان هر درختی را به‌صورت درخت دودویی نمایش داد.
- برای درختهای دودویی، زیردرخت سمت چپ و زیردرخت سمت راست با یکدیگر متمایز خواهد بود .
- یک درخت دودویی ممکن است شامل صفر گره باشد.

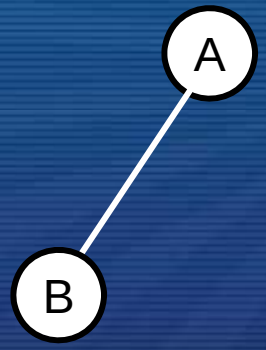
تفاوتهای درخت عادی با درخت دودویی

- یک درخت دودویی ممکن است شامل صفر گره باشد .
- در یک درخت دودویی ترتیب فرزندان دارای اهمیت بوده درحالی که در درخت عادی به این صورت نیست. .

الف:



ب :

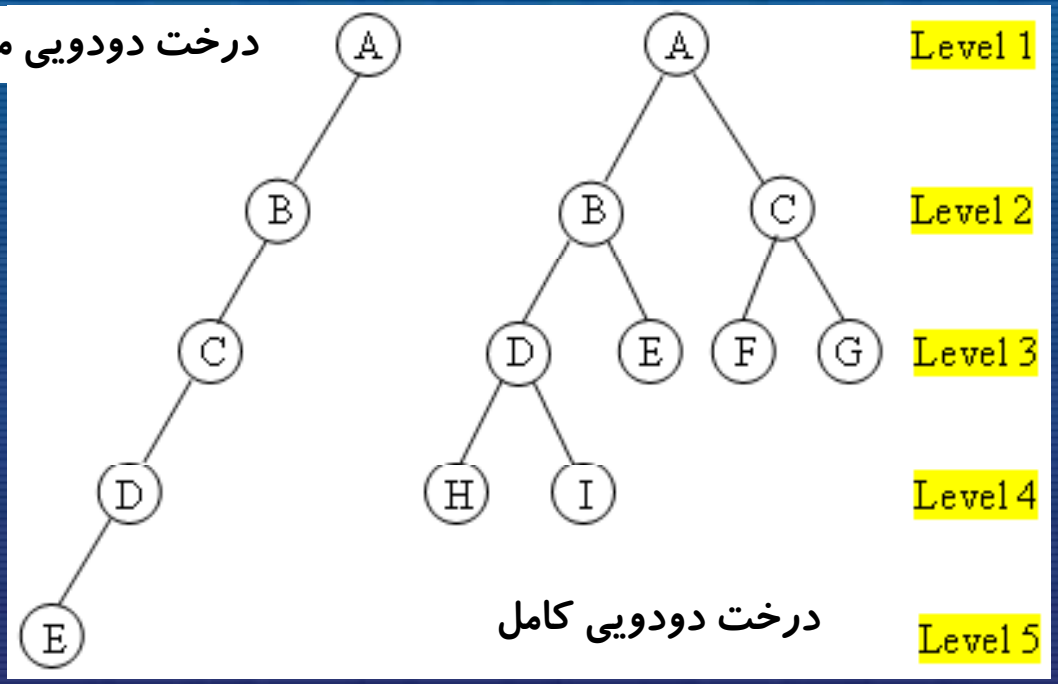


دو درخت دودویی متمایز

- درخت دودویی (ب) دارای زیردرخت راست تهی است درحالی درخت دودویی (الف) دارای زیردرخت چپ تهی می‌باشد .

دو نمونه از درختان دودویی:

درخت دودویی مورب به چپ



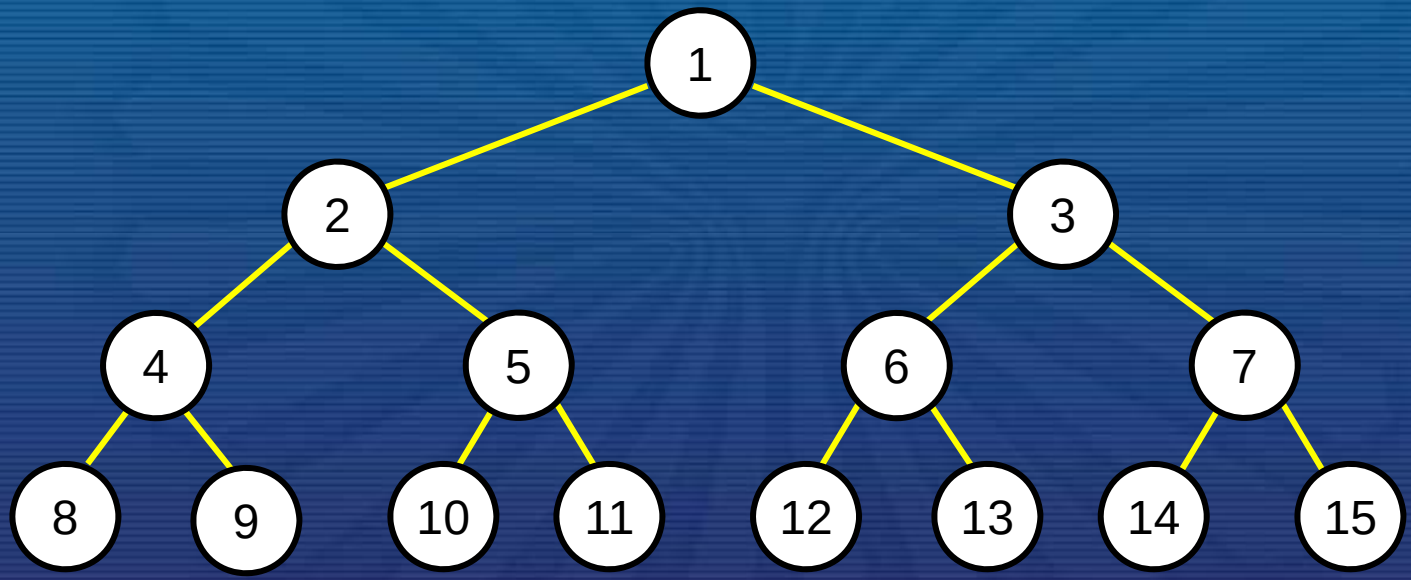
درخت دودویی کامل

خواص درختان دودویی

- حداکثر تعداد گره‌ها در سطح i ام یک درخت دودویی 2^{i-1} است ، که در آن $i \geq 1$ می باشد.
- حداکثر تعداد گره‌ها در یک درخت دودویی به عمق k، برابر 2^k است، که در آن $k \geq 1$. می باشد.
- برای هر درخت دودویی غیرتهی مانند T، اگر n_0 تعداد گره‌های پایانی و n_2 تعداد گره‌های درجه ۲ باشد، آنگاه خواهیم داشت $n_0 = n_2 + 1$.

درخت دودویی پر

- درخت دودویی پر با عمق k ، یک درخت دودویی است که دارای 2^{k-1} گره باشد $k \geq 0$.



یک درخت دودویی پر با عمق ۴

نمایش آرایه ای درخت دودویی

- به نحوه شماره گذاری درخت دودویی اسلاید قبل توجه کنید؛
- از آنجایی که گره‌ها از ۱ تا n شماره‌گذاری شده‌اند، یک آرایه یک‌بعدی می‌تواند برای ذخیره‌سازی گره‌ها استفاده شود (از موقعیت صفر آرایه استفاده نمی‌شود) .
- می توان به آسانی مکانهای پدر، فرزند چپ و فرزند راست هر گره را در آرایه مشخص کنیم.

اصل موضوعی

اگر یک درخت دودویی کامل با n گره (یعنی $[\log_2 n]+1$ = عمق) به ترتیب بالا تعریف شده باشد، آنگاه برای هر گره با اندیس i و $1 \le i \le n$ ، داریم :

۱- اگر $i \neq 1$ ، آنگاه پدر i در $[i/2]$ است. اگر $i = 1$ ، i ریشه است و پدری نخواهد داشت؛

۲- اگر $2i \le n$ ، آنگاه فرزند چپ i در $2i$ است. اگر $2i > n$ ، آنگاه i فرزند چپ ندارد؛

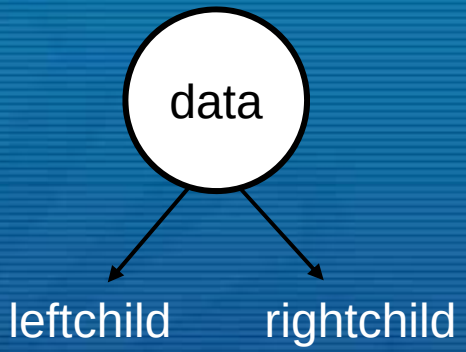
۳- اگر $2i + 1 \le n$، آنگاه فرزند راست i در $2i + 1$ است. اگر $2i+1 > n$، آنگاه i فرزند راست ندارد.

