نکات اولیه :

- ✓ زبان **C** نسبت به حروف بزرگ و کوچک حساس است که اصطلاحاً Case Sensitive نامیده می شود.
- ✓ سعی کنید در نوشتن برنامه رعایت نظم را داشته باشید. تورفتگی جملات را در یک بلوک رعایت کنید تا برنامه خواناتر شود.
- ✓ برای نامگذاری متغیرها و شناسه ها بهتر است از اسامی معنی دار استفاده شود.
- ✓ برای کار با محیط کامپایلر می توانید به ضمیمه آخر این جزوه مراجعه نمایید.
- ✓ این زبان دارای برخی واژه های کلیدی است که باید آنها را به خاطر بسپارید.
- ✓ در هر برنامه نکاتی جدید مطرح می شود که با فرض یادگیری شما، آن نکات ممکن است در برنامه های بعدی به کار رود ولی دوباره شرح داده نمی شود. پس تا یک برنامه را کاملاً یاد نگرفته اید به سراغ برنامه بعدی نروید.
- ✓ کوچکترین اشکال تایپی در گرامر زبان ازجمله کاما، نقطه ویرگول، نقل قول تکی و دوتایی و ... موجب توقف برنامه خواهد شد.

برنامه شماره ۱ : این برنامه جملاتی را با رنگهای تعیین شده بر روی مانیتور منعکس می نماید.

```
/*-------------------------------------
  Programmer : Masood Jafari Nokandeh
  -------------------------------------*/
#include<stdio.h>
#include<conio.h>
main()
{
  textbackground(BLUE) ;
  textcolor(YELLOW) ;
  clrscr() ;
  printf("Wellcom to ") ;
  printf("BORUJERD\nWe learn C & C++") ;
  gotoxy(25,10) ;   //move cursor to line 10, column 25
  cprintf("Press any key to exit.") ;
  getch() ;  //stop monitor to view outputs
}
```

نکات آموزشی :

- ✓ توضیحات شخصی، حجم برنامه اجرایی exe. را افزایش نمی دهند و موجب خوانایی بهتر متن برنامه می شوند. این توضیحات به صورت تک سطری با علامت // و به صورت چند سطری بین علامات /* و */ ظاهر خواهند شد و در محیط ویرایشگر کامپایلر معمولاً کمرنگ تر از بقیه نوشته ها دیده می شوند.
- ✓ دستور include موجب اتصال فایلهای راهنما header به برنامه می شوند تا بتوانیم از توابع درون آنها استفاده کنیم. این فایلها متنی بوده و معمولاً پسوند h. دارند. برخی از مهمترین این فایلها بدین شرح می باشند :

  stdio مخفف standard input output برای عملیات پایه ورودی و خروجی.

  conio مخفف console input output برای توابع مربوط به رنگ و پاک کردن صفحه نمایش و حرکت مکان نما و ... .

  math مخفف mathematics برای توابع محاسباتی و ریاضی از قبیل قدر مطلق و جذر و سینوس و ... .

  stdlib مخفف standard library شامل کتابخانه استاندارد زبان سی.

  graphics برای برنامه نویسی و کار در محیط گرافیکی.

  ctype برای انجام برخی تبدیلات کاراکتری از جمله تبدیل حروف کوچک به بزرگ و ... .

  string برای پردازش رشته ها ازقبیل محاسبه طول رشته و مقایسه رشته ها و ... .
- ✓ برای استفاده از رنگها می توانیم کد عددی آن و یا شناسه ثابت آنرا با حروف بزرگ بنویسیم. رنگ زمینه بین ۰ تا ۷ و رنگ متن بین ۰ تا ۱۵ می باشد.
- ✓ در دستور printf الگوی خروجی بین دو علامت نقل قول دوتایی قرار می گیرد. این دستور در حالت عادی مکان نما را به ابتدای سطر بعدی منتقل نمی کند. این کار با علامت \n انجام می شود.
- ✓ تفاوت printf با cprintf در آن است که نوشته های رنگی ممکن است در اولی تأثیری نداشته باشد و محل مکان نما در نظر گرفته نشود ولی در دومی این مسئله رعایت می شود.
- ✓ تابع gotoxy فقط برای انتقال مکان نما به محل دلخواه است. اولین عدد بیانگر شماره ستون x و دومین عدد بیانگر شماره سطر y می باشد. صفحه نمایش در حالت عادی دارای ۸۰ سطر و ۲۵ ستون می باشد.

- ✓ وجود تابع getch در پایان برنامه فقط موجب توقف مانیتور و دریافت کلید می شود. این کار مشاهده خروجی را در زمان تست برنامه آسانتر می کند وگرنه وجود آن ضرورتی ندارد و می توانیم در هنگام تست برنامه در محیط کامپایلر از <alt>+<f5> استفاده نماییم.
- ✓ هر برنامه در زبان سی حد اقل دارای یک تابع اصلی main است که از آنجا شروع به کار می نماید. در هنگام معرفی یا فراخوانی توابع، حتماً وجود جفت پرانتز () لازم است حتی اگر آرگومان یا پارامتری وجود نداشته باشد.
- ✓ بلوک مجموعه ای از دستورات است که بین دو علامت { و } قرار می گیرد. در بلوک هر دستور حتماً دارای نقطه ویرگول است.

برنامه شماره ۲ : این برنامه دو عدد صحیح را دریافت نموده و مجموع آنها را نمایش می دهد.

```
#include<stdio.h>
#include<conio.h>
main()
{
  int a, b, c ;
  textattr(WHITE + (MAGENTA << 4) ) ;
  clrscr() ;
  printf("Enter two number : ") ;
  scanf("%d%d", &a, &b) ;
  c = a + b ;
  printf("---------------------------\n") ;
  printf("Sum = %d\n", c) ;
  printf("Multiplication = %d\n", a*b) ;
  getch() ;
}
```

نکات آموزشی :

- ✓ به جای استفاده از دو دستور جداگانه می توانید با یک تابع textattr ویژگی متن و زمینه را با هم تعیین نمایید. در این تابع فرمول کلی به یکی از این دو شکل خواهد بود :

```
Textattr( رنگ قلم + ( رنگ زمینه << 4 )) ;
Textattr( رنگ قلم + ( رنگ زمینه << 4 ) + 128 ) ;
```

- ✓ شما باید رنگ زمینه را ٤ بیت به چپ شیفت دهید تا به محل صحیح خود bbb هدایت شود. << بیانگر عمل شیفت محاسباتی به چپ است. به ازای هر شیفت به چپ، عدد مربوطه در ۲ ضرب می شود. پس اینجا رنگ زمینه در ١٦ ضرب می شود. در شکل زیر معلوم می شود که چگونه عدد یک بایتی مربوط به attribute تنظیم و رمزگذاری می شود :

7	6	5	4	3	2	1	0
B	b	b	b	f	f	f	f

ffff بیانگر رنگ قلم می باشد (از ۰ تا ۱۵). bbb بیانگر رنگ زمینه است. (از صفر تا ۷) B نیز برای حالت چشمک زن است. اگر بیت هفتم فعال باشد، کاراکترها روشن و خاموش می شوند. برای فعال کردن حالت BLINK باید خود آن یا معادلش عدد ۱۲۸ را به مقدار attribute اضافه کنید. مثلاً textcolor(CYAN + BLINK);

کد مربوط به هر رنگ را در جدول زیر نوشته ایم:

BLACK	۰	BROWN	٦	LIGHTRED	۱۲
BLUE	۱	LIGHTGRAY	۷	LIGHTMAGENTA	۱۳
GREEN	۲	DARKGRAY	۸	YELLOW	١٤
CYAN	۳	LIGHTBLUE	۹	WHITE	۱۵
RED	٤	LIGHTGREEN	۱۰		
MAGENTA	٥	LIGHTCYAN	۱۱	BLINK (چشمک زن)	۱۲۸

- ✓ کلمه int بیانگر داده های صحیح است. این داده ۲ بایت حافظه را اشغال می کند محدوده مجاز آن ۳۲۷٦۷ تا ۳۲۷٦۸- می باشد.
- ✓ تابع scanf برای دریافت مقادیر ورودی از صفحه کلید به کار می رود. و شکل کلی آن بدین صورت است :

```
scanf(" الگوی پیمایش " , &نام متغیر );
```

- ✓ در الگوی پیمایش %d بیانگر داده صحیح می باشد. عملگر & نشانی یا آدرس فیزیکی متغیر در حافظه را به دست می آورد. هرگز بین نام متغیر و علامت & فاصله ندهید همچنین در داخل نقل قول الگوی پیمایش هیچ فاصله ای نباشد.
- ✓ در تابع printf می توان از عبارات ثابت یا متغیر استفاده نمود :

```
printf(" الگوی چاپ " , عبارت );
```

- ✓ علامت = برای جایگزینی یا انتساب(assignment) است و برای مقایسه و برابری به کار نمی رود. همچنین عملگر + برای جمع دو عدد به کار می رود. عملگر * نیز برای حاصل ضرب می باشد. عملگر / تقسیم را انجام می دهد. اگر دو طرف صحیح باشند، فقط خارج قسمت را به دست می آورد. عملگر ٪ برای محاسبه باقیمانده تقسیم صحیح به کار می رود. عملگرهای ++ و -- برای افزایش یا کاهش یک واحدی متغیرها به کار می روند.

برنامه شماره ۳ : این برنامه چندین عمل محاسباتی را بر روی اعداد صحیح و اعشاری انجام می دهد. نتایج عملیات بسیار واضح اند و نیازی به توضیحات اضافی ندارند.

```
#include<stdio.h>
#include<conio.h>
main()
{
  int a=29 , b, c, d=10 ;
  float  e, f, g ;

  clrscr() ;
  b = 5 ;

  c = a % b ;   //remain of integer division. c=4
  printf("c = %d\n", c) ;

  e = a / b ;      //integer division. e=5
  printf("e = %f\n", e) ;

  f = float(a) / float(b) ; //real division. e=5.8
  printf("f = %f\n", f) ;

  g = 83.2 / 25 ;
  printf("g = %f\n", g) ;    // g=3.328000
  printf("g = %5.2f\n", g) ; // g=3.33

  d += 5 ;      // (d=15)  d=d+5 ;
  printf("d = %d\n", d) ;

  d -= 5 ;      // (d=10)  d=d-5;
  printf("d = %d\n", d) ;

  d *= 5 ;      // (d=50)  d=d*5;
  printf("d = %d\n", d) ;

 d /= 5 ;      // (d=10)  d=d/5;
  printf("d = %d\n", d) ;

  d++  ;      // (d=11)  d=d+1;
  printf("d = %d\n", d) ;

  getch() ;
}
```

نکات آموزشی :

- ✓ در هنگام معرفی متغیرها می توان به آنها مقدار اولیه داد مثلاً : `int a = 29 ;` . این مقدار در طول اجرای برنامه قابل تغییر است.
- ✓ داده `float` برای اعداد حقیقی از نوع معمولی می باشد و ٤ بایت حافظه مصرف می نمایند. الگوی چاپ و پیمایش آنها `%f` می باشد.
- ✓ استفاده بجا از سطرهای خالی در متن برنامه، خوانایی آن را افزایش می دهد.
- ✓ در عملگر / هرگاه دو عامل سمت چپ و راست هر دو از نوع صحیح باشند، حاصل عبارت برابر است با خارج قسمت تقسیم صحیح (رجوع کنید به مقدار متغیر e). برای رفع این مشکل از نشانه های تبدیل نوع یا `type casting` استفاده می شود چون هر گاه یکی از دوعامل تقسیم از نوع اعشاری باشد، حاصل عبارت از نوع اعشاری خواهد بود (رجوع کنید به مقدار متغیر f).
- ✓ در هنگام نمایش محتوای متغیر g که از نوع اعشاری است، در دستور `printf` از الگوی `%5.2f` برای قالب بندی خروجی استفاده شده است که بیانگر آن است که فضای صحیح ٥ رقمی است و فضای اعشاری ۲ رقمی است که با توجه به رقم سوم آن گرد می شود.

