مبانی کامپیوتر و برنامه‌نویسی به زبان C

فصل نهم: رکوردها ، نوع‌های جدید و حافظه‌ی پویا

۹-۱ مقدمه

آرایه‌ها: تعریف خانه‌های هم‌نوع، هم‌طول و هم‌جوار

نیاز به تعریف عناصر گوناگون در کنار هم و منسوب کردن همه‌ی آن‌ها به یک نام

بازنویسی آن‌ها به‌طور یک‌جا، ارسال کل آن‌ها به یک تابع و برگرداندن همه‌ی آن‌ها به‌عنوان نتیجه

مثال: داده‌های مربوط به یک دانشجو

رکوردها: تعریف مجموعه‌ای از فقره‌های داده اکثراً غیرهم‌نوع و غیرهم‌طول به‌صورت هم‌جوار

۹-۲ رکوردها

معادلی برای structure

تعریف تعدادی فقره‌ی داده‌ی متفاوت تحت یک نام

مشابه تعریف رکورد در سازمان‌دهی منطقی داده‌ها روی حافظه‌های کمکی در فصل اول

استفاده از کلمه‌ی ساختار (که نام کلی‌تری است) در بعضی کتاب‌ها به این منظور

۹-۲-۱ تعریف و مقداردهی اولیه‌ی رکوردها و ارتباطشان با آرایه‌ها

قالب کلی:

struct ‹نام الگو› {‹تعریف عضوهای رکورد›} ‹لیست نام‌ها در صورت نیاز› ;

مثال رکورد یک دانشجو:

```
struct student_record                /* تعریف رکورد دانشجو */
{  long  student_number;             /* شماره‌ی دانشجویی */
   char  first_name[۲۱];             /* نام دانشجو */
   char  last_name[۳۱];              /* نام خانوادگی */
   char  gender_code;                /* کد جنسیت */
   float gpa;                        /* معدل کل */
   int   units_passed;               /* تعداد کل واحد گذرانده شده */
} student۱;                          /* تعریف متغیری با ساختمان این رکورد به‌نام student۱ */
```

student_record نام الگو، می‌تواند نوشته نشود

student۱ نام رکوردی متشکل از شش عضو:

student_number، first_name، last_name، gender_code، gpa و units_passed

عدم ذکر نام بعد از خاتمه‌ی تعریف رکورد: فقط تعریف یک الگو بدون تخصیص حافظه

مثال:

```
struct student_record hassan, ali, students[۵۰];
```

تعریف رکوردهایی مشابه student۱ برای hassan و ali و آرایه‌ی ۵۰ عنصری students

عدم ذکر نام الگو: عدم امکان تعریف مجدد به شکل بالا

امکان قرار دادن نام‌های ali و hassan و همچنین آرایه‌ی students بعد از نام student۱

نام‌های مشابه برای الگو، عضوها، عضوهای دو رکورد مستقل

رکوردها قابل تعریف به‌صورت تودرتو

شکل ۹-۱ تعریف رکورد دانشجو

تعریف رکوردی به‌طور کامل در داخل یک رکورد دیگر

نحوه‌ی ارائه‌ی مقدار اولیه برای عضوهای یک رکورد

عدم وجود تعداد کافی مقادیر

```
struct date_record                          /* تعریف رکورد تاریخ */
{  int day;                                 /* شماره‌ی روز */
   char month[۱۲];                          /* نام ماه */
   int year;                                /* شماره‌ی سال */
};

struct student_record                       /* تعریف رکورد دانشجو */
{  long   student_number;                   /* شماره‌ی دانشجویی */
   char   first_name[۲۱];                   /* نام دانشجو */
   char   last_name[۳۱];                    /* نام خانوادگی */
   struct date_record date_of_birth;        /* رکورد تاریخ تولد */
   char   gender_code;                      /* کد جنسیت */
   struct date_record diploma_date;         /* رکورد تاریخ اخذ دیپلم */
   float  gpa;                              /* معدل کل */
   int    units_passed;                     /* تعداد کل واحد گذرانده شده */
} student۱, hassan, ali, students[۵۰];      /* اسامی متغیرها با ساختمان رکورد دانشجو */
struct student_record student۲ =  /* مقداردهی اولیه‌ی متغیر دیگری با ساختمان رکورد دانشجو */
{۷۲۱۰۰۳۳۵,"Hassan","Tehrani",۲۹,"Tir",۳۴,'M',۱,"Day",۶۸،۱۵.۵،۱۱۰};
```

شکل ۹-۱: نمونه‌هایی از تعریف و مقداردهی اولیه‌ی انواع رکوردها.