نمایش لیست پیوندی درخت دودویی

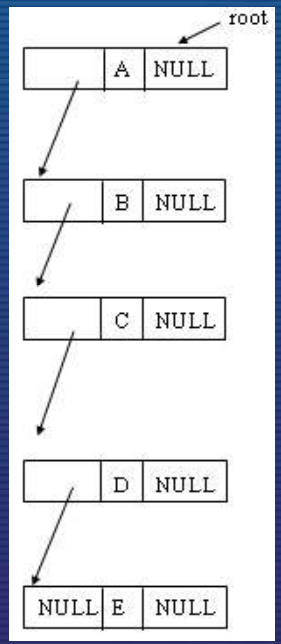
- نمایش ترتیبی(آرایه‌ای) برای درختان دودویی کامل مناسب به‌نظر می‌رسد، اما برای بسیاری از درختان دیگر باعث اتلاف حافظه می‌شود .

- با بکارگیری نمایش پیوندی، هر گره سه فیلد خواهد داشت: leftchild ، data و rightchild که در زبان C به شرح زیر تعریف می‌شوند:

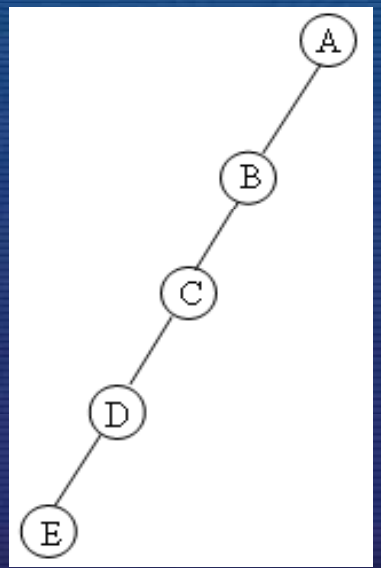
```
typedef  struct  node *treepointer ;
typedef  struct  node {
   int data ;
   treepointer leftchild , rightchild ;
};
```



یک درخت با متغیر (root) که به ریشه اشاره می‌کند، شناخته می‌شود :



نمایش لیست پیوندی



پیمایش درخت دودویی

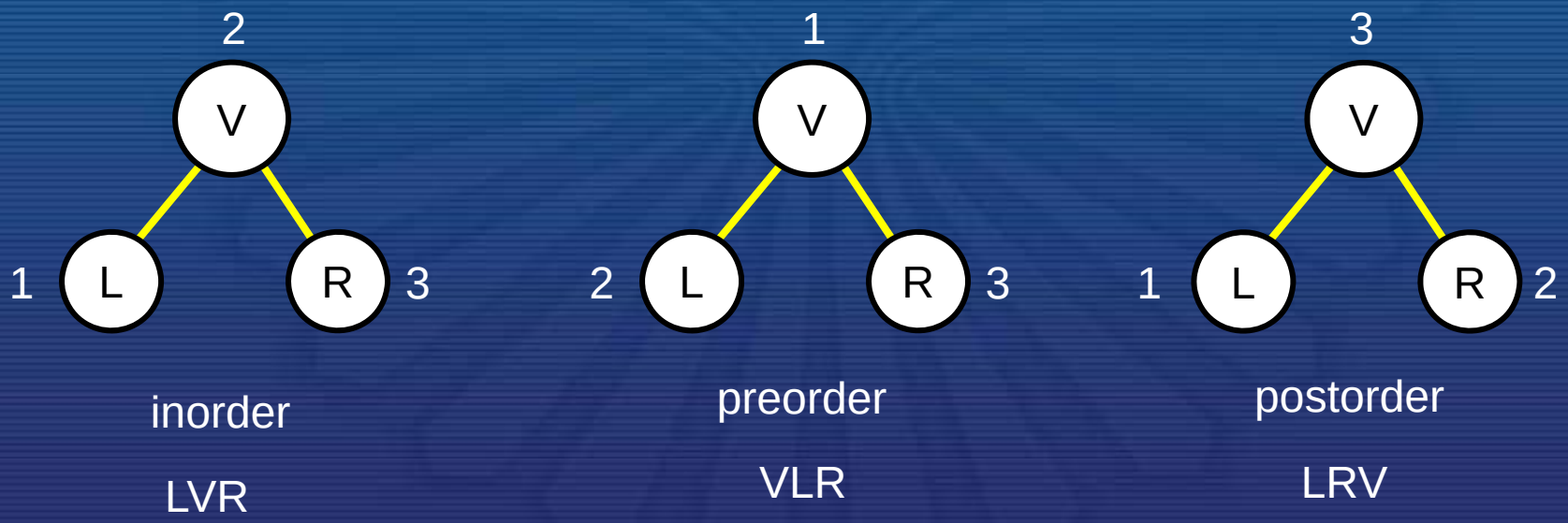
- عملکردی که معمولاً روی درختان دودویی صورت می گیرد، پیمایش درخت یا دستیابی به هر گره درخت فقط برای یکبار می‌باشد؛
- اگر L ، V و R به ترتیب حرکت به چپ، ملاقات کردن یک گره و حرکت به راست باشد، آنگاه شش ترکیب ممکن برای پیمایش یک درخت خواهیم داشت :

LVR ، LRV، VLR، VRL، RVL و RLV

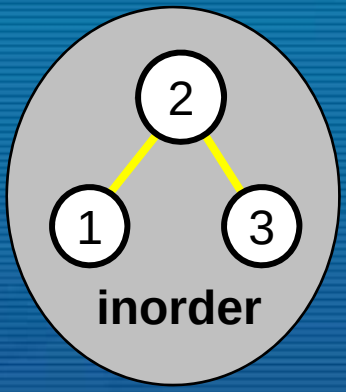
- اگر تنها حالتی را انتخاب کنیم که ابتدا به سمت چپ و بعد به سمت راست برویم، تنها سه ترکیبLVR ، LRV و VLR را خواهیم داشت؛

این سه حالت باتوجه به موقعیت V نسبت به L و R به ترتیب inorder ، postorder و preorder نامیده می شود .

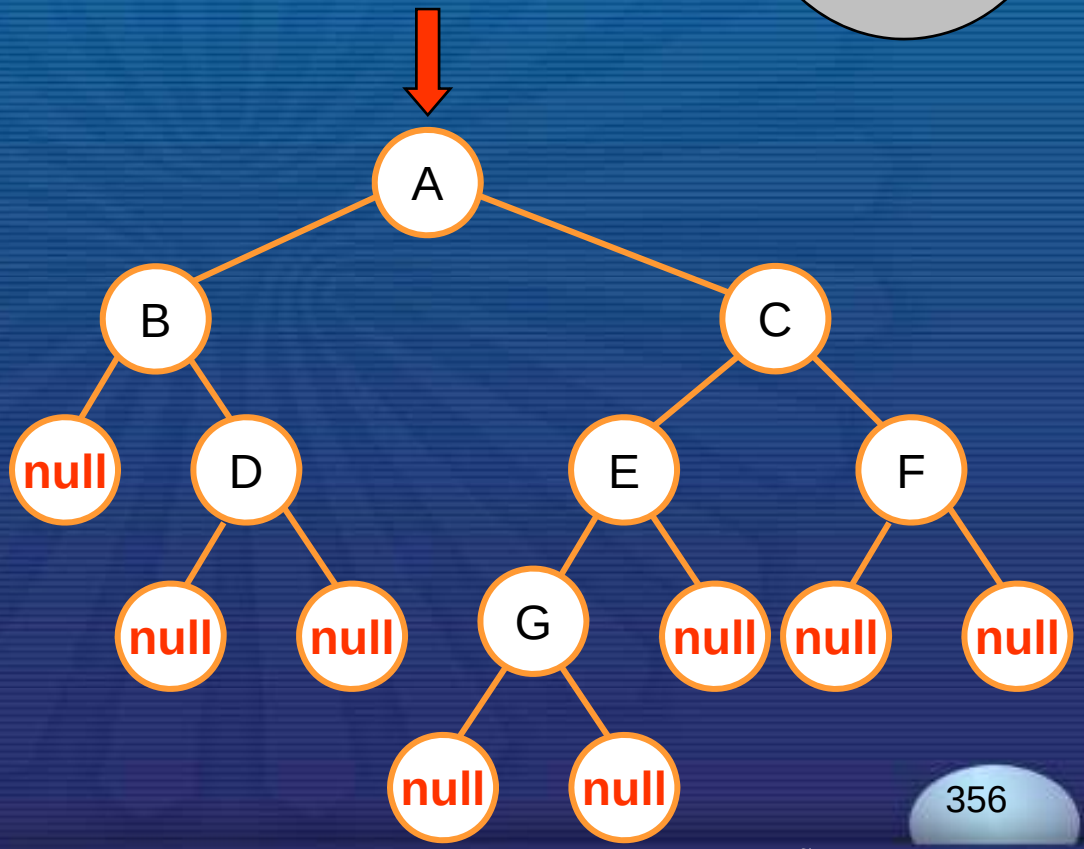
ترتیب ملاقات گره ها در روشهای پیمایش مختلف :