برنامه شماره ۴ : این برنامه تفاوت تقدم به کارگیری عملگرهای ++ یا -- را قبل و بعد از نام متغیر نشان می دهد. نتایج عملیات بسیار واضح اند و نیازی به توضیحات اضافی ندارند.

```
#include<stdio.h>
#include<conio.h>
main()
{
   int a=10, b=10, c, d ;
   clrscr() ;
   /*-------------------------
      first  c=a;      c=10
      then   a=a+1;    a=11
   -------------------------*/
   c = a ++ ;
   printf("a= %d  c= %d\n", a, c) ;

   /*-------------------------
      first  b=b+1;    b=11
      then   d=b;      d=11
   -------------------------*/
   d = ++ b ;
   printf("b= %d  d= %d\n", b, d) ;
   getch() ;
}
```

برنامه شماره ۵ : این برنامه شعاع یک دایره را دریافت داشته سپس محیط و مساحت آنرا نشان می دهد.

```
#include<stdio.h>
#include<conio.h>
#define P 3.14

main()
{
  float r, a, s ;
  textbackground(BLUE) ;
  clrscr() ;
  window(30, 10, 60, 16) ;
  textattr(YELLOW + (MAGENTA << 4) ) ;
  clrscr() ;
  cprintf("Enter radius : ") ;
  cscanf("%f", &r) ;
  a = P * 2 * r ;
  s = P * r * r ;
  cprintf("\n\r---------------------------\r\n") ;
  cprintf("Mohit = %f\r\n", a) ;
  cprintf("Masahat = %f", s) ;
}
```

نکات آموزشی :

- ✓ دستور define ماکرو تعریف می کند. کاربرد دیگر آن در معرفی ثابتها می باشد. این مقادیر در سراسر برنامه معتبرند.

#define	نام یا شناسه	مقدار ثابت

- ✓ تابع window پنجره نمایش را به یک کادر مستطیلی محدود می کند. ابتدا x و y نقطه چپ بالا و سپس x و y نقطه راست پایین.
- ✓ پس از دستور window ، عمل clrscr و تغییر رنگ در محدوده همان پنجره انجام می شود.
- ✓ درون پنجره در تابع cprintf علامت \n برای رفتن به سطر جدید و \r برای رفتن به ابتدای سطر فعلی به کار می رود. همچنین باید از تابع cscanf استفاده نماییم.
- ✓ از این پس برای جلوگیری از تکرار نکته های تکراری، دیگر تابع getch را در پایان برنامه ها نمی نویسیم.

برنامه شماره ۶: این برنامه کاربرد توابع را نشان می دهد. هردو تابع f1 و f2 برای محاسبه مجموع به کار می روند.

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

//-------------------------------
void f1(int a, int b)
{
  printf("Sum= %d\n", a+b) ;
}

//-------------------------------
int f2(int a, int b)
{
  int c;
  c = a + b ;
  return c ;
}

main()
{
  int x, y, z ;

  clrscr() ;
  printf("Enter two numbers : ") ;
  scanf("%d%d", &x, &y) ;
  f1(x, y) ;  //function has not a return value

  x = 10 ;
  y = 5  ;
  z = f2(x, y) ;  //function has a return value
  printf("%d + %d = %d\n", x, y, z) ;
}
```

نکات آموزشی :

- ✓ تابع f1 فقط دو ورودی دارد و مقدار بازگشتی ندارد. بنابراین نوع خروجی آن را در ابتدا void یا بی ارزش معرفی نموده ایم. که البته این امر اختیاری است. در هنگام فراخوانی نیز به صورت یک دستور مستقل به کار می رود همانند زیر برنامه.
- ✓ متغیرهای a و b آرگومانها یا همان پارامترهای تابع می باشند.
- ✓ تابع f2 مقدار بازگشتی دارد. نوع خروجی آن را بسته به نیازمان ازنوع int معرفی نموده ایم. که البته مقدار پیش فرض نیز می باشد. در هنگام فراخوانی نیز نمیتوان به صورت یک دستور مستقل به کار رود بلکه همانند یک عبارت محاسباتی در دستور واگذاری عمل می نماید.
- ✓ در توابعی که مقدار خروجی یا بازگشتی دارند، با استفاده از دستور return این کار را انجام می دهند.
- ✓ گاهی اوقات تابع main نیز مقدار خروجی دارد. چون سایر توابع توسط main فراخوانی می شوند و تابع main نیز توسط سیستم عامل فراخوانی می شود.

برنامه شماره ۷ : این برنامه مفهوم متغیرهای محلی و سراسری را نشان می دهد. همچنین راجع به پارامترهای مقداری بحث می کند.

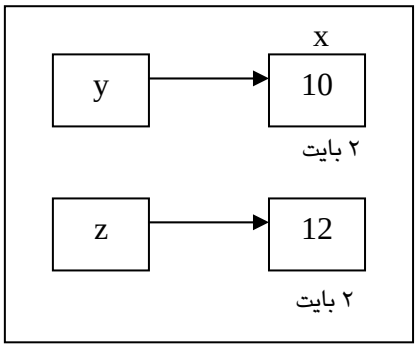
```
#include<stdio.h>
#include<conio.h>

int a , m, y ;  //global variables

//--------------------------------
void f(int a, int b)
{
  int c ;  //local variable
  a += 5 ; //local variable
  b += 3 ; //local variable
  m -= 2 ; //global variable (changed)
}

main()
{
  int x=7 , y=4 ; //local variable

  clrscr() ;
  a = 20 ; //global variable
  m = 10 ; //global variable
  printf("-------- before call function --------\n") ;
  printf("x= %d   y= %d    a= %d    m=%d\n\n", x, y, a, m) ;

  f(x, y) ;
  printf("-------- after call function ---------\n") ;
  printf("x= %d   y= %d    a= %d    m=%d\n\n", x, y, a, m) ;
 }
```

نکات آموزشی :

- ✓ متغیرهای تعریف شده در بلوک توابع حتی تابع main همگی از نوع محلی یا local هستند. بدین معنی که فقط در محدوده همان تابع معتبرند و در سایر بخشها شناخته نمی شوند. و چنانچه بر اساس تشابه اسمی در دو تابع متفاوت دوبار معرفی شوند، هیچ ارتباطی به هم ندارند و تغییرات انجام شده بر روی آنها تأثیری بر دیگر بخشها ندارد. طول عمر متغیر محلی فقط در زمان فراخوانی تابع است.
- ✓ استفاده از توابع و بدنبال آن استفاده از متغیرهای محلی، باعث کاهش مصرف حافظه و در نتیجه بالاتر رفتن سرعت برنامه و مدیریت بهتر برنامه توسط برنامه نویس می شود.
- ✓ آرگومانهای تابع معمولاً به صورت مقداری یا (value parameter) می باشند. یعنی به هنگام فراخوانی، هم مقادیر ثابت را می پذیرند و هم متغیر. در ضمن هرگونه تغییرات انجام شده بر روی آنها درون تابع، تأثیری در متغیرهای محل فراخوانی ندارد. به عبارت دیگر پارامترهای مقداری در حکم متغیرهای محلی می باشند.
- ✓ متغیرهایی که بیرون از تمامی توابع زیر قسمت include معرفی می شوند، از نوع سراسری یا همگانی (global) می باشند و در تمامی قسمتهای برنامه معتبرند. طول عمر آنها تا زمانی است که برنامه به اتمام برسد. هر تغییری در یک بخش مقدار متغیر سراسری را برای دیگر بخشها تغییر می دهد.
- ✓ تا جایی که بتوانیم، باید از متغیرهای سراسری کمتری استفاده کنیم تا حجم برنامه کاهش یابد و مدیریت آن آسانتر شود.
- ✓ چنانچه درون تابعی یک متغیر همنام با متغیر سراسری دوباره تعریف مجدد شود، درون آن تابع حکم متغیر محلی را خواهد داشت.

برنامه شماره ۸ : این برنامه راجع به مفهوم اشاره گرها یا pointer ها بحث می کند.

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
main()
{
  int x=10 ; //variable
  int *y , *z ;  //pointers

  clrscr() ;
  y = &x ; //assign x address to y pointer
  printf("x= %d    x= %d \n\n", x, *y) ;
  z = (int*) malloc(sizeof(int)) ;
  *z = 12 ;
  printf("new data= %d\n", *z) ;
  free(z) ; //destroy data and free memory pointer
}
```



نکات آموزشی :

- ✓ همانطور که در شکل بالا ملاحظه می فرمایید برخی از متغیرها بجای اینکه مستقیماً دارای مقدارداده ای باشند، نشانی خانه ای از حافظه را دارند که بدانها نشانگر یا پوینتر می گویند. اشاره گرها نقش مهمی در برنامه نویسی دارند. یک اشاره گر اینگونه معرفی می شود :

  نوع اشاره گر * نام متغیر , … ;

- ✓ یک اشاره گر به خودی خود حافظه ای ندارد و تهی یا NULL می باشد. و به دوشکل حافظه می گیرد. یکی آنکه با استفاده از عملگر & آدرس متغیر دیگری را در اختیار گیرد. دیگر آنکه توسط دستور malloc مستقیماً حافظه اخذ نماید که این امر نیازمند استفاده از فایل راهنمای stdlib می باشد و شکل کلی آن بدین صورت است :

  نام اشاره گر = (*نوع داده) malloc (سایز مورد نیاز);

- ✓ عملگر sizeof میزان حافظه مصرفی یک نوع داده یا متغیر دلخواه را نشان می دهد.
- ✓ برای آزاد سازی حافظه گرفته شده توسط malloc از دستور free استفاده می نماییم. پس از این عمل، اشاره گر تهی خواهد شد.
- ✓ برای دسترسی به محتوای داده ای، قبل از نام اشاره گر علامت * را می نویسیم.

برنامه شماره ۹ : این برنامه راجع به تفاوت پارامترهای مقداری و پارامترهای مرجع بحث می کند.

```
#include<stdio.h>
#include<conio.h>
//-------------------------------------------------------------
void f(int a, int *b)
{
  a += 5 ;     //value parameter. will not change
  (*b) += 5 ;  //refrence or variable parameter. must be changed
}
//-------------------------------------------------------------
main()
{
  int x=10 , y=10 ;
  clrscr() ;
  printf("---- before call function -----\n") ;
  printf("x= %d     y= %d \n\n", x, y) ;
  f(x , &y);
  printf("---- after call function ------\n") ;
  printf("x= %d     y= %d \n\n", x, y) ;
}
```

نکات آموزشی :

- ✓ اگر پارامترهای ورودی تابع به صورت اشاره گر باشند، از نوع متغیر یا مرجع (refrence parameter) خواهند بود. بنابراین در هنگام فراخوانی حتماً باید یک مقدار متغیر جایگزین نمود و نمی توان از مقدار ثابت استفاده کرد.

- ✓ چنانچه تغییری بر روی پارامترهای مرجع درون توابع انجام شود، این تغییرات بر روی متغیر جایگزین شده در محل فراخوانی تابع نیز اثر می گذارد. یعنی تابع از این طریق می تواند چندین مقدار خروجی داشته باشد.
- ✓ در مثال بالا a یک پارامتر مقداری و b یک پارامتر مرجع می باشد.

برنامه شماره ۱۰ : این برنامه نمره درس ادبیات یک دانش آموز را دریافت داشته سپس نتیجه قبولی یا مردودی او را اعلام می کند.