۹-۲-۲ عملیات روی عضوهای رکورد و کل رکورد

هر عضو رکورد یک متغیر معمولی، امکان شرکت در عمل‌هایی که برای آن نوع متغیر تعریف شده

دسترسی به عضوهای یک رکورد با عملگر عضو رکورد (علامت نقطه)

نام رکورد قبل از نقطه و نام عضو بعد از نقطه

عملگر نقطه دارای بالاترین تقدم عملیات بوده و ترتیب اجرا از چپ به راست

استفاده از عملگر مزبور به‌صورت متوالی در رکوردهای تودرتو

قرارداد: عدم وجود فاصله در دو طرف عملگر نقطه

```
strcpy(student۱.first_name, “Reza”);/* student۱ درج نام در فیلد مربوطه از رکورد */
ali.student_number = ۵۲۱۰۰۳۳۵;        /* ali ثبت شماره‌ی دانشجویی در فیلد مربوطه از رکورد */
hassan.date_of_birth.year = ۱۳۲٤; /* hassan درج سال تولد در رکورد تاریخ تولد از رکورد */
students[۱۵].gpa = ۱۷.۵;              /* students تعیین معدل کل در رکورد شماره ۱۵ آرایه‌ی */
student۲.units_passed = ۱۲۰;        /* student۲ ثبت تعداد واحد گذرانده شده در رکورد */
                        /* students تعیین ماه اخذ دیپلم در رکورد تاریخ اخذ دیپلم از آخرین رکورد آرایه‌ی */
strcpy(students[٤۹].diploma_date.month, “Farvardin”);
                        /* students ثبت نام خانوادگی در رکورد شماره‌ی پنج از آرایه‌ی */
strcpy(students[۵].last_name, “RezaianzadeheMotlagheFardeAala”);
scanf(“%s,%s,%ld”,    /* ali خواندن نام، نام خانوادگی و شماره‌ی دانشجویی به داخل فیلدهای مربوطه در رکورد */
      ali.first_name, ali.last_name, &ali.student_number);
printf(“%ld,%s,%f\n”,          /* ali چاپ شماره‌ی دانشجویی، نام خانوادگی و معدل از رکورد */
       ali.student_number, ali.last_name, ali.gpa);
                        /* students مقایسه‌ی معدل‌های موجود در رکوردهای شماره‌ی ٤ و ۵ از آرایه‌ی */
if (students[٤].gpa == students[۵].gpa)
   printf(“The two gpa’s are equal.\n”);
                        /* students مقایسه‌ی نام‌های موجود در رکوردهای شماره‌ی ٤ و ۵ از آرایه‌ی */
if (!strcmp(students[٤].last_name, students[۵].last_name))
```

شکل ۹-۲: نمونه‌هایی از انجام انواع عملیات روی عضوهای رکورد.

انجام عملیات روی کل رکورد به‌صورت یک‌جا

تخصیص رکوردها به‌صورت یک‌جا، ارسال به یک تابع ویا بازگرداندبه عنوان نتیجه‌ی تابع

عدم امکان انجام هر عمل دیگر (مقایسه‌ی رکوردها، خواندن و نوشتن آن‌ها به‌صورت یک‌جا، . . .)

تخصیص یک رکورد به رکورد دیگری با ساختمان دقیقاً مشابه

انتقال محتویات هر فیلد از رکورد مبدأ به فیلد متناظر از رکورد مقصد

امکان مقداردهی اولیه فقط در تعریف رکورد

```
                                /* students حلقه‌ی تکرار برای خواندن داده‌های لازم به داخل ۵۰ رکورد آرایه‌ی */
for (i = ۰; i < ۵۰; i++)
{  scanf("%ld%s%s%c%f%d", &students[i].student_number,
         students[i].first_name, students[i].last_name,
         &students[i].gender_code, &students[i].gpa,
         &students[i].units_passed);
   scanf("%d%s%d", &students[i].date_of_birth.year,
         students[i].date_of_birth.month,
         &students[i].date_of_birth.day);
   scanf("%d%s%d", &students[i].diploma_date.year,
          students[i].diploma_date.month,
          &students[i].diploma_date.day);
}
                              /* تخصیص داده‌های موجود در رکورد شماره‌ی پنج از آرایه‌ی students به رکورد hassan */
hassan = students[۵];

                        /* تخصیص داده‌های موجود در رکورد شماره‌ی صفر از آرایه‌ی students به رکوردهای ali و student۱ */
ali = student۱ = students[۰];

                      /* تخصیص داده‌های موجود در رکورد hassan به رکورد student۲ و آخرین رکورد از آرایه‌ی students */
student۲ = students[۴۹] = hassan;
```

شکل ۹-۳: نمونه‌هایی از تخصیص رکوردها به یک‌دیگر.