پیمایش Inorder

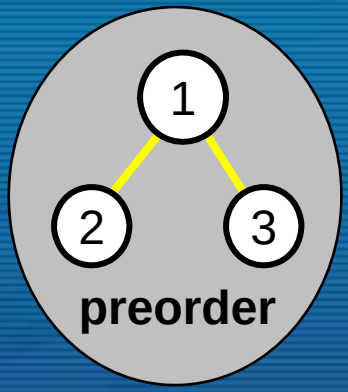



```
void  inorder (treepointer  ptr)
{
    if (ptr)
    {
        inorder (ptr->leftchild);

        printf ("%d", ptr->data);

        inorder (ptr->rightchild);
    }
}
```

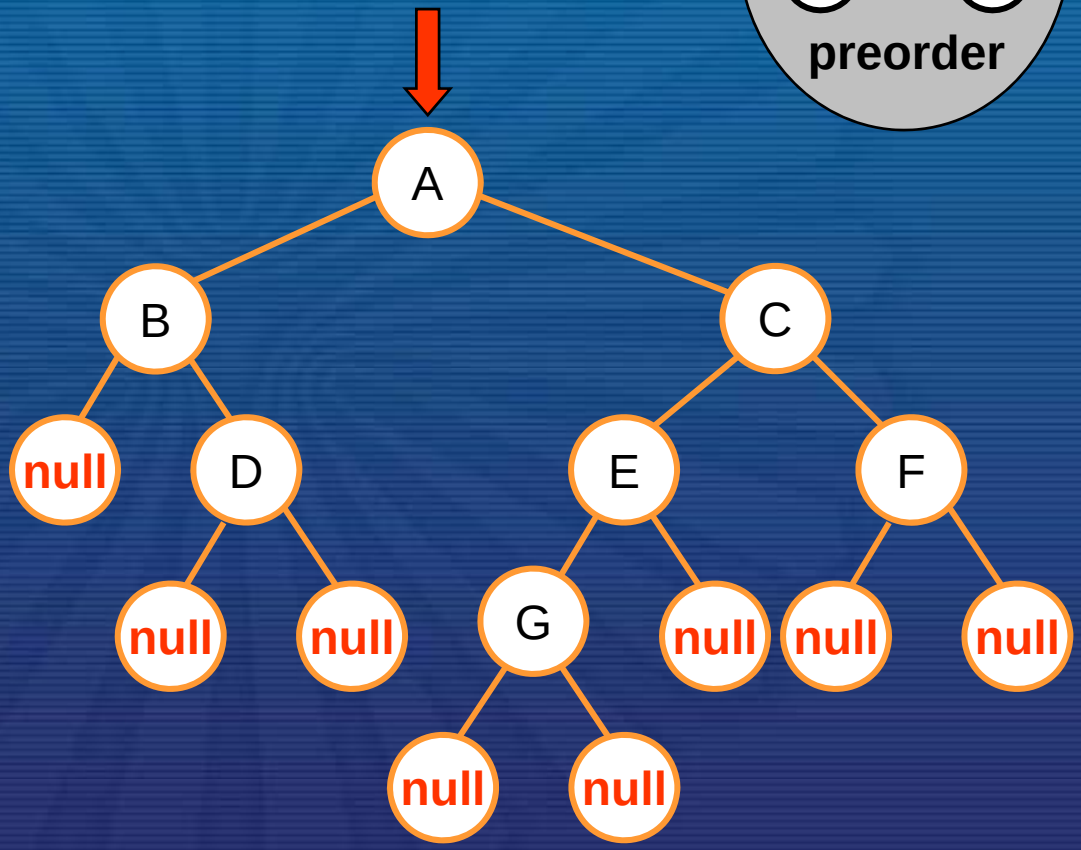


خروجی : B D A G E C F

پیمایش preorder

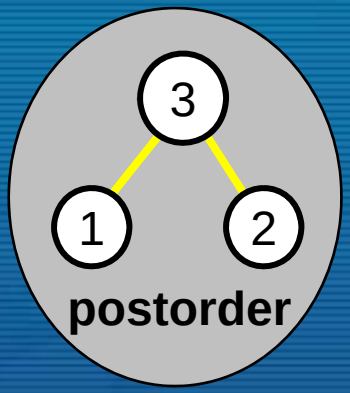



```
void preorder (treepointer  ptr)
{
   if  (ptr)
   {
       printf ("%d", ptr->data);
       preorder (ptr->leftchild);
       preorder (ptr->rightchid);
   }
}
```

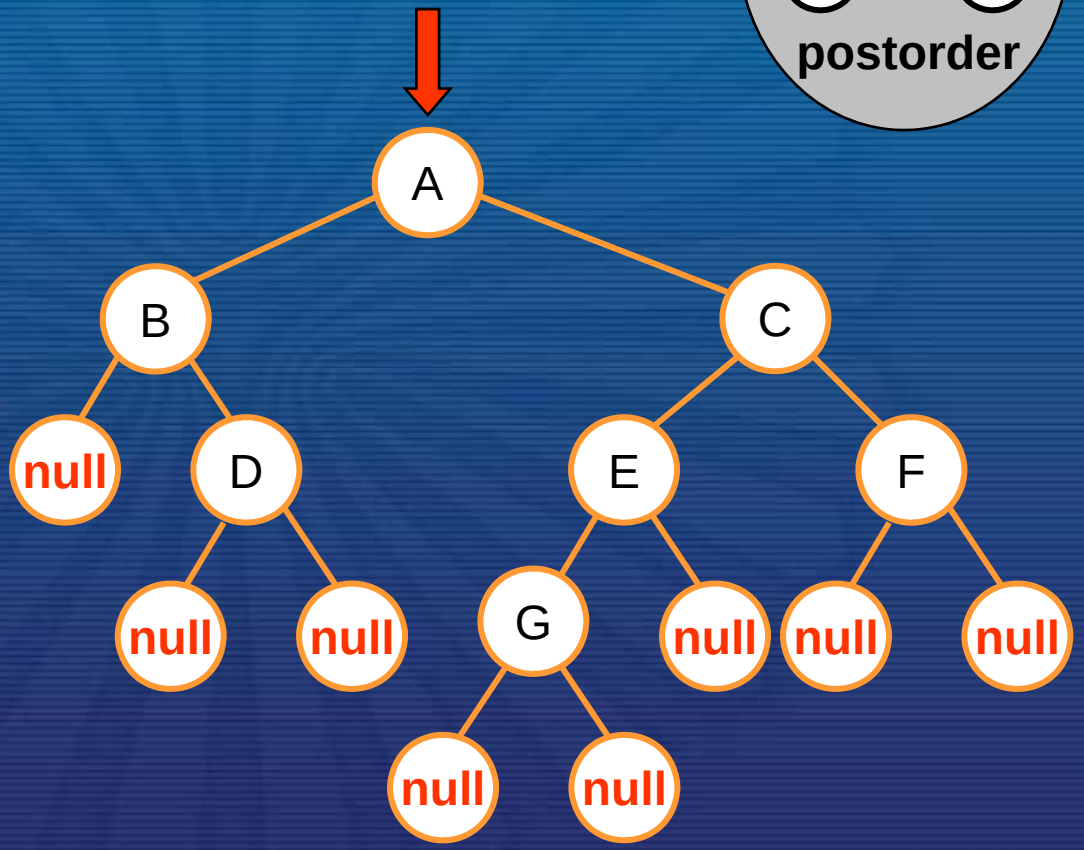


خروجی : A B D C E G F

پیمایش postorder




```
void  postorder (treepointer ptr)
{
   if (ptr)
   {
      postorder (ptr->leftchild);
      postorder (ptr->rightchild);
      printf "%d", ptr->data);
    }
}
```



خروجی : D B G E F C A

فصل یازدهم : ساختار درخت – پیمایش درخت دودویی

- درختان دودویی را درصورتی مساوی می‌نامیم که ساختاری نظیر هم داشته و اطلاعات موجود در گره‌های نظیرشان باهم برابر باشد .

- منظور از ساختار متناظر این است که هر انشعاب از درخت اول با انشعابی در درخت دوم مطابقت داشته باشد .

درختان جستجوی دودویی

- اگر مجموعه داده‌ها را به صورت یک درخت جستجوی دودویی در آوریم اعمال جستجو، حذف و درج بهینه تر و بهتر اجرا خواهند شد.

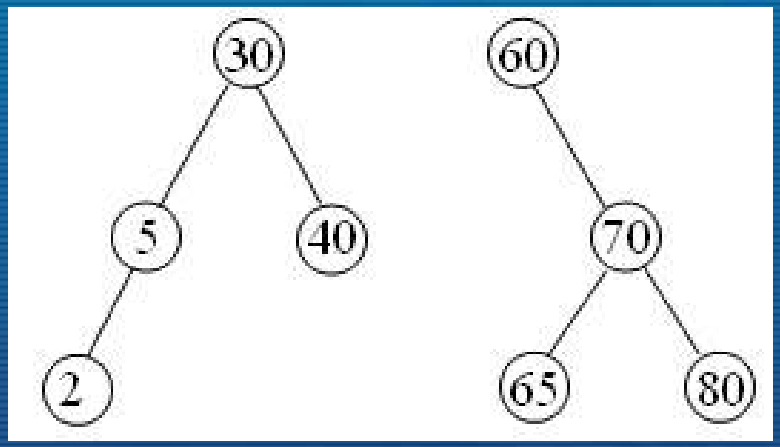
- هر عنصر دارای یک کلید است و دو عنصر نباید دارای یک کلیـد یکسـان باشـند، درواقع کلیدها منحصر بفرد هستند.