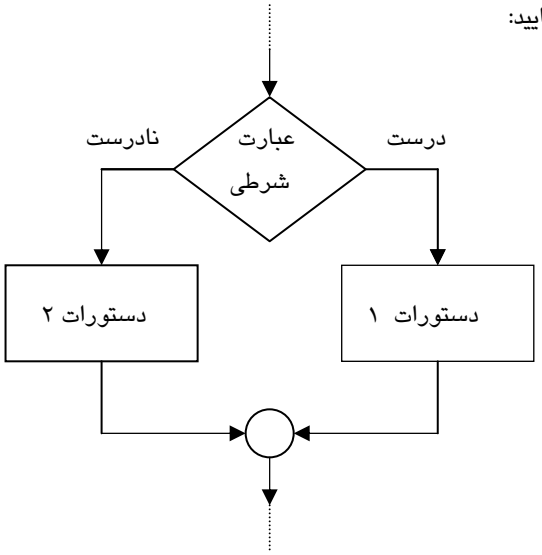
```
#include<stdio.h>
#include<conio.h>
main()
{
  float x ;
  clrscr() ;
  printf("Enter number (0..20)  :  ") ;
  scanf("%f", &x) ;
  if (x>=10) printf("Passed\n") ;
  else printf("Failed\n") ;
}
```

نکات آموزشی :

- ✓ از دستور شرطی if برای مقایسه نمره دانش آموز استفاده شده است. توجه کنید که بر خلاف زبانهای دیگر اینجا کلمه then وجود ندارد. شکل کلی دستور بدین گونه است :

```
if ( عبارت شرطی )  دستو ۱  ; else  دستور ۲  ;
```

- ✓ ابتدا عبارت بررسی می شود اگر حاصل آن درست بود یا حتی مقداری مخالف صفر بود، به معنی true است و دستور ۱ اجرا می شود و دستور ۲ اجرا نخواهد شد. و چنانچه حاصل عبارت نادرست و یا مقدار صفر بود، به معنی false است و عکس قضیه فوق اتفاق خواهد افتاد.
- ✓ عملگرهای مقایسه ای زبان C مشابه سایر زبانهاست. فقط دو تفاوت دارد : یکی آنکه برای نامساوی، علامت =! را می نویسیم و دیگر آنکه برای تساوی دو چیز، از علامت == استفاده می کنیم.
- ✓ چنانچه هر یک از دستورات ۱ یا ۲ هر کدام بیش از یک دستور باشند، از بلوک استفاده می نماییم.
- ✓ در شکل زیر نمودار کلی عملکرد دستور if را ملاحظه می فرمایید:



برنامه شماره ۱۱ : این برنامه نمره درس ادبیات یک دانش آموز را دریافت داشته سپس کیفیت تحصیلی او را بر اساس نمرات خیلی بد، بد، خوب و خیلی خوب درجه بندی می کند. در ضمن نمرات خارج از محدوده را بعنوان خطا گزارش می نماید.

```
#include<stdio.h>
#include<conio.h>
main()
{
  float x ;
  clrscr() ;
```

```
  printf("Enter number (0..20)  :  ") ;
  scanf("%f", &x) ;

  if ((x<0) || (x>20))
    printf("Data error") ;
  else if (x<5)
         printf("Very bad") ;
       else if (x<10)
              printf("Bad") ;
            else if (x<15)
                   printf("Good") ;
                 else
                   printf("Very Good") ;
}
```

نکات آموزشی :

- ✓ در این برنامه از مفهوم دستورات شرطی تودرتو استفاده شده است.
- ✓ عملگر || شرطهای منطقی را باهم or می نماید. همچنین عملگر های && و ~ به ترتیب برای and و not به کار می روند.

برنامه شماره ۱۲ : این برنامه اندازه سه ضلع یک مثلث فرضی را دریافت داشته سپس محیط و مساحت آن رادر صورت وجود نشان می دهد. در صورت عدم وجود مثلث، پیغام خطا گزارش می نماید. می دانیم که در هر مثلث، مجموع هر دو ضلع از ضلع سوم آن بزرگتر می باشد.

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
main()
{
  float a, b, c;
  float p, m, s;

  clrscr() ;
  printf("Enter 3 sides of a triangle :  ") ;
  scanf("%f%f%f", &a, &b, &c) ;
  if ((a+b>c) && (a+c>b) && (b+c>a)) {
    m = a+b+c ;
    p = m / 2 ;
    s = sqrt(p* (p-a)* (p-b)* (p-c)) ;
    printf("Mohit = %f\n", m) ;
    printf("Masahat = %f\n", s) ;
  }
  else
    printf("\aThis is not a valid triangle!\n") ;
}
```

نکات آموزشی :

- ✓ تابع sqrt برای محاسبه ریشه دوم یا جذر یک عدد می باشد و نیاز به فایل راهنمای math.h دارد. در صورت منفی بودن عدد، محاسبه ریشه دوم آن ممکن نیست و برنامه با خطا متوقف می شود.
- ✓ علامت \a در دستور printf موجب نواختن بوق beep کامپیوتر می شود.

برنامه شماره ۱۳ : این برنامه ضرایب معادله درجه دوم را دریافت داشته سپس ریشه های آن را در صورت وجود محاسبه می نماید.

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
main()
{
  float a, b, c;
```

```
  float x1, x2, d;
  clrscr() ;
  printf("  AX^2 + BX + C = 0\n") ;
  printf("------------------------\n") ;
  printf("Enter 3 nubmers :  ") ;
  scanf("%f%f%f", &a, &b, &c) ;
  d = pow(b, 2) - 4 * a * c ; //delta=b^2-4ac

  if (d<0)
    printf("No root!") ;
  else if (d==0) {
         printf("There is one double root\n") ;
         x1 = -b / (2*a) ;
         printf("X= %f", x1) ;
       }
       else  {
         printf("There are two roots\n") ;
         x1 = (-b + sqrt(d)) / (2*a) ;
         x2 = (-b - sqrt(d)) / (2*a) ;
         printf("X1= %f    X2= %f", x1, x2) ;
       }
}
```

نکات آموزشی :

- ✓ تابع pow عدد اول را به توان عدد دوم می رساند.
- ✓ همچنین توابع int برای محاسبه جزء صحیح، abs برای قدرمطلق، sin برای سینوس، log برای لگاریتم طبیعی در پایه e، log10 برای لگاریتم در پایه ۱۰، cos برای کسینوس، tan برای تانژانت، asin برای آرک سینوس، acos برای آرک کسینوس و atan برای آرک تانژانت به کار می رود.
- ✓ تابع کتانژانت وجود ندارد. می توانید از معکوس تانژانت استفاده نمایید.
- ✓ مقدار دقیق تر عدد π از این طریق به دست می آید : `p = 4 * atan(1) ;`
- ✓ توجه کنید که در توابع مثلثاتی اندازه زاویه برحسب رادیان می باشد. پس ابتدا درجه را با این فرمول بر حسب رادیان تبدیل نمایید :

  `r = (d * atan(1)) / 45 ;`
- ✓ زبان سی توابع قدرتمند دیگری برای محاسبه سینوس و کسینوس و تانژانت هیپربولیک دارد که برای مطالعه بیشتر به کتب مرجع و کاملتر مراجعه نمایید.
- ✓ همیشه تقدم یا اولویت انجام عملگرها را به خاطر داشته باشید و برای جلوگیری از اشتباهات، ابتدا آن را بر روی کاغذ نوشته و تست نمایید. جدول کلی اولویت عملگرها در پایان این جزوه آمده است.

برنامه شماره ۱۴ : این برنامه اشکال گوناگون حلقه for را به کار می گیرد.

```
#include<stdio.h>
#include<conio.h>
main()
{
  int a, b, c, d;

  clrscr() ;
  printf("-------------------------------------------\n") ;
  for (a=1 ; a<=10 ; a++)
    printf("%4d", a) ;

  printf("\n-------------------------------------------\n") ;
  b = 10 ;
  for ( ; b>=1 ; ) {
    printf("%4d", b) ;
    b-- ;
  }
```

```
  printf("\n-------------------------------------------\n") ;
  c = 5 ;
  for ( ; ; ) {
    printf("%4d", c) ;
    c += 5 ;
    if (c>50) break ;
  }

  printf("\n-------------------------------------------\n") ;
  for (d=1 ; d<=10 ; d++) {
    textcolor(GREEN) ;
    cprintf("Iran\r\n") ;  //print 10 times
    if (d % 2 == 0) continue ;
    textcolor(RED) ;
    cprintf("Borujerd\r\n", d) ;  //print 5 times
  }
}
```

نکات آموزشی :

- ✓ حلقه a اعداد ۱ تا ۱۰ را نمایش می دهد و همچنین حلقه b اعداد ۱۰ تا ۱ را نمایش می دهد.
- ✓ حلقه c مضربهای عدد ۵ را نشان می دهد (از ۱ تا ۵۰). در این جا با استفاده از دستور break ، حلقه کاملاً متوقف می شود.
- ✓ حلقه d کلمه iran را ۱۰ بار می نویسد. اما کلمه borujerd را ۵ بار می نویسد. چون توسط دستور continue ، وقتی که شمارنده حلقه به مضربهای ۲ می رسد، از ادامه دستورات بعدی صرفنظر نموده و دوباره به اولین دستور داخل بلوک حلقه، باز می گردد.
- ✓ شکل کلی دستور حلقه سازی for بدین صورت می باشد:

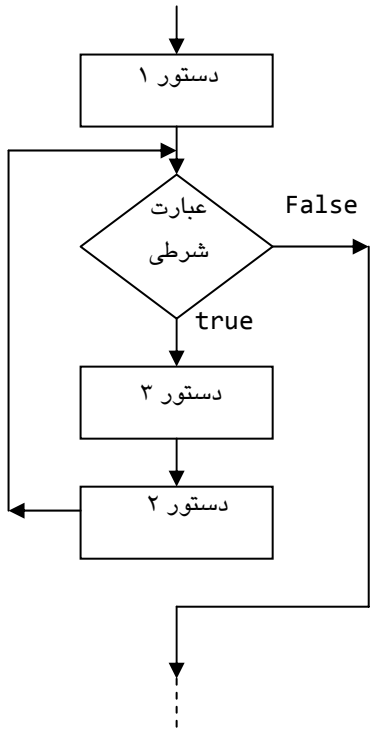
for (دستور ۱ ; عبارت شرطی ; دستور ۲) ; دستور ۳

معمولاً دستور ۱ برای مقداردهی اولیه به شمارنده حلقه به کار می رود. دستور ۲ برای کاهش یا افزایش شمارنده است. تا وقتی که عبارت شرطی برقرار است، حلقه ادامه می یابد. دستور ۳ نیز دستورات داخل حلقه یا تنه حلقه می باشد.

چون دستور ۱ فقط یکبار اجرا می شود، پس می توان آن را به قبل از شروع حلقه منتقل نمود.

بررسی عبارت شرطی و انجام دستور ۲ را می توان به درون بلوک دستور ۳ اضافه نمود.

- ✓ حلقه for لزوماً یک حلقه شمارشی نیست و می تواند از نوع نامعین و شرطی باشد.
- ✓ در حلقه a در دستور printf الگوی %4d بیانگر آن است که یک عدد صحیح را در فضای ٤ کاراکتری نمایش دهد. اگر ارقام عدد کمتر از ٤ حرف باشد، به تعداد لازم از سمت چپ حرف فاصله درج می شود. ولی اگر عدد ٤ رقمی یا بزرگتر باشد، به طور کامل چاپ می شود.
- ✓ فلوچارت کلی حلقه for بدین شکل است.



برنامه شماره ۱۵ : این برنامه با استفاده از دو حلقه for تودرتو، جدول ضرب ۱۰ در ۱۰ را به صورت منظم نمایش می دهد.

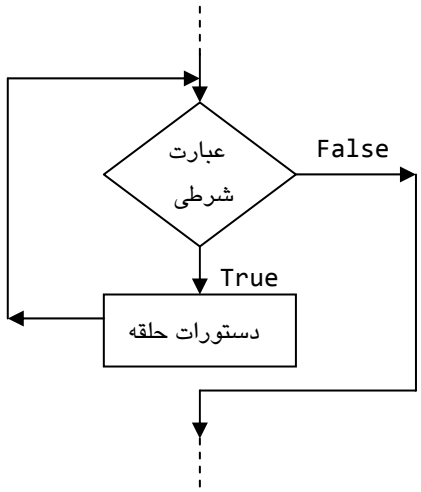
```
#include<stdio.h>
#include<conio.h>
main()
{
  int a, b;
  clrscr() ;
  for (a=1 ; a<=10 ; a++) {
    for (b=1 ; b<=10 ; b++)
      printf("%4d", a*b) ; //print one row

    printf("\n") ; //go to next line
  }
}
```

نکات آموزشی :

- ✓ حلقه های متداخل به گونه ای هستند که یکی کاملاً درون دیگری قرار گرفته است و هیچ یک نباید دیگری را قطع کند.
- ✓ حلقه ای که زودتر آغاز شده، دیرتر از بقیه پایان می یابد و بالعکس.
- ✓ تعداد دفعات کل تکرار حلقه ها ی متداخل، مساوی است با حاصل ضرب تعداد دفعات هر کدام از حلقه ها در یکدیگر.