- کلیدهای واقع در زیردرخت غیرتهی چپ باید کمتر از مقدار کلید واقع در ریشـه زیردرخت راست باشد.

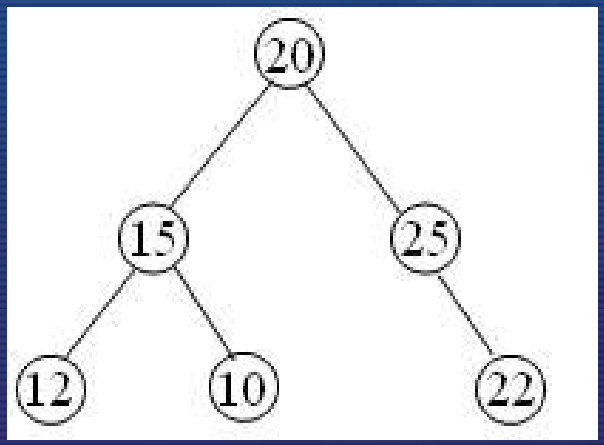
- کلیدهای واقع در زیردرخت غیرتهی راست باید بزرگتر از مقدار کلید واقع در ریشه زیردرخت چپ باشد .

- زیردرختان چپ و راست نیز خود درختان جستجوی دودویی می‌باشند.

دو نمونه از درختان جستجوی دودویی :

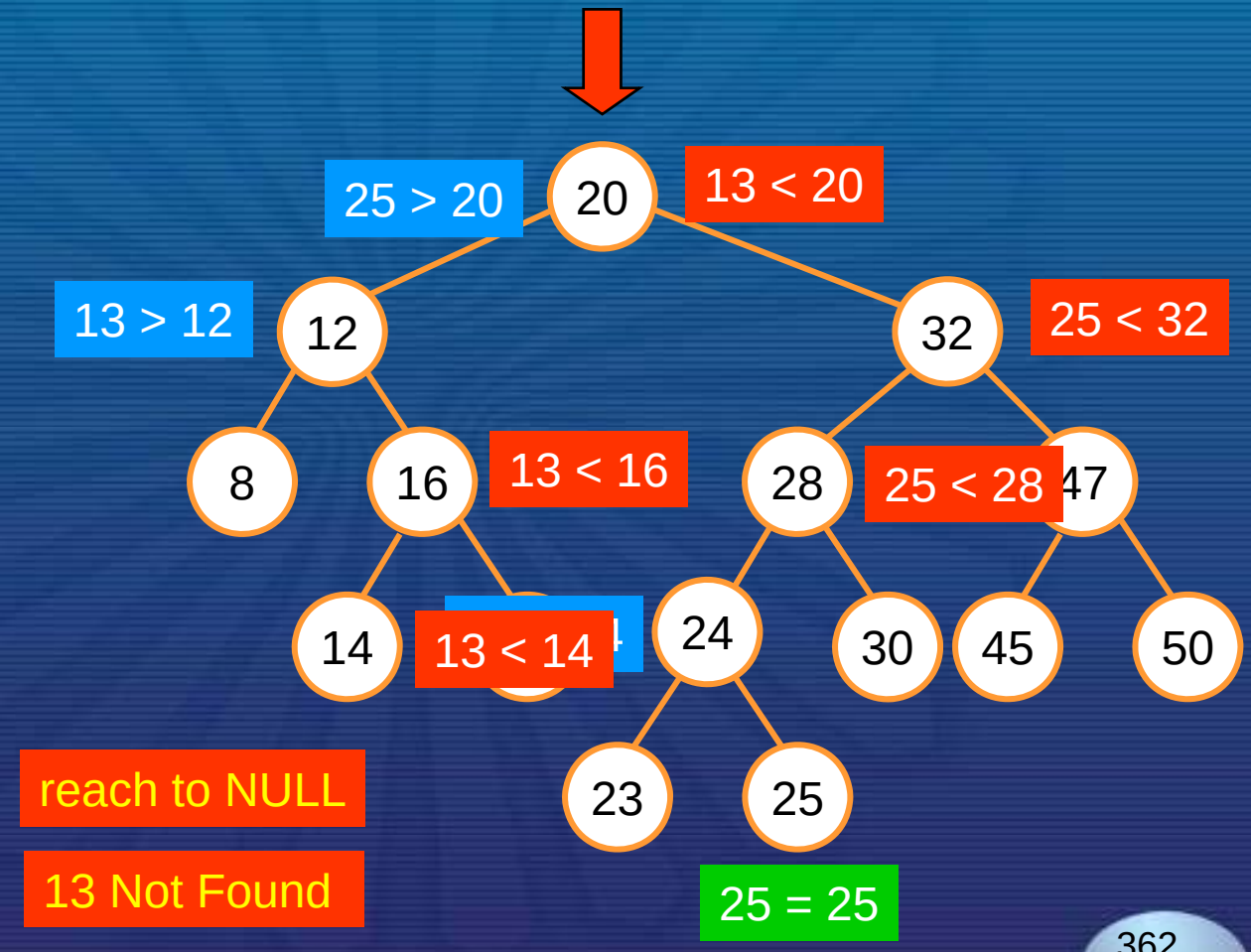


درخت مقابل یک درخت جستجوی دودویی نیست! (چرا؟)



جستجوی یک درخت دودویی

جستجوی عدد ۲۳



تابع search درخت جستجوی دودویی را به‌صورت بازگشتی جستجو می‌کند:

```
treepointer  search (treepointer  root , int  key)
{
   if (!root)
      return NULL ;
   if (key = = root->data)
      return  root ;
   if (key < root->data)
   return  search (root->leftchild , key) ;
   return  search (root-> rightchild , key) ;
}
```

تابع غیر بازگشتی برای جستجوی درخت جستجوی دودویی :

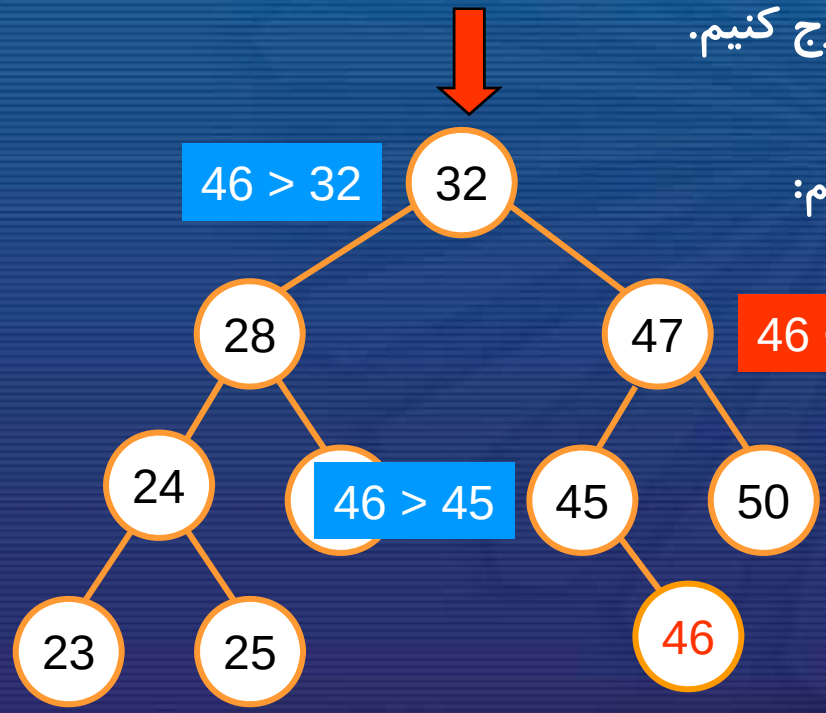
```
treepointer search2 (treepointer  tree , int  key)
{
   while (tree)
   {
      if (key = = tree->data) return  tree;
      if (key <  tree->data)
          tree = tree->leftchild ;
      else
          tree = tree->rightchild ;
   }
   return NULL ;
}
```

- اگر h ارتفاع یا عمق یک درخت جستجوی دودویی باشد، می‌توان با استفاده از search یا search2 عمل جستجو را در مدت O(h) انجام داد .

- تعداد مقایسه های لازم برای دستیابی به یک گره در درخت جستجوی دودویی، یک واحد بیشتر از سطح آن گره است .

درج عنصر در درخت جستجوی دودویی

- درخت جستجوی دودویی نمی تواند کلید تکراری داشته باشد،
- برای درج عنصر جدید باید مطمئن شویم که کلید ورودی در درخت وجود نداشته باشد، پس در درخت به جستجوی کلید ورودی می پردازیم در صورت ناموفق بودن جستجو می توانیم کلید جدید را در محلی که جستجو خاتمه پذیرفته درج کنیم.