برنامه شماره ۱۶ : این برنامه نحوه استفاده از حلقه های while و do while را نشان می دهد.

```
#include<stdio.h>
#include<conio.h>
#define n 10
main()
{
  int x, y ;

  clrscr() ;
  x = 10 ;
  while (x<=98) {
    printf("%10d", x) ;
    x += 2 ;
  }

  printf("\n\n") ;
  y = 3 ;
  do {
    printf("%10d", y) ;
    y += 3 ;
  }  while (y<100) ;
}
```



نکات آموزشی :

- ✓ شکل بالا مربوط به فلوچارت حلقه while است. در این حلقه ابتدا شرط بررسی می شود و سپس درصورت درست بودن آن، دستورات حلقه تکرار می شوند. تفاوت ان با حلقه do while این است که این حلقه ابتدا یکبار دستورات را اجرا می کند و سپس عبارت شرطی را بررسی می کند. شکل کلی این دو دستور بدین شرح می باشد:

while (عبارت شرطی) دستور یا بلوک ;

do دستور یا بلوک (عبارت شرطی) ;

- ✓ در حلقه x اعداد زوج دو رقمی نمایش داده می شود. حلقه y نیز مضربهای یک رقمی و دو رقمی عدد ۳ را نمایش می دهد.

برنامه شماره ۱۷ : این برنامه ۱۰ عدد دلخواه را دریافت داشته سپس مجموع و میانگین آنها را نشان می دهد.

```
#include<stdio.h>
#include<conio.h>
#define n 10
main()
{
  int i ;
  float sum=0, avg, x;
  clrscr() ;
  for (i=1 ; i<=n ; i++) {
    printf("Enter number : ") ;
    scanf("%f", &x) ;
    sum += x ;
  }
  avg = sum / n ;
  printf("----------------------\n");
  printf("Sum = %f\n", sum) ;
  printf("Average = %f\n", avg) ;
}
```

برنامه شماره ۱۸ : این برنامه تعدادی عدد مثبت را دریافت داشته سپس مجموع و میانگین آنها را نشان می دهد.

```
#include<stdio.h>
#include<conio.h>
main()
{
  int i, n=0 ;
  float sum=0, avg, x;
  clrscr() ;
  while (1) {
    printf("Enter number. Negative to quit:  ") ;
    scanf("%f", &x) ;
    if (x<0) break ;
    n++ ;
    sum += x ;
  }
  if (n>0)  avg= sum / n ; else avg=0 ;
  printf("----------------------\n");
  printf("Count = %d\n", n) ;
  printf("Sum = %f\n", sum) ;
  printf("Average = %f\n", avg) ;
}
```

نکات آموزشی :

- ✓ تعداد اعداد در ابتدا مشخص نیست. با ورود عدد منفی، برنامه از حلقه خارج می شود.
- ✓ اگر کاربر در ابتدا عددی منفی وارد نماید، آنگاه تعداد اعداد ورودی یعنی n صفر خواهد بود. در این صورت برای محاسبه میانگین، مشکل تقسیم بر صفر و پیغام خطای division by zero را خواهیم داشت که این امر به کمک یک دستور if برطرف شده است.

برنامه شماره ۱۹ : این برنامه با استفاده از تابع، بزرگترین مقسوم علیه مشترک و کوچکترین مضرب مشترک دو عدد را نشان می دهد.

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

long int bmm(long int a , long int b)
{
  long int c ;
  while (b!=0) {
    c = b ;
    b = a % b ;
    a = c ;
  }

  return a ;
}
```

```
main()
{
  long int a, b, c, d;
  clrscr() ;
  printf("Enter 2 numbers : ") ;
  scanf("%ld%ld", &a, &b) ;
  c = bmm(a, b) ;
  d = (a * b) / c ;
  printf("------------------------\n") ;
  printf("BMM= %ld\n", c) ;
  printf("KMM= %ld\n", d) ;
}
```

نکات آموزشی :

- ✓ نوع داده long int براع اعداد صحیح بزرگ است و ٤ بایت حافظه مصرف می کند. الگوی چاپ و پیمایش آن %ld می باشد.

برنامه شماره ۲۰ : این برنامه با استفاده از چند تابع، اول بودن یک عدد را بررسی می کند و یا آنرا به عوامل اول تجزیه می نماید.

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
//-----------------------------------
void f1(long int a)
{
  long int b=1 ;
  if (a==0) return ;
  printf("%10ld", b) ;
  if (a==1) return;
  for (b=2 ; b<=(a/2) ; b++)
    if (a % b ==0) printf("%10ld", b) ;
  printf("%10ld\n", a) ;
}
//-----------------------------------
void f2(long int a)
{
  long int b=2 ;
  while (a>1) {
    if (a % b ==0) {
      printf("%10ld", b) ;
      a /= b ;
    }
    else b++ ;
  }
  printf("\n") ;
}
//-----------------------------------
void f3(long int a)
{
  long int b=2 , p=0;
  while (a>=1) {
    if ((a==1) && (p==0)) break;
    if (a % b ==0) {
      p++ ;
      a /= b ;
    }
    else if (p!=0) {
         printf("%ld ^ %ld\n", b, p) ;
         p = 0 ;
        }
       else  b++ ;
  }
}
```

نکات آموزشی :

- ✓ تابع f1 تمامی مقسوم علیه های عدد دریافتی را بر روی مانیتور نمایش می دهد. مثلاً اگر ورودی عدد 12 باشد، آنگاه خروجی عبارت است از :

  1 2 3 4 6 12

- ✓ تابع f2 عدد دریافتی را به عوامل اول تجزیه نموده و آنها را کنار هم می نویسد. مثلاً اگر ورودی عدد 120 باشد، آنگاه خروجی عبارت است از :

  2 2 2 3 5

- ✓ تابع f3 عدد دریافتی را به عوامل اول تجزیه نموده و آنها را کنار هم می نویسد. مثلاً اگر ورودی عدد 600 باشد، آنگاه خروجی عبارت است از :

  2 ^ 3
  3 ^ 1
  5 ^ 2

```
int primer(long int a)
{
  long int flag ;
  long int b ;
  if ((a==1) || (a==0))
    flag = 0  ; // a is a primer number
  else
    flag = 1  ; //default: a is a primer number

  for (b=2 ; b<=sqrt(a) ; b++)
    if (a % b == 0) {
      flag = 0 ;  // a is not a primer number
      break ;
    }
  return flag ;
}
```

- ✓ تابع primer بررسی می کند که آیا عدد دریافتی از نوع اعداد اول است یا خیر. در اینجا از یک متغیر کنترل کننده به نام flag یا پرچم استفاده کرده ایم. فرض اولیه بر آن است که عدد ورودی اول است. اگر در طی حلقه تکرار به موردی برخوردیم که به جز عدد یک و خود عدد، مقسوم علیه دیگری پیدا شد، آنگاه ضمن خروج از حلقه، فرض اولیه را نقض نموده و عدد 0 را در پرچم قرار می دهیم. این یک مقدار بازگشتی برای تابع است. پس از فراخوانی با توجه به مقدار برگشتی از تابع، اول بودن با نبودن عدد مورد نظر معلوم می گردد.

```
//-----------------------------------
main()
{
  long int x;
  clrscr() ;
  printf("Enter  number : ") ;
  scanf("%ld", &x) ;

  if (primer(x)==0)
    printf("This is not a primer number\n") ;
  else
    printf("This is a primer number\n") ;

  printf("\nAll divisor...\n") ;
  f1(x) ;

  printf("\nExplode...\n") ;
  f2(x) ;

  printf("\nExplode...\n") ;
  f3(x) ;
}
```

برنامه شماره ۲۱: با استفاده از تابع primer ، اعداد اول کوچکتر یا مساوی 50000 را نمایش داده در پایان تعداشان را می نویسد.

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

//-----------------------------------
int primer(long int a)
{
  long int flag=1 ;  // a is a primer number
  long int b ;

  for (b=2 ; b<=sqrt(a) ; b++)
    if (a % b == 0) {
      flag = 0 ;  // a is not a primer number
      break ;
    }
  return flag ;
}
```

```
main()
{
  long int x;
  long int count=0 ;

  clrscr() ;
  for (x=2 ; x<=50000 ; x++)
    if (primer(x)!=0) {
      count++ ;
      printf("%10ld" , x) ;
      if (count % 192 ==0) getch() ;
    }
  printf("\n\n%ld primer numbers are existed", count ) ;
}
```

نکات آموزشی :

- ✓ چون تعداد خروجی های این برنامه بسیار زیاد است و از یک صفحه بیشتر می شود، با استفاده از تابع getch تأخیری به برنامه داده ایم تا کاربر بتواند اطلاعات را مطالعه نموده و پس از زدن یک کلید، ادامه آنها را ببیند.

برنامه شماره ۲۲ : با استفاده از دوتابع مجزا، اول بودن یک عدد بسیار بزرگ را بررسی نموده و زمان اجرای الگوریتمها را مقایسه می کند.

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
#include<time.h>
//-----------------------------------------------------
int f1(long int a) //low performance and slow algorithm
{
  long int flag=1 ;  // a is a primer number
  long int b ;
  for (b=2 ; b<=(a/2) ; b++)
    if (a % b == 0) {
      flag = 0 ;  // a is not a primer number
      break ;
    }
  return flag ;
}

//-----------------------------------------------
int f2(long int a) //optimized and fast algorithm
{
  long int flag=1 ;  // a is a primer number
  long int b ;
  for (b=2 ; b<=int(sqrt(a)) ; b++)
    if (a % b == 0) {
      flag = 0 ;  // a is not a primer number
      break ;
    }
  return flag ;
}

//-----------------------------------------------
main()
{
  long int x=24999983 ;  // 24,999,983 is a big primer number
  time_t t1, t2 ;
  clrscr() ;
  t1 = time(NULL) ;   //start timer (current time)
  if (f1(x)!=0)
     printf("%10ld is primer." , x) ;
  else
      printf("%10ld is not primer." , x) ;
  t2 = time(NULL) ;   //stop timer (next time)
  printf("\n") ;
  printf("Execute time= %d seconds.\n", t2-t1 ) ;
```

نکات آموزشی :

- ✓ در تابع f1 الگوریتم ضعیفی داریم. چون برای بررسی اول بودن یک عدد اول، عوامل بخش پذیر را تا نصف آن عدد چک می کند و این کار زمان بر است. مثلاً برای عدد ۹۷ باید بخش پذیری آن را بر اعداد ۲ تا ٤٨ را بررسی کنیم.
- ✓ در تابع f2 الگوریتم بهینه ای داریم. چون برای بررسی اول بودن یک عدد اول، عوامل بخش پذیر را تا جذر آن عدد چک می کند و این کار سریعتر انجام می شود. مثلاً برای عدد ۹۷ باید بخش پذیری آن را بر اعداد ۲ تا ۹ را بررسی کنیم.
- ✓ برای محاسبه زمان از تابع زمان سنج یا کرنومتری بنام time استفاده می کنیم. پارامترNULL به معنی تهی است. قبل از انجام عملیات زمان را بر حسب ثانیه درt1 ذخیره می کنیم و این عمل را پس از انجام دستورات موردنظر در t2 انجام می دهیم. تفاضل این دومقدار، زمان مصرفی را نشان می دهد.

```
  printf("-------------------------------\n");
  t1 = time(NULL) ;   //start timer(current time)
  if (f2(x)!=0)
     printf("%10ld is primer." , x) ;
  else
      printf("%10ld is not primer." , x) ;
  t2 = time(NULL) ;   //stop timer (next time)
  printf("\n") ;
  printf("Execute time= %d seconds.\n", t2-t1 ) ;
}
```

✓ استفاده از تابع time نیازمند افزودن فایل راهنمای time.h به برنامه است. تنها عیب این روش آن است که دقت زمان در حدثانیه است و مقدارهای کمتر یا کسرهای ثانیه ای را محاسبه نمی کند. روش فوق برای زمان سنجی در عملیات محاسباتی خیلی سنگین و یا کامپیوترهای کند و قدیمی، مناسب می باشد.

برنامه شماره ۲۳: حد مجموع ۱۰ جمله اول از سری زیر را محاسبه می کند: $s = \sqrt{7 + \sqrt{7 + \sqrt{7 + \ldots}}}$

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
#define n 10
#define x 7
main()
{
  int i ;  double s=0 ;
  clrscr() ;
  for(i=1 ; i<=n ; i++) {
    s = sqrt(x + s) ;
    printf("%lf\n", s) ;
  }
}
```

برنامه شماره ۲۴: حد مجموع ۱۰ جمله اول از سری زیر را محاسبه می کند: $s = \sqrt{1 + \sqrt{2 + \sqrt{3 + \ldots}}}$

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
#define n 10
main()
{
  int i ; double s=0 ;
  clrscr() ;
  for(i=n ; i>=1 ; i--)
    s = sqrt(i + s) ;

  printf("%lf\n", s) ;
}
```

نکات آموزشی :

✓ در اینجا مجبوریم از شمارنده معکوس استفاده کنیم. در غیر اینصورت اعداد داخل رادیکال معکوس خواهند شد.

برنامه شماره ۲۵: مجموع ۱۰ جمله اول از سری زیر را محاسبه می کند: $s = \frac{1}{2} - \frac{2}{3} + \frac{3}{4} - \ldots$

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
#define n 20
main()
{
  int i ;    double s=0 ; //sum
  int k=1 ; //sign of each scentence
  clrscr() ;
  for(i=1 ; i<=n ; i++) {
    s = s + k * (float(i) /float(i+1)) ;
    k = k * -1 ;
    printf("%lf\n", s) ;
  }
}
```

نکات آموزشی :

✓ در اینجا برای تعویض علامت جملات به صورت یک در میان مثبت و منفی، از ضریب k استفاده می کنیم تا به طور متوالی در ۱- ضرب شود.

برنامه شماره ۲۶ : ۸۰ جمله اول از سری فیبوناچی را نشان می دهد. در سری فیبوناچی اولین و دومین عدد، یک می باشند و بقیه اعداد از جمع دو جمله ماقبل خود به دست می آیند : ... و ۱۳ و ۸ و ۵ و ۳ و ۲ و ۱ و ۱

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
#define n 80
main()
{
  int i ;
  double a=1 , b=1 ; //first & second number in fibonatchi series
  double c;  //temporary for exchange
  clrscr() ;

  printf("Fibonatchi series numbers....\n\n") ;
  for(i=1 ; i<=n ; i++) {
    printf("#%2d=%19.0lf\n", i, a) ;
    c = b;
    b = a + b ;
    a = c ;
    if (i % 24 ==0) getch() ;
  }
}
```

برنامه شماره ۲۷ : یک منو شامل چهار گزینه نشان می دهد و با توجه به انتخاب کاربر، عباراتی را بر روی مانیتور نشان می دهد.

```
#include<conio.h>
#include<stdio.h>
main()
{
  char x ;

  while (1) {
    clrscr() ;
    printf("0- Exit\n") ;
    printf("1- First\n") ;
    printf("2- Second\n") ;
    printf("3- 3th\n") ;
    printf("-------------------\n") ;
    printf("Enter selection : ") ;
    x = getche() ;
    printf("\n");

   switch (x) {
      case '0' :
         printf("Good-Bye\n") ;
         break ;

      case '1' :
         printf("Iran\n") ;
         break ;

      case '2' :
         printf("Borujerd\n") ;
         break ;

      case '3' :
         printf("Gorgan\n") ;
         break ;
```

نکات آموزشی :

✓ شکل کلی دستور چند انتخابی بدین شرح می باشد :

```
switch (عبارت) {
  case مقدار ۱ :
    دستورات ;
    break ;

  case مقدار ۲ :
    دستورات ;
    break ;
    .
    .
    .
  case مقدار آخر :
    دستورات ;
    break ;

  default :
    دستورات ;
}
```

```
      default :
        printf("Bad selection. Try again!\n") ;
    }

    if (x=='0') break; //exit loop
    printf("\nPress any key to continue...") ;
    getch() ;
  }
}
```

- ✓ متغیر x از نوع کاراکتری و تک حرفی معرفی شده است.
- ✓ در حلقه while عبارت شرطی (1) همیشه درست می باشد. پس این حلقه مدام تکرار می شود تا به شرط خروج خود یعنی عدد 0 برسد.
- ✓ با استفاده از دستور چند انتخابی switch مقادیر مختلف متغیر x بررسی شده است. اگر x مساوی با هر یک از مقادیر مورد نظر باشد، عملی انجام می شود و گرنه گزینه default اجرا خواهد شد.
- ✓ وجود دستور break در هر گزینه ضروری است وگرنه بقیه دستورات در گزینه های زیرین اجرا خواهند شد.
- ✓ تابع getch می تواند خروجی کاراکتری نیز داشته باشد. در این تابع حرف دریافتی از صفحه کلید، دیده نمی شود و نیازی هم به enter ندارد.
- ✓ در تابع getche حرف دریافتی از صفحه کلید، دیده می شود و نیازی به enter ندارد.
- ✓ در تابع getchar حرف دریافتی از صفحه کلید، دیده شده و همچنین نیازمند ی کلید enter می باشد.
- ✓ یادآور می شویم که داده های کاراکتری در نقل قول تکی قرار می گیرند اما مقادیر رشته ای و چند حرفی در نقل قول جفت قرار می گیرند.

برنامه شماره ۲۸ : این برنامه ۱۰ عدد را خوانده و در آرایه قرار می دهد. سپس آنها را به عکس ترتیب خوانده شدن، نشان می دهد.

```
#include<stdio.h>
#include<conio.h>
#define n 10

main()
{
  int i ;
  int x[n] ;

clrscr() ;

  for(i=0 ; i<=n-1 ; i++) {
    printf("Enter number: ") ;
    scanf("%d", &x[i]) ;
  }

  printf("--- Reversed ---\n") ;
  for(i=n-1 ; i>=0 ; i--)
    printf("%d\n", x[i]) ;
}
```

نکات آموزشی :

- ✓ در معرفی آرایه فقط تعداد اعضاء را می نویسیم. در این زبان اولین اندیس آرایه حتماً از صفر شروع می شود. پس اگر x یک آرایه ۱۰ تایی باشد، همیشه از خانه x[0] شروع شده و به x[9] ختم می شود.
- ✓ اندیس آرایه همیشه از نوع عددی می باشد. برای دسترسی به عناصر آرایه باید شماره خانه یا همان اندیس را بنویسیم.
- ✓ فضای اشغالی آرایه در حافظه مساوی است با حاصل ضرب تعداد خانه های آرایه در سایز نوع داده آرایه. مثلاً در اینجا x ده خانه دارد و هر خانه یک عدد دوبایتی دارد پس در مجموع ۲۰ بایت اشغال می کند.
- ✓ عناصر درون آرایه همیشه از یک نوع هستند. مثلاً همه از نوع کاراکتری یا صحیح یا اعشاری و ... می باشند.
- ✓ نام آرایه به تنهایی اشاره گری است که به محل اولین بایت شروع آن در حافظه اشاره می کند. بنابر این می تواند به عنوان پارامتر مرجع یا متغیر درون توابع به کار گرفته شود.
- ✓ گاهی اوقات آرایه تعداد مشخصی ندارد مثلاً int x[]; . پس به هر تعداد می توان به هر تعداد می توان آن را به کار گرفت ولی باید مراقب حدود آن بود تا نتایج نادرست به بار نیاید.

برنامه شماره ۲۹ : این برنامه ۱۰ عدد را مستقیماً در آرایه قرار می دهد. سپس آنها را نشان می دهد.

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
main()
{
  int i ;
  //10 numbers in list
  int x[]= {12, 15, 3, 19, 8, 6, 0, 5, 18, 11} ;

  clrscr() ;
  for(i=0 ; i<=9 ; i++) {
    printf("%d\n", x[i]) ;
  }
}
```

آرایه x

12	15	3	19	8	6	0	5	18	11
۰	۱	۲	۳	۵	٤	٦	۷	۸	۹

نکات آموزشی :

- ✓ در اینجا مستقیماً مقادیر آرایه را در هنگام معرفی آن مشخص نموده ایم. بنابر این نیازی به ورودی نیست. از این روش بیشتر برای تست الگوریتمها استفاده می شود. چون داده ورودی از صفحه کلید نداریم، بنابراین بررسی نتایج برنامه سریعتر خواهد بود.

برنامه شماره ۳۰ : این برنامه نحوه کار با متغیرهای رشته ای و برخی توابع را نشان می دهد.

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

main()
{
  char *a, b[30], c[30] ;

  clrscr() ;
  textcolor(WHITE) ;
  strcpy(a,"In the ") ;        // a= "In the"
  strcat(a, "name of god.") ;  // a= "In the name of god"
  printf("%s\n", a) ;
  printf("Enter name and family : ") ;
  gets(b) ;  //read full string
  textcolor(GREEN) ;
  cprintf("\nWellcom %s\n", b) ;

  printf("\nPlease re-enter name and family : ") ;
  scanf("%s", b) ;  //read first word in string
  textcolor(RED) ;
  cprintf("\nExcuse me %s\n", b) ;
}
```

نکات آموزشی :

- ✓ متغیر رشته ای را به دو صورت می توان معرفی نمود : یکی به صورت اشاره گر کاراکتری که طول این رشته ها متغیر است و دیگری به صورت آرایه کاراکتری با طول ثابت.
- ✓ متغیر رشته ای را نمی توان با دستور واگذاری = مقداردهی کرد. بلکه با استفاده از تابع strcpy این کار انجام می شود.
- ✓ تابع strcat چند رشته را با هم ترکیب کرده و به اولین آرگومان رشته ای خود می چسباند.
- ✓ برای چاپ و یا پیمایش مقادیر رشته ای از الگوی %s استفاده می شود. در دستور scanf نیز نباید علامت & را بنویسیم.
- ✓ تابع gets برای دریافت رشته ها به کار می رود. در این تابع حرف فاصله جزو رشته محسوب می شود و رشته ورودی را بطور کامل می خواند. اما در تابع scanf حرف فاصله بعنوان جداکننده داده های ورودی است و فقط اولین کلمه را می خواند.
- ✓ برای استفاده از توابع رشته ای از فایل راهنمای string.h استفاده کنید.

برنامه شماره ۳۱ : این برنامه نحوه کار با متغیرهای رشته ای و برخی توابع را نشان می دهد.

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<ctype.h>
main()
{
  char *a, *b;
  int i;

  clrscr() ;
  strcpy(a,"aLirEza") ;
  strcpy(b,"ALIreza") ;
  if (strcmp(a,b)==0) printf("IRAN\n") ;
  else printf("BORUJERD\n") ;  //this is done because (a<>b)

  //convert a and b to upper case
  for (i=0; i<=strlen(a)-1 ; i++) {
    a[i] = toupper(a[i]) ;
    b[i] = toupper(b[i]) ;
  }

  if (strcmp(a,b)==0) printf("IRAN\n") ;  //this is done because (a=b)
  else printf("BORUJERD\n") ;
}
```

نکات آموزشی :

- ✓ برای مقایسه دو رشته نمی توان از عملگرهای == و شبیه آن استفاده نمود بلکه از تابع strcmp استفاده می نماییم. نتیجه این تابع یک عدد است که اگر منفی باشد یعنی رشته اول از دومی کوچکتر است. اگر مثبت باشد یعنی رشته اول از دوم بزرگتر است و اگر صفر باشد یعنی دو رشته با هم مساویند.
- ✓ در مقایسه رشته ها حروف بزرگ و کوچک انگلیسی تفاوت دارند.
- ✓ بزرگتر یا کوچکتر بودن ربطی به طول رشته ندارد بلکه به ترتیب الفبایی همانند لغات دیکشنری بستگی دارد. توجه کنید که حروف الفبایی بزرگ انگلیسی دارای کد اسکی کوچکتری نسبت به حروف کوچک می باشند.
- ✓ تابع touopoer یک حرف را به حروف بزرگ تبدیل می کند و تابع tolower یک حرف را به حروف کوچک تبدیل می کند. این دو تابع در فایل راهنمای ctype.h موجود می باشند.
- ✓ تابع strlen طول رشته یا همان تعداد کاراکترها را حساب می کند. در زبان سی بر خلاف زبان پاسکال طول رشته در خود رشته نگهداری نمی شود. بلکه در آخرین کاراکتر رشته، علامت \0 یا تهی قرار می گیرد.

برنامه شماره ۳۲ : این برنامه مربوط به بررسی طول رشته و چگونگی پردازش آن می باشد.

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
main()
{
  char *a ;
  int i;

  clrscr() ;
  strcpy(a,"I want to learn C programming language.") ;
  printf("%s\n", a) ;
  printf("%d characters.\n\n", strlen(a)) ;
  //set string to 17 characters
  a[17] = '\0' ; //NULL character
  printf("%s\n", a) ;
  printf("%d characters.\n\n", strlen(a)) ;
```

```
  //restore string to last status
  a[17] = ' ' ;
  printf("%s\n", a) ;
  printf("%d characters.\n", strlen(a)) ;
}
```

I		w	a	n	t		t	o		l	e	a	r	n		C		p	r	o	g	r	a	m	m	i	n	g		l	a	n	g	u	a	g	e	.	\0
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39		

نکات آموزشی :

- ✓ در اینجا a یک رشته ۳۹ حرفی است که از حرف شماره ۰ تا ۳۸ را دارد. ما به عمد حرف شماره ۱۷ را برابر \0 یا تهی قرار می دهیم. پس به هنگام چاپ فقط ۱۷ حرف اول آن دیده می شود : I want to learn C
- ✓ سپس با قراردادن فضای خالی در حرف شماره ۱۷ دوباره به حالت قبلی باز می گردیم.
- ✓ قراردادن یک کاراکتر در محل مورد نظر با دستور = امکان پذیر است و باید از علامت نقل قول تکی استفاده شود.