برای مثال کلید 46 را به درخت زیروارد می کنیم:

کلید 46 را در درخت جستجو می کنیم؛

جستجو ناموفق است، پس کلید را در محل وارد می کنیم.

حذف عنصری از درخت جستجوی دودویی

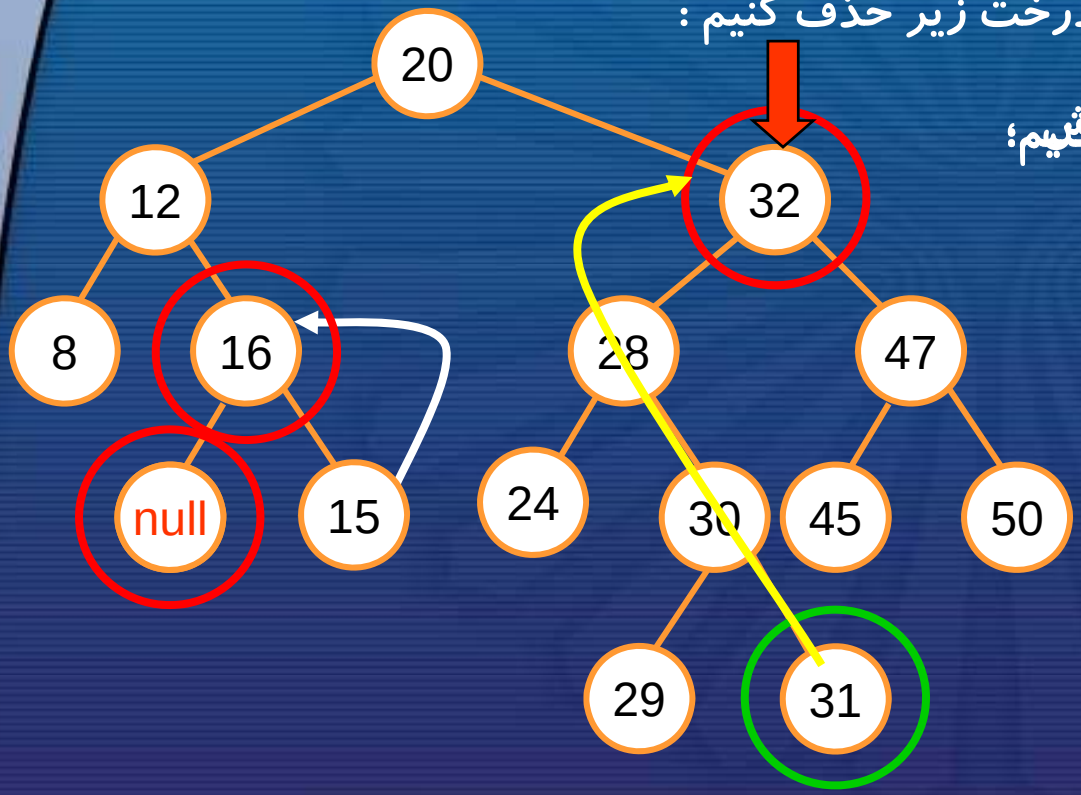
- حذف گره از درخت جستجوی دودویی را در ۳ حالت بررسی می کنیم :

۱- اگر گره مورد نظر برگ درخت باشد؛ در این صورت براحتی مقدار اشاره گر فرزند والد را NULL قرار می دهیم.

۲- اگر گره مورد نظر یک فرزند داشته باشد؛ دراین صورت مقدار فرزند در گره جایگزین شده و با گره فرزند (که یک برگ است) مشابه ۱ عمل می کنیم.

۳- اگر گره مورد نظر دو فرزند داشته باشد؛ دراین صورت گره را با بزرگترین عنصر در زیردرخت چپ و یا کوچکترین عنصر در زیردرخت راست آن گره جایگزین و تعویض می‌کنیم. سپس این فرآیند را با حذف این عنصر جایگزین از زیردرختی که از آن گرفته شده است، دنبال می‌کنیم.

می خواهیم عناصر ۱۴ و ۱۶ و ۳۲ را از درخت زیر حذف کنیم :



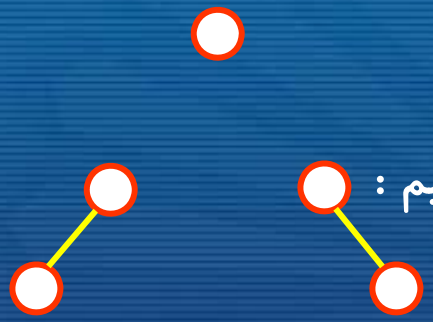
برای حذف گره حاوی عنصر ۳۲ عمل می‌کنیم؛
جایگزین می کنیم،

عمق یک درخت جستجوی دودویی

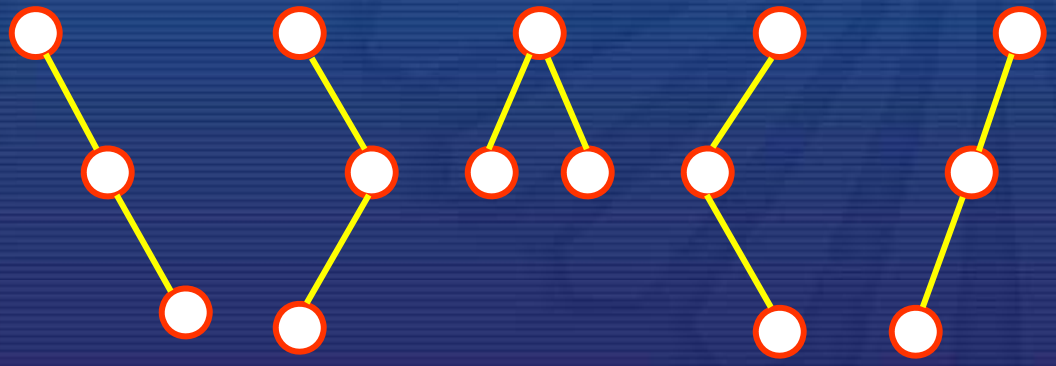
- اگر به اندازهٔ کافی دقت نشود، عمق یک درخت جستجوی دودویی با n عنصر می‌تواند به بزرگی n باشد؛
- درختان جستجو با بیشترین عمق $O(\log_2 n)$ ، درختان جستجوی متعادل نامیده می‌شوند .
- درختان جستجوی متعادلی وجود دارند که عمل جستجو، درج و حذف را در زمان $O(h)$ ممکن می‌سازند .

درختان دودویی متمایز

- اگر n ،تعداد گره ها در درخت باشد، می دانیم :
  - اگر n=0 یا n=1 باشد، آنگاه فقط یک درخت داریم :
  - اگر n=2 باشد، می‌توانیم دو درخت دودویی داشته باشیم :



  - اگر n=3 باشد، می‌توان ۵ درخت دودویی داشت و ... :



اصول کامپیوتر ۲

رشته علوم کامپیوتر

۴ واحد درسی

فصل دوازدهم

نام منبع و مؤلف :

- ✓ اصول کامپیوتر ۲
- ✓ دکتر داود کریم زادگان مقدم
- ✓ انتشارات دانشگاه پیام نور ۱۳۸۳

فصل دوازدهم : فایل

اهداف کلی و رفتاری

مقدمه

دستگاههای ورودی و خروجی استاندارد

انواع فایل

باز کردن و بستن فایل

توابع کار با فایل ها

فایلهای ورودی و خروجی

هدف کلی

آشنایی با انواع فایل و نحوه استفاده از توابع مربوط به آنها

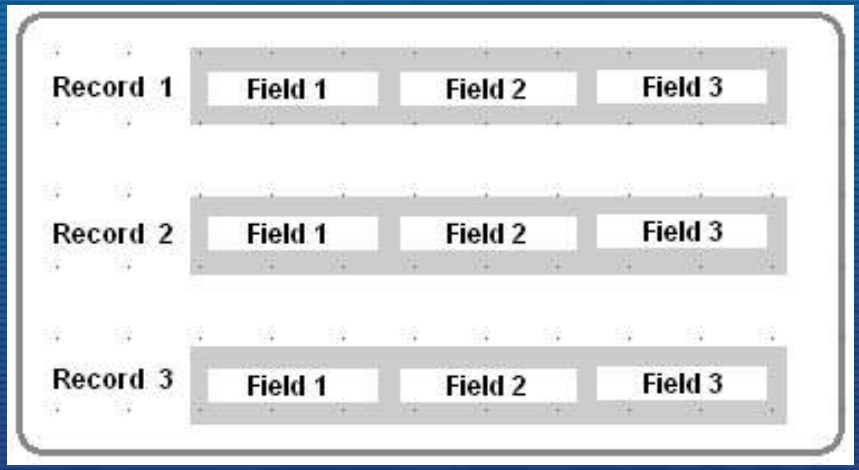
هدف های رفتاری

- تعریف فایل، رکورد و فیلد
- انواع فایل و نحوهٔ ذخیره و بازیابی داده‌ها در فایل
- نحوه باز کردن و بستن فایل ها
- نحوه استفاده از انواع توابع مربوط به فایل ها
- دستگاههای ورودی و خروجی استاندارد

مقدمه Introduction

- فایل نوعی ساختمان داده است که بر روی حافظه جانبی مثل دیسک ، نوار و غیره تشکیل می‌گردد.
- هر فایل شامل مجموعه‌ای از داده‌های مرتبط به هم است.
- داده‌های مربوط به هریک از اجزای فایل ، یک رکورد نام دارد .