برنامه شماره ۳۳ : این برنامه رابطه بین آرایه و تابع را نشان می دهد.

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

//------------------ -----------------
void get_list(int list[] , int count)
{
  int i;
  for(i=0 ; i<=count-1 ; i++) {
    printf("Enter number : ") ;
    scanf("%d", &list[i]) ;
  }
}

//-------------------------------------
void show_list(int list[] , int count)
{
  int i;
  for(i=0 ; i<=count-1 ; i++)
    printf("%8d", list[i]) ;

  printf("\n") ;
}

//-----------------------------------
main()
{
  int x[]={12,8,0,5,16} ;
  int y[10] ;
  clrscr();
  show_list(x, sizeof(x)/2) ;

  printf("------------------\n") ;
  get_list(y, 10) ;
  printf("------------------\n") ;
  show_list(y, 10) ;
}
```

نکات آموزشی :

- ✓ تابع get_list یک آرایه با طول نامحدود را می خواند. پارامتر count بیانگر تعداد عناصر موجود در آرایه است. همین نکات در باره تابع show_list صدق می کند.
- ✓ به هنگام فراخوانی توابع فوق باید اندازه آرایه را بدانیم. این یک روش ابداعی برای نوشتن توابعی جامع تر با کارایی بیشتر می باشد. البته روش دیگری نیز وجود دارد و آن این است که یک آرایه نامحدود داشته باشیم و تعداد عناصر را در خانه شماره صفر نگهداری نماییم، شبیه نگهداری طول رشته در زبان پاسکال.
- ✓ کلمه void تأکید می کند که تابع مقدار خروجی یا بازگشتی ندارد. البته پارامتر آرایه ای در حکم پارامتر متغیر یا مرجع می باشد.
- ✓ دقت کنید که آرایه x پنج عضوی است ولی آرایه y ، ده عضوی است. عملگر sizeof اندازه داده ها را برحسب بایت محاسبه می نماید. چون هر عدد int دوبایت اشغال می کند، پس برای محاسبه تعداد عناصر آرایه باید سایز را بخش بر دو نماییم.

برنامه شماره ۳۴ : این برنامه مربوط به مرتب سازی، جستجوی خطی و باینری می باشد.

```
#include<stdio.h>
#include<conio.h>
#define n 20

//---------------------------------------
void get_list(int list[] , int count)
{
  int i;
  for(i=0 ; i<=count-1 ; i++) {
    printf("Enter number : ") ;
    scanf("%d", &list[i]) ;
  }
}

//----------------------------------------
void show_list(int list[] , int count)
{
  int i;
  for(i=0 ; i<=count-1 ; i++) {
    textcolor(BLUE) ;
    cprintf("[%d]=", i);
    textcolor(LIGHTGREEN) ;
    cprintf("%d    ", list[i]) ;
    if (wherex() >74) printf("\n") ;
  }
  printf("\n") ;
  textcolor(WHITE) ;
}

//-----------------------------------------------
int linear_search(int list[], int count, int data)
{
  int i ;
  int p=-1 ; //default: data not found in list
  for(i=0 ; i<=count-1 ; i++)
    if (list[i] == data) {
      p = i ; //position of data in list
      break ;
    }
  return p ;
}

//-----------------------------------------------
// direction=1  ascending sort
// direction=0 descending sort
//-----------------------------------------------
void sort_list(int list[] , int count, int direction)
{
  int i, j, temp ;
  for(i=0 ; i<=count-1-1 ; i++)
    for(j=i+1 ; j<=count-1 ; j++)
      if ( ( (direction==1) && (list[i] > list[j]) ) ||
          ( (direction==0) && (list[i] < list[j]) )    )
      {
        // swap list[i] with list[j]
        temp = list[i] ;
        list[i] = list[j] ;
        list[j] = temp ;
      }
}
```

نکات آموزشی :

- ✓ تابع wherex محل فعلی شماره ستون مکان نما را نشان می دهد و تابع wherey محل فعلی شماره سطر را معلوم می کند.
- ✓ در تمام توابع، count بیانگر تعداد عناصر آرایه می باشد.
- ✓ برای مرتب سازی، پارامتر direction درنظر گرفته شده که بیانگر جهت مرتب سازی می باشد. ۱ یعنی صعودی و ۰ یعنی نزولی.
- ✓ در روش جستجوی ترتیبی، داده ها لازم نیست حتماً مرتب شده باشند. حداقل یک و حداکثر n مقایسه نیاز داریم.
- ✓ در روش جستجوی ترتیبی، داده ها باید مرتب شده باشند. حداقل به یک و حداکثر به $\log_2(n)$ مقایسه نیاز داریم. در صورت وجود اعشار،آن را به عدد بالاتر گرد نمایید.
- ✓ روش جستجوی باینری، بسیار سریعتر از روش خطی است. برای درک بیشتر به کتب مرجع و کاملتر مراجعه نمایید.
- ✓ تابع get_list برای دریافت داده هاست ولی برای سریع تر شدن تست برنامه، از آرایه ای با مقادیر ثابت استفاده شده است.
- ✓ توصیه مهم ما به شما آن است که هرگز از نوشتن برنامه های بزرگ و طولانی هیچگونه ترسی به دل خود راه ندهید.

```
//-----------------------------------------------------------------
// direction=1  ascending sort
// direction=0 descending sort
//-----------------------------------------------------------------
int binary_search(int list[], int count, int data, int direction)
{
  int top=0, down=count-1, mid ;
  int p=-1 ; //default: data not found in list

  while (top<=down) {
    mid = (top + down) / 2 ;
    if (list[mid] == data) {
      p = mid ; //position of data in list
      break ;
    }
    else if ( ((direction==1)  && (data < list[mid])) ||
              ((direction==0) && (data > list[mid]))    )
        {
          down = mid - 1 ;
        }
        else
          top = mid + 1 ;
  }
  return p ;
}

//-----------------------------------
main()
{
  int x[n] = {12, 8, 34, 2, 19, 12, 6, 51, 49, 37, 17, 26, 33, 5, 74, 65, 29, 13, 15, 9} ;
  int num, pos ;

  clrscr();
  printf("\n--------- Unsorted list ---------\n") ;
  show_list(x, n) ;

  printf("\nEnter number to search : ") ;
  scanf("%d", &num) ;
  pos = linear_search(x, n, num) ;
  if (pos== -1)
    printf("Data not found!\a\n") ;
  else
    printf("Find in position %d\n", pos) ;

  sort_list(x, n, 1) ;
  printf("\n---------- Sorted list ----------\n") ;
  show_list(x, n) ;

  printf("\nEnter number to search : ") ;
  scanf("%d", &num) ;
  pos = binary_search(x, n, num, 1) ;
  if (pos== -1)
    printf("Data not found!\a\n") ;
  else
    printf("Find in position %d\n", pos) ;
}
```

برنامه شماره ۳۵ : این برنامه نحوه کار با کلیدهای مکان نما و حرکت Cursor را نشان می دهد.

نکات آموزشی :

- ✓ کلیدهای معمولی، یک کد اسکن دارند ولی در کلیدهای ویژه، دو کد توسط بافر صفحه کلید خوانده می شود که اولین کد آن صفر می باشد. بنابراین به دوبار عمل getch نیاز داریم. مثلاً کلید Home دو کد دارد: ابتدا کد 0 و سپس 73 .
- ✓ داده های کاراکتری را هم می شود با مقادیر عددی مقایسه یا چاپ نمود و هم با مقادیر حرفی.
- ✓ تابع gotoxy مکان نما را به ستون و سطر مورد نظر هدایت می کند.
- ✓ در حالت متنی، صفحه نمایش دارای ۸۰ ستون است و تعداد سطرهای آن در محیط ویندوز۹۸ و dos ، ۲۵ تاست و در محیط ویندوز اکس پی، ۵۰ سطر دارد.

```
#include<stdio.h>
#include<conio.h>
main()
{
  int x, y ;
  char  ch1, ch2 ;

  textcolor(YELLOW) ;
  textbackground(BLUE) ;
  clrscr();

  while (1) {
    ch1 = getch() ;
    if (ch1==27) break ; //Escape key;
    if (ch1==0) { //special key
      ch2 = getch() ;
      x = wherex() ;
      y = wherey() ;
      switch (ch2) {
        case  79 : //End key
          gotoxy(80, y) ;
          break ;
        case  71 : //Home key
          gotoxy(1, y) ;
          break ;
        case  72 : //Up arrow
          if (y>1) gotoxy(x, y-1) ;
          break ;
        case  80 : //Down
          if (y<50) gotoxy(x, y+1) ;
          break ;
        case  75 : //Left arrow
          if (x>1) gotoxy(x-1, y) ;
          break ;
        case  77 : //Right key
          gotoxy(x+1, y) ;
          break ;
      } //switch
    } //if
  }  //while
}
```

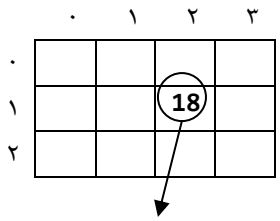
برنامه شماره ۳۶ : این برنامه یک آرایه را دریافت داشته و محتوای آن را معکوس می نماید.

```
#include<stdio.h>
#include<conio.h>
#define n 10

//---------------------------------------
void get_list(int list[] , int count)
{
  int i;

  for(i=0 ; i<=count-1 ; i++) {
    printf("Enter number : ") ;
    scanf("%d", &list[i]) ;
  }
}
```

```
void reverse_list(int list[] , int count)
{
  int i, temp ;
  for(i=0 ; i<=(count-1)/2 ; i++) {
    temp = list[i] ;
    list[i] = list[count-1-i] ;
    list[count-1-i]  = temp ;
  }
}

//----------------------------------------
void show_list(int list[] , int count)
{
  int i;

  for(i=0 ; i<=count-1 ; i++) {
    textcolor(DARKGRAY) ;
    cprintf("[%d]=", i);
    textcolor(LIGHTRED) ;
    cprintf("%d    ", list[i]) ;
    if (wherex() >74) printf("\n") ;
  }
  printf("\n") ;
  textcolor(WHITE) ;
}

main()
{
  int x[n] ;
  int num ;

  clrscr();
  printf("---------- Data Entry ---------\n") ;
  get_list(x, n) ;

  printf("\n--------- Your list ---------\n") ;
  show_list(x, n) ;
  printf("\n------- Reversed list -------\n") ;
  reverse_list(x, n) ;
  show_list(x, n) ;
}
```

برنامه شماره ۳۷ : این برنامه دو ماتریس ٤ × ۳ را دریافت داشته و با هم جمع می کند.