به هریک از اجزای یک رکورد ، فیلد گفته می‌شود . لذا می‌توان گفت که هر کورد مجموعه‌ای از چند فیلد است.



فایل حاوی چند رکورد و فیلد

داده‌ها ممکن است به چهار روش در فایل ذخیره شده و سپس مورد بازیابی قرار گیرند:

- داده‌ها، کاراکتر به کاراکتر در فایل نوشته شده و سپس کاراکتر به کاراکتر از فایل خوانده شوند .
- داده‌ها به صورت رشته‌ای از کاراکترها ، در فایل نوشته شده و سپس به صورت رشته‌ای از کاراکترها مورد دسترسی قرار گیرند .
- داده‌ها در حین نوشتن بر روی فایل ، با فرمت خاصی نوشته شده و سپس با همان فرمت خوانده شوند.
- داده‌ها به شکل ساختمان (رکورد) بر روی فایل نوشته شده و سپس به صورت ساختمان از فایل خوانده شوند .

انواع فایل

- این دو نوع فایل (روش ذخیره سازی) در موارد زیر با هم تفاوت دارند :
  - تعیین انتهای خط .
  - تعیین انتهای فایل .
  - نحوهٔ ذخیره شدن اعداد بر روی دیسک .

- دسترسی به اطلاعات موجود در فایلهای text کندتر از فایلهای باینری است .

- اختلاف دیگر فایلهایtext و باینری در تشخیص انتهای فایل است در حالت text کاراکتر ۲۶ مشخص‌کنندهٔ انتهای فایل است .
(این کاراکتر با فشار دادن کلید Ctrl به همراه کلید Z تولید می‌شود)

- در فایل باینری ممکن است عدد 1A (در مبنای ۱۶) و یا ۲۶ (در مبنای ۱۰) جزئی از اطلاعات بوده ، بیانگر انتهای فایل نباشد .

از نظر نحوهٔ ذخیره و بازیابی داده‌ها در فایل دو روش وجود دارد :

- سازمان فایل ترتیبی . (sequential)
- سازمان فایل تصادفی . (random)

بازکردن و بستن فایل

- هر فایل قبل از اینکه بتواند مورد استفاده قرار گیرد ، باید باز گردد .
- مواردی که در حین بازکردن فایل باید مشخص شوند عبارتند از:
  - نام فایل .
  - نوع فایل ازنظر ذخیرهٔ اطلاعات(text یا binary)
  - نوع فایل ازنظر ورودی ـ خروجی .

- اگر فایلی قبلاً وجود نداشته باشد، در حین بازشدن باید به‌عنوان فایل خروجی باز شود .
- اگر فایلی قبلاً وجود داشته باشد و به‌عنوان خروجی بازگردد ، اطلاعات قبلی آن از بین می‌رود .
- تابع fopen برای باز کردن فایل مورد استفاده قرار گرفته و دارای الگوی زیر است:

```
FILE   *fopen (char *filename , *mode)
```

مقادیری که می‌توانند بجایmode در تابع fopen قرار گیرند:

mode	مفهوم
r (rt)	فایلی از نوع text را بعنوان ورودی باز می‌کند .
w (wt)	فایلی از نوع text را بعنوان خروجی باز می‌کند .
a (at)	فایل را طوری باز می‌کند که بتوان اطلاعاتی را به انتهای آن اضافه نمود .
rb	فایلی از نوع باینری را بعنوان ورودی باز می‌کند .
wb	فایلی از نوع باینری را بعنوان خروجی باز می‌کند .
ab	فایل موجود از نوع باینری را طوری باز می‌کند که بتوان اطلاعاتی را به انتهای آن اضافه نمود .
r + (r+t)	فایل موجود از نوع textرا بعنوان ورودی و خروجی باز می‌کند .
w + (w+t)	فایلی از نوع text را بعنوان ورودی و خروجی باز می‌کند .
a + (a+t)	فایل موجود از نوع text را بعنوان ورودی و خروجی باز می‌کند .
r + b	فایل موجود از نوع باینری را بعنوان ورودی و خروجی باز می‌کند .
a + b	فایل احتمالاً موجود از نوع باینری را بعنوان ورودی و خروجی باز می‌کند .
w + b	فایل از نوع باینری را بعنوان ورودی و خروجی باز می‌کند .

- برای تشخیص این که آیا فایل با موفقیت باز شده است یا خیر می‌توان اشاره‌گر فایل را با NULL مقایسه کرد.

- اگر اشاره‌گر فایل برابر باNULL باشد بدین معنی است که فایل باز نشده است :

```
if (( fp= fopen ("A : test" , "w"))== NULL )
 {
   printf ("cannot open file \ n") ;
   exit (0) ;
 }
```

- بعد از استفاده از فایل و قبل از خروج از برنامه باید، فایل توسط برنامه بسته شود.
- بستن فایل توسط تابع fclose انجام می‌شود که دارای الگوی زیر است :

```
int  fclose (FILE  *fp) ;
```

- اگر فایلهای زیادی همزمان در برنامه باز باشند می توان با تابع زیر همه ی آنها را باهم بست :

```
fcloseall() ;
```

توابع کار با فایلها

توابع putc و getc

◄ برای نوشتن یک کاراکتر در فایلی که قبلا باز شده است، از توابع putc و fputc استفاده می‌شود .

الگوی تابع putc به صورت زیر است :

```
int  putc (char   ch , FILE *fp)
```

◄ برای خواندن کاراکترها از فایل ، می‌توان از دو تابع getc وfgetc استفاده نمود .

الگوی تابع getc به صورت زیر است :

```
int  getc (FILE  *fp)
```

- موقعیت سنج فایل هر لحظه نشان می دهد که داده ها از کجای فایل باید خوانده شوند، و یا کجای فایل باید نوشته شوند.
- در هنگام خواندن داده‌ها از فایل باید بتوان انتهای فایل را بررسی نمود.
- در حین خواندن داده‌ها از فایل text پس از رسیدن به انتهای فایل تابع getc یا fgetc علامت EOF را برمی‌گرداند .
- در فایل باینتری و متنی برای تست کردن انتهای فایل از تابع feof استفاده می‌گردد :

```
int  feof (FILE  *fp)
```

مثال : برنامه زیر کاراکترهایی را از ورودی خوانده و در یک فایل متنی قرار می دهد . سپس داده‌های موجود در این فایل را خوانده و به فایل دیگری منتقل می کند . آخرین کاراکتر ورودی، نقطه در نظر گرفته شده است:

```
#include<stdio.h>
#include<stdlib.h>
void  main (void)
{
  FILE  *in , *out ;
  char ch ;
  in = fopen ("F1.txt" , "w") ;
  if  ( in = = NULL )
  {
     printf ("cannot open  F1.txt \n") ;
     exit(1) ;
   }
  do
  {
     ch = getchar( ) ;
     putc (ch , in) ;
  } while (ch ! = '.') ;
  fclose(in) ;
```

```
out = fopen ("F2.txt" , "w") ;
if ( out = = NULL )
{
   printf ("cannot open  F2.txt \n") ;
   exit(1) ;
}
in = fopen ("F1.txt " , "r") ;
if  ( in = = NULL )
{
   printf ("can not open  F1.txt \n") ;
   exit(1) ;
 }
 ch = getc(in) ;
 while (ch! = EOF)
 {
     putc(ch , out) ;
     ch = getc (in) ;
 }
 fclose(in) ;
 fclose(out) ;
}
```

توابع putw و getw

این دو تابع مشابه getc و putc هستند، با این تفاوت که برای خواندن و نوشتن مقادیر صحیح از یک فایل به یک فایل دیگر بکار برده می شوند.

دستور زیر عدد صحیح ۵۰ را در فایلی که fp به آن اشاره می کند می نویسد :

```
putw(50 , fp) ;
```

توابع fputs و fgets

برای نوشتن رشته‌ها در فایل ، از تابع fputs و برای خواندن رشته‌ها از فایل ، از تابع fgets استفاده می‌گردد:

```
int  fputs (const char *str , FILE *fp)
char *fgets (char *str , int  length , FILE *fp)
```

فایل های ورودی و خروجی

می‌توان یک فایل را هم به عنوان وسیلهٔ ورودی و هم به عنوان وسیلهٔ خروجی مورد استفاده قرار داد برای این منظور کافی است در تابع fopen بجای mode از یکی از عبارات زیر استفاده شود :

r + t یا r+

برای باز کردن فایل text موجود بعنوان ورودی استفاده می شود .

w+ یا w + t

برای ایجاد یک فایل text به عنوان خروجی استفاده شود .

a+ یا a + t

برای ایجاد فایل text و یا باز کردن فایل text موجود ، به عنوان ورودی و خروجی استفاده می شود.

r + b

برای باز کردن فایل باینری موجود ، بعنوان ورودی استفاده می شود .

w + b

برای ایجاد یک فایل باینری بعنوان خروجی استفاده می شود.

a + b

برای ایجاد و یا بازکردن فایل موجود باینری بعنوان ورودی و خروجی استفاده می شود.

توابع دیگر

تابع rewind()

تابع زیر موقعیت سنج فایل را طوری تغییر می دهد که به اول فایل اشاره کند :

```
void  rewind (FILE  *fp)
```

تابع remove()

برای حذف فایلهای غیرضروری می‌توان از تابع remove استفاده کرد . الگوی این تابع در فایل stdio .h قرار داشته و به صورت زیر است :

```
int  remove (char *filename)
```

توابع fprintf و fscanf

این دو تابع فرمت دار ورودی و خروجی همانند printf و scanf کار می کنند با این تفاوت که بر روی فایل ها استفاده می شوند :

```
int  fprintf (FILE *fp , "*control_string , …" , char arg , …)

int  fscanf (FILE *fp , "*control_string , …" , char arg , …)
```

این دو تابع از سرعت کمی برخوردارند که توصیه می‌شود از آنها استفاده نگردد .

توابع fwrite و fread

برای ورودی خروجی رکورد و همچنین سایر ورودی خروجیها می‌توان از دو تابع fread و fwrite استفاده نمود که از سرعت بالایی برخوردارند .

```
int  fread (void *buffer , int num_byte , int count , FILE *fp)
int  fwrite (void *buffer , int num_byte , int count , FILE *fp)
```

در الگوهای فوق ، پارامتر buffer در مورد تابع fread به ساختمان داده یا متغیری اشاره می‌کند که داده‌های خوانده شده از فایل باید در آن قرار گیرند و در تابع fwrite به محلی از حافظه اشاره می‌کند که داده‌های موجود در آن محل باید در فایل نوشته شوند.

تابع fseek

برای دستیابی به داده های توابع به صورت تصادفی ، استفاده می شود.

الگوی این تابع به صورت زیر است:

```
int  fseek( FILE  *fp , long  int  num_bytes , int  origin ) ;
```

- این تابع موقعیت سنج فایل را به اندازه ی num_bytes و نسبت به محل origin به جلو می برد.
- origin یک ماکروی از قبل تعریف شده است که یکی از مقادیر زیر را می تواند قبول کند:

Origin	نام ماکرو	مقدار ماکرو
شروع از ابتدای فایل	SEEK_SET	0
از موقعیت جاری	SEEK_CUR	1
انتهای فایل	SEEK_END	2

دستگاههای ورودی و خروجی استاندارد

- وقتی اجرای یک برنامه به زبان C آغاز می‌شود ، پنج فایل بطور اتوماتیک باز می‌شوند که اشاره‌گرهای آنها در جدول زیر مشاهده می‌گردند .

اشاره‌گر فایل	نام دستگاه (فایل)
stdin	دستگاه ورودی استاندارد (صفحه کلید)
stdout	دستگاه خروجی استاندارد (صفحه نمایش)
stderr	دستگاه استاندارد جهت ثبت پیامهای خطا (صفحه نمایش)
stdprn	دستگاه استاندارد چاپ (چاپگر موازی)
stdaux	پورت سری (serial port)

اصول کامپیوتر ۲

رشته علوم کامپیوتر
۴ واحد درسی
فصل سیزدهم

نام منبع و مؤلف :

- ✓ اصول کامپیوتر ۲
- ✓ دکتر داود کریم زادگان مقدم
- ✓ انتشارات دانشگاه پیام نور ۱۳۸۳

فصل سیزدهم : توابع کتابخانه ای

توابع کنترل صفحه و فونت های نقطه پویا

توابع گرافیکی

توابع تبدیل نوع

توابع ریاضی

توابع کاراکتری

توابع رشته‌ای

هدف کلی

آشنایی با برخی توابع کتابخانه ای متداول در زبان C و نحوه استفاده از آنها

هدف های رفتاری

- آشنایی با توابع کتابخانه ای تبدیل نوع و نحوه استفاده از آنها
- آشنایی با توابع کتابخانه ای ریاضی و نحوه استفاده از آنها
- آشنایی با توابع کتابخانه ای کاراکتری و نحوه استفاده از آنها
- آشنایی با توابع کتابخانه ای رشته ای و نحوه استفاده از آنها
- آشنایی با توابع کتابخانه ای تخصیص حافظه پویا و نحوه استفاده از آنها
- آشنایی با توابع گرافیکی و نحوه استفاده از آنها

مقدمه Introduction

- زبان C با یکسری توابع کتابخانه‌ای که عملیات و محاسبات پر کاربرد را انجام می‌دهند کامل می‌شود.
- معمولاً عملیاتی که وابسته به نوع کامپیوتر است به صورت توابع کتابخانه‌ای نوشته می‌شوند.
- هر تابع الگویی دارد که نوع تابع و نوع پارامترهای آن را مشخص می‌کند.

- الگوی هر تابع در فایل header مربوطه جای دارد.

تابع clrscr()

این تابع برای پاک کردن صفحهٔ نمایش در خروجی در مُد متنی به کار می‌رود و آرگومان ندارد. الگوی آن به شکل زیر است و در فایل conio.h قرار دارد:

```
void clrscr (void)
```

توابع تبدیل نوع

این توابع در زبانC برای انجام تبدیل نوع داده‌ها به یکدیگر به کار می‌روند و در فایل stdlib.h قرار دارند .

تابع atoi()

این تابع برای تبدیل نوع رشته به نوع integer به کار می‌رود. الگوی آن به صورت زیر است:

```
int atoi (const char *s)
```

تابع atof()

این تابع برای تبدیل نوع رشته به نوع ممیز شناور به کار می‌رود. الگوی آن به صورت زیر است:

```
double atof (const char *s)
```

تابع atol()

این تابع برای تبدیل نوع رشته به نوعlong به کار می‌رود. الگوی آن به صورت زیر است:

```
long atol (const char *s)
```

توابع ریاضی

این توابع در زبانC برای انجام برخی اعمال ریاضی مانند محاسبات مثلثاتی و عددی مورد استفاده قرار می‌گیرند. الگوی اغلب آنها در فایلmath.h قرار دارد.

تابع sqrt ()

این تابع جذر عدد مثبت را محاسبه می‌کند و الگوی آن به صورت زیر است:

```
double sqrt (double x)
```

تابع pow ()

این تابع توانهای یک مبنا را محاسبه می‌کند و الگوی آن به صورت زیر است:

```
double pow (double x , double y)
```

تابع abs ()

این تابع برای محاسبهٔ قدرمطلق اعداد صحیح به کار می‌رود. الگوی این تابع در فایل stdlib.h و نیز فایلmath.h وجود دارد و به صورت زیر است:

```
int abs (int x)
```

تابع sin ()

این تابع برای محاسبهٔ سینوس زاویه برحسب رادیان به کار می‌رود و دارای الگوی زیر است:

```
double sin (double arg)
```

تابع cos ()

این تابع برای محاسبهٔ کسینوس زاویه برحسب رادیان به کار می‌رود و الگوی آن به صورت زیر است:

```
double cos (double arg)
```

تابع tan ()

این تابع برای محاسبهٔ تانژانت زاویه‌ای که برحسب رادیان است به کار می‌رود:

```
double tan (double arg)
```

توابع asin() و acos() و atan()

این توابع برای محاسبه معکوس توابع مثلثاتی به کار می روند:

```
double asin (double arg)
double acos (double arg)
double atan (double arg)
```

تابع atan2 ()

این تابع دو آرگومان را دریافت، اولین آرگومان را بر دومین آرگومان تقسیم و آرک‌تانژانت نتیجهٔ حاصل را محاسبه می‌کند. الگوی تابع به صورت زیر تعریف شده است :

```
double atan2 (double y , double x)
```

تابع () log

این تابع لگاریتم طبیعی یک عدد مثبت را محاسبه می‌کند:
(پایهٔ لگاریتم طبیعی عدد e =۷۱۸۲۰۰۰/۲ است)

```
double log (double x)
```

تابع () log10

این تابع لگاریتم مبنای ۱۰ اعداد مثبت را محاسبه می‌کند:

```
double log10 (double x)
```

توابع کاراکتری

توابع کاراکتری برای ورودی ـ خروجی کاراکترها و مقایسهٔ آنها با یکدیگر، و تبدیل حروف کوچک و بزرگ به یکدیگر استفاده می‌شود. الگوی این توابع در فایل ctype. h قرار دارد.

تابع tolower ()

این تابع کاراکتری را به عنوان آرگومان می‌پذیرد و آن را به حرف کوچک انگلیسی تبدیل می‌کند. الگوی تابع به صورت زیر است:

```
int tolower (int ch)
```

تابع toupper ()

این تابع کاراکتری را به عنوان آرگومان می‌پذیرد و آن را به حرف بزرگ انگلیسی تبدیل می‌کند. الگوی این تابع به صورت زیر است:

```
int toupper (int ch)
```

توابع رشته‌ای

این توابع معمولاً برای مقایسه و جستجوی کاراکترها یا رشته‌ای در رشتهٔ دیگر به کار می‌روند. الگوی توابع رشته‌ای در فایل"string. h" قرار دارد.