```
#include<stdio.h>
#include<conio.h>
#define m 3
#define n 4

//----------------------------------------
void get_matrix(int matrix[m][n])
{
  int row, col;

  for(row=0 ; row<=m-1 ; row++)
    for(col=0 ; col<=n-1 ; col++) {
      printf("[%d,%d]= ", row, col) ;
      scanf("%d", &matrix[row][col]) ;
    }
}
```

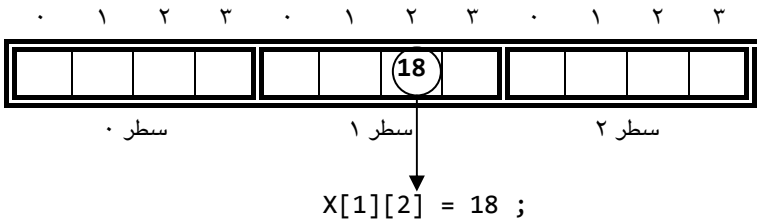


X[1][2] = 18 ;

```
void show_matrix(int matrix[m][n])
{
  int row, col;
  for(row=0 ; row<=m-1 ; row++) {
    for (col=0 ; col<=n-1 ; col++)
      printf("%5d", matrix[row][col]) ;

    printf("\n") ;
  }
}

//-----------------------------------------------------
void sum_matrix(int a[m][n], int b[m][n], int c[m][n])
{
  int row, col;
  for(row=0 ; row<=m-1 ; row++)
    for (col=0 ; col<=n-1 ; col++)
      c[row][col] = a[row][col] + b[row][col] ;
}

main()
{
  int x[m][n], y[m][n], z[m][n] ;
  clrscr();
  printf("---------- First matrix ---------\n") ;
  get_matrix(x) ;
  printf("\n---------- Second matrix ---------\n") ;
  get_matrix(y) ;

  printf("---------------------------------\n") ;
  show_matrix(x) ;
  printf("---------------------------------\n") ;
  show_matrix(y) ;

  sum_matrix(x, y, z) ;
  printf("\n------ Sum of two matrixes ------\n") ;
  show_matrix(z) ;
}
```

نکات آموزشی :

- ✓ همانطور که در شکل صفحه قبل دیدید، برای دسترسی به عناصر یک آرایه دوبعدی باید ابتدا شماره سطر و سپس شماره ستون را بنویسیم.
- ✓ تعداد عناصر آرایه دوبعدی برابر است با حاصل ضرب تعداد سطرها در تعداد ستونها.
- ✓ ماتریسهایی قابل جمع اند که ابعادشان مثل هم باشد.
- ✓ از نظر ساختار حافظه ای، آرایه فقط به صورت یک بعدی است و شکل دوبعدی آن فقط حاصل تصور ذهن ماست و وجود خارجی ندارد. برای درک بهتر به شکل زیرین توجه کنید :



برنامه شماره ۳۸ : این برنامه یک ماتریس مربع را دریافت داشته و آنرا به همراه ترانهاده اش، چاپ و تحلیل می نماید.

```
#include<stdio.h>
#include<conio.h>
#define n 10
//-----------------------------------------
void get_matrix(int matrix[n][n], int count)
{
  int row, col;
  for(row=0 ; row<=count-1 ; row++)
    for(col=0 ; col<=count-1 ; col++) {
      cprintf("[%d,%d]= ", row, col) ;
      scanf("%d", &matrix[row][col]) ;
    }
}
```

```
void transpose_matrix(int matrix[n][n], int count)
{
  int row, col, temp ;
  for(row=1 ; row<=count-1 ; row++)
    for(col=0 ; col<=row-1 ; col++) {
      //exchange rows with columns
      temp = matrix[row][col] ;
      matrix[row][col] = matrix[col][row] ;
      matrix[col][row] = temp ;
    }
}

//-------------------------------------------
int is_balance(int matrix[n][n], int count)
{
  int row, col ;
  int flag=1 ; //default: matrix is balanced
  for(row=1 ; row<=count-1 ; row++)
    for(col=0 ; col<=row-1 ; col++)
      if (matrix[row][col] != matrix[col][row]) {
       flag=0 ; //matrix is not balanced
       break ;
      }
  return flag ;
}

//-------------------------------------------
int is_zero(int matrix[n][n], int count)
{
  int row, col ;
  int flag=1 ; //default: matrix is zero
  for(row=0 ; row<=count-1 ; row++)
    for(col=0 ; col<=count-1 ; col++)
      if (matrix[row][col] != 0) {
       flag=0 ; //matrix is not zero
       break ;
      }
  return flag ;
}

//-------------------------------------------
void show_matrix(int matrix[n][n], int count)
{
  int row, col;
  for(row=0 ; row<=count-1 ; row++) {
    for (col=0 ; col<=count-1 ; col++)
      cprintf("%5d", matrix[row][col]) ;
    printf("\n") ;
  }
}

main()
{
  int x[n][n] ;
  clrscr();
  textcolor(WHITE) ;
  printf("--- Enter matrix ---\n") ;
  get_matrix(x, 4) ;

  printf("\n------- Your matrix -------\n") ;
  textcolor(GREEN) ;
  show_matrix(x, 4) ;
  transpose_matrix(x, 4) ;
  textcolor(WHITE) ;
```

نکات آموزشی :

- ✓ ثابت سراسری n بیانگر حداکثر ابعاد آرایه یا ماتریس مربع است. و count نیز تعداد سطر یا ستون مورد نیاز کاربر است که می تواند از ۲ تا n باشد. در این مثال از ماتریس مربعی ٤ در ٤ استفاده شده است.
- ✓ تابع transpose ترانهاده یک ماتریس را محاسبه می کند. در ماتریس نهاده جای سطرها با ستونها تعویض می گردد.
- ✓ تابع is_balance برای بررسی ماتریس متقارن به کار می رود. در ماتریس متقارن عناصر نسبت به قطر اصلی حالت آینه ای و مساوی دارند: x[i][j] = x[j][i].
- ✓ تابع is_zero بررسی می کند که آیا تمامی عناصر یک آرایه، صفر هستند؟ که در این صورت بدان ماتریس صفر می گویند.

```
  printf("\n---- Transposed matrix ----\n") ;
  textcolor(RED) ;
  show_matrix(x, 4) ;

  printf("-------------------------------\n") ;
  if (is_zero(x, 4)==1) printf("Zero matrix\n") ;
  else printf("Non-zero matrix\n") ;

  if (is_balance(x, 4)==1) printf("Balanced matrix\n") ;
  else printf("Unbalanced matrix\n") ;
}
```

برنامه شماره ۳۹ : این برنامه با استفاده از تا بع خودفراخوان (recursive) فاکتوریل یک عدد صحیح را محاسبه می کند.

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

//-----------------------------
long int fact(int n)
{
  if ((n==0) || (n==1))
    return 1 ;
  else return n * fact(n-1) ;
}

void main(void)
{
  int x ;
  long int k;

  x = 5 ;
  k = fact(x) ;
  printf("%d! = %ld\n" , x, k) ;
}
```

نکات آموزشی :

- ✓ مثال: یک اغذیه فروش یا ساندویچی، از دیگران سفارش و پول دریافت می کند و به آنها غذا تحویل می دهد. آیا این فرد می تواند یکبار هم که شده برای خودش غذا تهیه کرده و از آن استفاده نماید؟ مسلماً جواب شما مثبت است. (مثالهای بیشتری هم وجود دارد)
- ✓ ایده خودفراخوانی نخستین بار توسط یک ریاضیدان به نام "آکِرمَن" مطرح شد. هر گاه تابعی در درون دستوراتش، نام خودش را فرابخواند عمل خود فراخوانی (recursion) انجام داده است. خود فراخوانی ممکن است به دفعات مختلف تکرارشود و باید توسط شروطی کنترل شود تا پس از طی مراحلی، متوقف شود. در غیر اینصورت یک حلقه بی پایان دارد و الگوریتم را دچار مشکل می سازد.
- ✓ برای خود فراخوانی، برنامه نیازمند حافظه ای به نام پشته یا stack می باشد. اگر تعداد خود فراخوانی ها در یک لحظه بیش از حد باشد، برنامه با پیغام خطای سرریزی پشته یا stack overflow مواجه خواهد شد.
- ✓ استفاده از توابع فراخوان راه حل برنامه نویس را کوتاه می کند. اما اجرای برنامه را کندتر می نماید و همیشه نیز برای همه مسایل، قابل برنامه نویسی نیست. برای درک خودفراخوانی ممکن است به تمرینات بیشتر و زمان طولانی تری نیاز داشته باشید.
- ✓ به هنگام خودفراخوانی، اجرای مابقی دستورات تابع، موقتاً ناتمام می ماند. پس از آخرین خودفراخوانی، برنامه به محل ماقبل خود رجوع می کند و بقیه دستورات را به اتمام می رساند و همینطور الی آخر. پس اولین خود فراخوانی، دیرتر از بقیه به اتمام می رسد و بالعکس آخرین فراخوانی، زودتر از بقیه پایان می یابد. این روش به LIFO (Last input is the first output) معروف است. برای مثال در نظر بگیرید که چد بشقاب را به ترتیب از زیر به بالا روی هم قرار می دهید. موقع برداشتن، آخرین بشقاب که روی بقیه است، اول از همه برداشته می شود و برای برداشتن اولین بشقاب که زیر بقیه قرار دارد، باید صبر کنیم تا همه آنها برداشته شوند.
- ✓ در این مثال فاکتوریل عدد ۵ را محاسبه نموده ایم :

ترتیب فراخوانی	ترتیب پاسخ دهی یا بازگشت
Fact(5) = 5 * fact(4) = 5 * ?	Fact(1) = 1
Fact(4) = 4 * fact(3) = 4 * ?	Fact(2) = 2 * 1 = 2
Fact(3) = 3 * fact(2) = 3 * ?	Fact(3) = 3 * 2 = 6
Fact(2) = 2 * fact(1) = 2 * ?	Fact(4) = 4 * 6 = 24
Fact(1) = 1	Fact(5) = 5 * 24 = 120

برنامه شماره ۴۰ : این برنامه با استفاده از تابع خودفراخوان، N امین عدد سری فیبوناچی را محاسبه می کند. قبلاً راجع به اعداد سری فیبوناچی سخن گفته ایم.

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

//-----------------------------
long int fibo(int n)
{
  if ((n==1) || (n==2))
    return 1 ;
  else return fibo(n-1) + fibo(n-2) ;
}

void main(void)
{
  long int k;

  clrscr() ;
  k = fibo(7) ;
  printf("%ld\n" ,k) ;
}
```

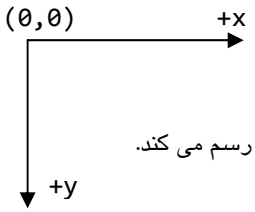
برنامه شماره ۴۱ : برای رفع خستگی و ایجاد تنوع، در اینجا به اولین برنامه گرافیکی می پردازیم. درک مقدمات گرافیک، بسیار مهم است. اگر کامپیوتروارد حالت گرافیکی نشود، هیچ دستور ترسیماتی در محیط text کار نخواهد کرد.

```
#include<graphics.h>
#include<stdlib.h>
#include<stdio.h>
#include<conio.h>

int main(void)
{
   int gd = DETECT, gm, code ;
   int x, y;

   initgraph(&gd, &gm, "");
   code = graphresult();
   if (code != grOk) {
      printf("Graphics error!\n");
      exit(1);
   }

   x = getmaxx() ;
   y = getmaxy() ;
   line(0, 0, x, y);
   setcolor(RED) ;
   circle(x/2 ,y/2, 50) ;

   getch();
   closegraph();
   return 0;
}
```

نکات آموزشی :

- ✓ صفحه نمایش درحالت متن (text) دارای ۲۵ یا ۵۰ سطر و ۸۰ ستون می باشد. در حالت گرافیکی دارای تعدادی نقطه یا پیکسل است. مثلاً ۴۸۰ × ۶۴۰ نقطه. برای رفتن به حالت گرافیکی، شرکت بورلند فایلهایی را به عنوان درایور نرم افزاری ارائه داده است که دارای پسوند BGI. می باشند. همچنین برای ارائه انواع فونت یا قلم، فایلهایی با پسوند CHR. وجود دارند. دستور initgraph با توجه به درایور و مود گرافیکی، صفحه نمایش را به حالت گرافیکی می برد. حالت پیش فرض مود گرافیکی DETECT یا همان عدد صفر می باشد. یعنی سیستم بهترین گزینه ممکن را خودش کشف نماید.

- ✓ چنانچه فایلهای فوق الذکر در مسیر جاری اجرای برنامه موجود باشند، از یک رشته خالی استفاده می کنیم وگرنه باید مسیر را مشخص کنیم. برای علامت \ در بیان درایو و مسیر باید از \\ استفاده کنیم. چون \ یک علامت ویژه است مثل \n و ... .
- ✓ gm بیانگر مود گرافیکی و gd معرف درایور گرافیکی است. پس از initgraph تابع graphresult نتیجه عمل را مشخص می کند. اگر نتیجه کار grOk یا همان صفر باشد، بدین معنی است که عمل گرافیکی بدون خطا آغاز شده است. وگرنه با دستور exit از برنامه خارج می شویم. کد یک به معنی خطاست که به سیستم عامل برگردانده می شود. برای استفاده از تابع exit نیازمند فایل راهنمای stdlib.h می باشیم.
- ✓ توابع getmaxx و getmaxy به ترتیب حداکثر طول و حداکثر عرض در مختصات صفحه را به دست می آورند. نقطه مبدأ مختصات 0,0 می باشد. محور مختصات گرافیکی در زبان سی بدین شکل می باشد :



- ✓ دستور line به ترتیب مختصات نقطه اول و دوم یک خط را دریافت داشته و آنرا رسم می کند.
- ✓ دستور setcolor رنگ ترسیمات را مشخص می کند.
- ✓ دستور circle با توجه به طول و عرض مرکز و اندازه شعاع، دایره ای رسم می کند. این دایره توخالی است.
- ✓ دستور closegraph حالت گرافیکی را پایان داده و صفحه را پاک می کند.
- ✓ با دستور return مقدار صفر یعنی بدون خطا، توسط تابع main به سیستم عامل ارجاع می شود. که البته این امر اختیاری است و برای تأکید بیشتر است. گاهی اوقات برنامه بدون error کامپایل و اجرا می شود اما دارای warning یا هشدار است. این مسئله نیز یکی از همین موارد است.

برنامه شماره ۴۲ : چندین شکل گرافیکی بر روی صفحه رسم می نماید.