تابع strset ()

این تابع محتویات رشته را با کاراکتر پر می‌کند. الگوی این تابع به صورت زیر است:

```
strset (str , 'x') ;
```

این تابع رشتهٔ str را با x پر می‌کند.

تابع strnset ()

این تابع کاراکتری را به تعداد دفعات مشخصی در رشته کپی می‌کند. الگوی این تابع به صورت زیر است:

```
char *strnset (char *str , char ch , singned count)
```

تابع memchr ()

این تابع کاراکتری را در آرایه جستجو و محل اولین وقوع آن را مشخص می‌کند. اگر این کاراکتر پیدا شد، شمارهٔ آن محل را در اشاره‌گر کاراکتری قرار می‌دهد وگرنه کاراکتر تهی را در اشاره‌گر منظور می‌کند. الگوی تابع به صورت زیر است:

```
void *memchr (const void *buffer , int ch , unsigned count)
```

تابع memcpy ()

این تابع قسمتی از آرایه را در آرایهٔ دیگر کپی می‌کند. الگوی این تابع به صورت زیر است:

```
void *memcpy (void *to , const void *from , unsigned count)
```

تابع strcspn ()

این تابع رشته‌ای را در رشتهٔ دیگر جستجو می‌کند و اولین محل از این رشته را که حتی یکی از کاراکترهای رشتهٔ مورد جستجو در آن محل باشند، به عنوان نتیجهٔ عمل برمی‌گرداند:

```
int strcspn (const *str1, const *str2)
```

تابع strlwr ()

این تابع رشته‌ای را به عنوان آرگومان می‌پذیرد و کلیهٔ حروف بزرگ آن را به کوچک تبدیل می‌کند:

```
int strlwr (char *str)
```

تابع struper ()

این تابع در رشتهٔ کاراکتری، کلیهٔ حروف کوچک را به حروف بزرگ تبدیل می‌کند:

```
char *struper (char *str)
```

تابع strrev ()

این تابع کاراکترهای رشته را معکوس می‌کند؛ یعنی کاراکتر ابتدا را به انتهای آن منتقل می‌کند و این عمل را برای کلیهٔ کاراکترها انجام می‌دهد:

```
char *strrev (char *str)
```

تابع strncmp ()

این تابع تعداد مشخصی از کاراکترهای دو رشته را با یکدیگر مقایسه می‌کند (زیررشته‌ای از یک رشته را با زیررشته‌ای از رشتهٔ دیگر مقایسه می‌کند). الگوی این تابع به صورت زیر است:

```
int strncmp (const *str1 , const char *str2, unsigned count)
```

در الگوی فوق به تعدادcount کاراکتر، از دو رشتهٔstr1 وstr2 با شروع از ابتدای رشته‌ها با یکدیگر مقایسه می‌شوند و یک عدد صحیح به عنوان نتیجهٔ عمل برمی‌گرداند.

تابع strncpy ()

این تابع تعداد مشخصی از کاراکترهای رشته را در رشته‌ای دیگر کپی می‌کند. الگوی این تابع به صورت زیر است:

```
char *strncpy (char *str1 , const char *str2 , unsigned count)
```

توابع تخصیص حافظهٔ پویا

تابع free ()

این تابع موجب می‌شود تا حافظهٔ اخذ شده از سیستم به آن بازگردانده شود. الگوی تابع در فایل"stdlib. h" قرار دارد و به شکل زیر است:

```
void free (void *)
```

تابع malloc ()

این تابع برای اخذ حافظه از سیستم، در حین اجرای برنامه، به کار می‌رود و دارای الگوی زیر است:

```
void *malloc (unsigned size)
```

الگوی این تابع در فایل"stdlib. h" قرار دارد. میزان حافظه‌ای که این تابع از سیستم اخذ می‌کند با آرگومانsize مشخص می‌گردد. آدرس حافظهٔ گرفته شده از سیستم با تابع malloc در اشاره‌گر قرار می‌گیرد. اگر این اشاره‌گر تهی(\0) باشد، بدین معنی است که حافظهٔ لازم تخصیص نیافته است.

تابع realloc ()

این تابع برای تغییر میزان حافظهٔ اختصاص یافته با توابع تخصیص حافظه، مثلmalloc ، به کار می‌رود. الگوی این تابع به صورت زیر است و در فایل"stdlib. h" قرار دارد:

```
*realloc (void *ptr , unsigned size)
```

توابع گرافیکی

با توجه به کاربردهای متداول توابع گرافیکی، تعدادی از آنها در نرم افزار توربوC را، که الگوی آنها در کتابخانهgraphics.h قرار دارند، بررسی می‌کنیم.

تابع clreol()

این تابع موجب می‌شود که اطلاعات خط جاری از محل فعلی مکان‌نما تا انتهای خط جاری از چپ به راست پاک شود. الگوی آن به صورت زیر است :

```
void  clreol()
```

تابع delline()

این تابع موجب می‌شود که یک خط حذف شود. الگوی آن به صورت زیر است:

```
void delline()
```

تابع insline()

این تابع موجب می‌شود که یک خط خالی زیر خطی که مکان نما روی آن قرار دارد ایجاد شود و بقیهٔ خطوط نیز یک سطر به سمت پایین حرکت کنند. الگوی آن به شکل زیر است:

```
void insline()
```

تابع gotoxy()

این تابع موجب می‌شود که در محیط متن بتوان مکان‌نما را در محل دیگری از پنجرهٔ فعال قرار داد. الگوی آن به شکل زیر است که در آن x و y مختصات جدید مکان‌نما را تعیین می‌کند:

```
void gotoxy(int x , int y)
```

تابع moveto()

این تابع در محیط گرافیک، مکان‌نما را به محلی با مختصات x و y انتقال می‌دهد و الگوی آن به صورت زیر است.

```
void far moveto(int x , int y)
```

تابع getx() و gety()

این دو تابع مختصات جاری مکان‌نما را روی صفحهٔ گرافیکی برمی‌گردانند و الگوی آنها به صورت زیر است.

```
int far getx(void)

int far gety(void)
```

تابع textcolor()

این تابع رنگ متن را مشخص می‌کند. الگوی آن به صورت زیر است:

```
void textcolor(int color)
```

آرگومان این تابع مقادیر صفر تا ۱۵ را داراست.

تابع textbackground()

این تابع برای تعیین رنگ زمینهٔ متن به کار برده می‌شود. الگوی آن به صورت زیر است:

```
void textbackground(int color)
```

آرگومان این تابع مقادیر صفر تا ۶ را داراست .

تابع textmode()

این تابع برای تغییر حالت صفحه تصویر به کار می‌رود. الگوی آن به صورت زیر است:

```
void textmode(int mode)
```

آرگومان این تابع مقادیر 7,3,2,1,0 را داراست.

تابع line()

این تابع خطی از نقطه‌ای به نقطهٔ دیگر رسم می‌کند. الگوی آن به صورت زیر است:

```
void far line(int startx , int starty , int endx , int endy)
```

تابع lineto()

این تابع خطی از مکان جاری به نقطهٔ دیگر رسم می‌کند. الگوی آن به صورت زیر است:

```
void far lineto(int x , int y)
```

تابع circle()

این تابع دایره‌ای با مرکز و شعاع مشخص رسم می‌کند. الگوی آن به صورت زیر است:

```
void circle(int x , int y , int radius)
```

تابع ellips()

این تابع بیضی‌ای به مرکز x و y و دو شعاع xradius و yradius با رنگ جاری رسم می‌کند:

```
void far ellipse(int x , int y , int start , int end , int xradius , int yradius)
```

در این الگو start و end آغاز و پایان بیضی را بر حسب درجه نشان می‌دهد. در صورتی که اولی صفر و دومی 360 باشد، بیضی به شکل کامل نمایش داده می‌شود.

تابع arc()

این تابع کمانی از start تا end را در امتداد دایرهٔ فرضی با مختصات مرکز x و y و نیز شعاع radius رسم می‌کند و الگوی آن به صورت زیر است:

```
void far arc(int x , int y , int star , int end , int radius)
```

تابع bar()

این تابع میله‌ای مستطیل شکل رسم و با رنگ و الگوی جاری پر می‌کند. الگوی آن به صورت زیر است:

```
void far bar(int left , int top , int right , int bottom)
```

سایت مرجع دانشجوی پیام نور

- ✓ نمونه سوالات پیام نور : بیش از ۱۱۰ هزار نمونه سوال همراه با پاسخنامه تستی و تشریحی
- ✓ کتاب ، جزوه و خلاصه دروس
- ✓ برنامه امتحانات
- ✓ منابع و لیست دروس هر ترم
- ✓ دانلود کاملا رایگان بیش از ۱۴۰ هزار فایل مختص دانشجویان پیام نور