```
#include<graphics.h>
#include<stdlib.h>
#include<stdio.h>
#include<conio.h>

int main(void)
{
  int gd = DETECT, gm, code ;
  int i, x, y, r, c;

  initgraph(&gd, &gm, "");
  code = graphresult();
  if (code != grOk) {
    printf("Graphics error!\n");
    exit(1);
  }

  x = 100 ;  y = 100 ;
  r = 5 ;
  for (i=1; i<=10; i++) {
    circle(x, y, r) ;
    r += 5 ;
  }
  getch();

  cleardevice() ;
  setcolor(GREEN) ;
  x = 100 ;  y = 100 ;
  r = 5 ;
  for (i=1; i<=10; i++) {
    circle(x, y, r) ;
    r += 5 ;
    x += 5 ;
  }
  getch() ;
```

نکات آموزشی :

- ✓ دستور cleardevice صفحه گرافیکی را پاک می کند.
- ✓ دستور setviewport همانند دستور window ، صفحه نمایش را به یک پنجره محدود می کند که به ترتیب مختصات نقطه چپ بالا و نقطه راست پایین می باشد. پارامتر پنجم مربوط به آن است که آیا قسمتهای اضافی اشکال از کادر بیرون باشند یا برش خورده شوند؟ اگر صفر باشد اضافات تصویر از کادر خارج می شود وگرنه برش می خورد.
- ✓ توجه کنید که پس از اجرای این دستور، نقطه سمت چپ بالا به عنوان 0,0 در نظر گرفته می شود.

```
  setviewport(200, 200, 400,400, 1) ;
  setcolor(BLUE) ;
  x = 100 ;    y = 100 ;
  r = 5 ;
  for (i=1; i<=10; i++) {
    circle(x, y, r) ;
    r += 5 ;
    x += 5;
    y += 5 ;
  }

  getch();
  closegraph();
}
```

برنامه شماره ۴۳ : چندین شکل گرافیکی بر روی صفحه رسم می نماید. که برخی از آنها توپر یا هاشوردار می باشند.

```
#include<graphics.h>
#include<stdlib.h>
#include<stdio.h>
#include<conio.h>

int main(void)
{
  int gd = DETECT, gm, code ;
  int i;

  initgraph(&gd, &gm, "");
  code = graphresult();
  if (code != grOk) {
    printf("Graphics error!\n");
    exit(1);
  }

  for (i=1 ; i<=11 ; i++) {
    setfillstyle(i, MAGENTA) ;
    bar(50, 50, 200, 200) ;
    getch() ;
  }

  setfillstyle(1, LIGHTGREEN) ;
  setcolor(RED) ;
  bar3d(100,300, 200, 350, 20, 1) ;
  getch() ;

  ellipse(300, 300, 0, 180, 50, 30) ;
  getch() ;

  setcolor(LIGHTBLUE) ;
  ellipse(350, 100, 0, 360, 40, 25) ;
  rectangle(320 ,50, 380, 150) ;
  setfillstyle(2, LIGHTBLUE) ;
  floodfill(350, 100, LIGHTBLUE) ;
  getch() ;
  closegraph();
}
```

نکات آموزشی :

- ✓ دستور setfillstyle نحوه رنگ آمیزی درون اشکال بسته را مشخص می کند. پارامتر اول، نوع هاشور و پارامتر دوم نوع رنگ را مشخص می کند.
- ✓ دستور bar یک مستطیل توپر رسم می کند. مختصات دو کنج یا رأس مقابل آن را باید بنویسیم.
- ✓ دستور bar3d دارای حالتی سه بعدی است. پارامتر پنجم میزان عمق را مشخص می کند. پارامتر ششم مربوط به خط بالای مستطیل است. اگر مخالف صفر باشد، خط بالای مستطیل رسم خواهد شد وگرنه رسم نمی شود. این حالت برای مواقعی خوب است که بخواهیم چند مستطیل را بصورت روی هم انباشته رسم نماییم.
- ✓ شکل کلی پارامترهای رسم بیضی در دستور ellipse بدین صورت است :

(قطرعمودی ,قطرافقی ,زاویه پایان ,زاویه شروع ,عرض ,طول)

- ✓ دستور rectangle مستطیل توخالی رسم می کند و پارامترهایی آن همانند line می باشد.
- ✓ دستور floodfill برای رنگ آمیزی درون یک محیط بسته می باشد که باهم یک رنگ می باشند.

برنامه شماره ۴۴ : متنهایی را بر روی صفحه می نویسد. نقاطی با رنگهای مختلف را به صورت تصادفی ایجاد می کند.

```
#include<graphics.h>
#include<stdlib.h>
#include<stdio.h>
#include<conio.h>

int main(void)
{
  int gd = DETECT, gm, code ;
  int i, int x, y, x1, y1, color;

  initgraph(&gd, &gm, "");
  code = graphresult();
  if (code != grOk) {
    printf("Graphics error!\n");
    exit(1);
  }

  setviewport(50,50,150,150, 1) ;
  x = getmaxx() ;
  y = getmaxy() ;

  randomize() ;
  while (kbhit()==0) {
    x1 = random(x+1) ;
    y1 = random(y+1) ;
    color = random(16) ;
    putpixel(x1, y1, color) ;
  }

  while (kbhit()==0) {
    x1 = random(x+1) ;
    y1 = random(y+1) ;
    color = random(16) ;
    putpixel(x1, y1, color) ;
  }
  getch() ;

  setviewport(0, 0, x, y, 1) ;
  settextstyle(3, 0, 4) ;
  setcolor(YELLOW) ;
  outtextxy(220, 250, "Borujerd.") ;
  moveto(100, 300) ;
  outtext("Gorgan.") ;
  settextstyle(4, 1, 8) ;
  outtextxy(400, 100, "IRAN.") ;
  getch() ;
  closegraph();
}
```

نکات آموزشی :

- ✓ دستور putpixel در مختصات تعیین شده با رنگ مشخص، یک نقطه می نگارد. عکس این عمل تابع getpixel است که رنگ نقطه تعیین شده را برمی گرداند :

  متغیر صحیح = getpixel(عرض , طول) ;

- ✓ تابع random یک عدد تصادفی بین ۰ تا n-1 از آرگومان به دست می آورد. برای جلوگیری از تکرار نتایج از دستور randomize استفاده می شود.
- ✓ تابع kbhit مربوط به زده شدن صفحه کلید است (همانند تابع keypreesed در زبان توربوپاسکال). اگرکلیدی زده شود، مقدار خروجی آن عددی غیر صفر است.
- ✓ تابع settextstyle برای تنظیم مشخصات متن می باشد. اولین عدد بیانگر نوع قلم است (بین ۰ تا ٤). دومین پارامتر برای جهت متن است که ۰ یعنی افقی و ۱ یعنی عمودی. سومین عدد مربوط به سایز یا اندازه متن است.
- ✓ تابع outtext در محل جاری مکان نما، متنی را می نویسد. مکان نما در حالت متنی دیده می شود و در حالت گرافیک قابل رویت نمی باشد.
- ✓ تابع outtextxy نیز همین گونه است ولی می توان محل شروع متن را مشخص نمود.
- ✓ تابع moveto مکان نما را به مختصات دلخواه هدایت می نماید.

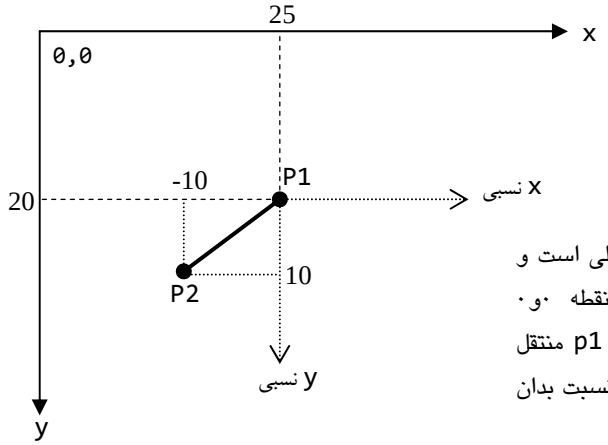
برنامه شماره ۴۵ : متنی را در وسط صفحه می نویسد. خطوط را نیز با روش مختصات دهی نسبی رسم می نماید.

```
#include<graphics.h>
#include<stdlib.h>
#include<stdio.h>
#include<conio.h>

int main(void)
{
  int gd = DETECT, gm, code ;
  int x, y ;
```

نکات آموزشی :

- ✓ تابع textwidth با توجه به نوع و اندازه قلم، پهنای افقی متن مورد نظر را به دست می آورد. تابع textheight ارتفاع عمودی متن را محاسبه می کند.
- ✓ دستور linerel از محل فعلی مکان نما، یک خط تا نقطه تعیین شده بعدی رسم می کند. مختصات نقطه فعلی بعنوان مبداء در نظر گرفته شده و نقطه بعدی نسبت به آن سنجیده می شود. در این روش از مختصات نسبی استفاده می شود.

```
  initgraph(&gd, &gm, "");
  code = graphresult();
  if (code != grOk) {
    printf("Graphics error!\n");
    exit(1);
  }

  settextstyle(1, 0, 6) ;
  x = (getmaxx() - textwidth("NASTARAN")) / 2 ;
  y = (getmaxy() - textheight("NASTARAN")) / 2 ;
  setcolor(MAGENTA) ;
  outtextxy(x, y, "NASTARAN") ;
  getch() ;

  setcolor(YELLOW) ;
  moveto(100, 100) ;
  linerel(50, 0) ;
  linerel(0, 70) ;
  linerel(-50, 0) ;
  moveto(300, 200) ;
  linerel(50, 0) ;
  linerel(50, -50) ;
  linerel(-50,0) ;

  getch() ;
  closegraph();
}
```



در این شکل p1 نقطه فعلی است و p2 نقطه بعدی است. نقطه ۰و۰ مبداء مختصات موقتاً به p1 منتقل می شود و نقطه p2 نیز نسبت بدان سنجیده می شود.

برنامه شماره ۴۵ : مفهوم رکورد را شرح می دهد. رکورد اطلاعات یک دانش آموز را معرفی می نماید.

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

struct student {
  int code ;
  char name[10] ;
  float no ;
}

main()
{
  struct student z, x, y = {3321,  "alavi", 14.5} ;

  clrscr() ;
  printf("%d  %s  %2.2f\n\n", y.code, y.name, y.no) ;
  printf("Enter student code: ") ;
  scanf("%d", &x.code) ;
  printf("Enter student name: ") ;
  scanf("%s", &x.name) ;
  printf("Enter student no: ") ;
  scanf("%f", &x.no) ;
  z = x ;
  printf("\n%d  %s  %2.2f\n\n", z.code, z.name, z.no) ;
  getch() ;
}
```