برنامه‌ی نمونه‌ی ۹-۱ خواندن رکوردهای دانشجویان، مرتب‌سازی و چاپ داده‌های مرتب شده
الگوریتم جدید مرتب‌سازی، جابه‌جا نمودن داده‌های دانشجویان به‌طور یکجا
مقایسه‌ی این برنامه با برنامه‌ی نمونه‌ی ۸-۶ استفاده از آرایه‌های معمولی

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAXNOS ۱۰۰                                  /* حداکثر تعداد دانشجو */
#define LINEPPG ۲۵                                  /* تعداد سطر مورد چاپ در یک صفحه */
#define SORTED ۱                                    /* وضعیت مبنی بر مرتب بودن داده‌ها */
#define LINEWIDTH ۷۸                                /* عرض سطر مورد چاپ */

main()
{  struct std_rec                                   /* تعریف رکورد دانشجو */
   {  long  std_numb;
      char  std_first[۲۱];
      char  std_last[۳۱];
      float gpa;
   };
   struct std_rec std[MAXNOS], tmp;                 /* تعریف آرایه و رکورد کمکی برای ذخیره و مرتب‌سازی داده‌ها */
   int i, nofst, sort_stat;
   char dash[LINEWIDTH + ۱];                        /* تعریف آرایه‌ی حاوی خط افقی */

   scanf("%d", &nofst);
   for (i = ۰; i < nofst; i++)                      /* حلقه‌ی تکرار خواندن داده‌های دانشجویان */
      scanf("٪ld٪۲۰s٪۳۰s٪۵f",
            &std[i].std_numb, std[i].std_first,
            std[i].std_last, &std[i].gpa);
   sort_stat = !SORTED;
   while (sort_stat != SORTED)                      /* حلقه‌ی تکرار بررسی مرتب‌بودن داده‌ها */
                                                    /* حلقه‌ی تکرار مقایسه‌ی رکوردهای متوالی */
      for (sort_stat = SORTED, i = ۰; i < nofst - ۱; i++)
         if (strcmp(std[i].std_last, std[i + ۱].std_last) > ۰)
         {  tmp = std[i];                           /* جابه‌جاکردن دو رکورد متوالی از طریق انتساب رکورد به رکورد */
            std[i] = std[i + ۱];
            std[i + ۱] = tmp;
            sort_stat = !SORTED;
         }
   memset(dash, '-', LINEWIDTH);                    /* ساختن خط افقی در حافظه جهت چاپ در لیست */
   dash[LINEWIDTH] = '\۰';
   for (i = ۰;i < nofst;i++)                        /* حلقه‌ی تکرار چاپ داده‌های مرتب شده */
   {  if (!(i % LINEPPG))
      {  printf("\f  Seq  Student_no First Name          ");
         printf("  Last Name                      GPA\n");
         printf("%s\n", dash);                      /* چاپ خط افقی */
      }
      printf(" ٪۵d ٪۷ld      ٪-۲۰s  %-۳۰s ٪۵.۲f\n\n",
             i + ۱, std[i].std_numb, std[i].std_first,
             std[i].std_last, std[i].gpa);
   }
```

شکل ۹-۷: متن برنامه‌ی ۹-۱ مرتب‌سازی رکوردهای دانشجویان.

۹-۲-۳ رکوردها و توابع

پارامترهایی با ساختمان رکورد در تعریف یک تابع

ارسال آرگومان متناظر در احضار

برگرداندن رکورد به عنوان نتیجه‌ی تابع

ارسال آدرس یک رکورد به توابع و دستیابی به عضوها از طریق آدرس

استفاده‌ی مشترک از تعریف رکورد (تعاریف سراسری)

استفاده از رکوردها و متغیرهای تعریف شده در داخل یک تابع فقط در محدوده‌ی همان تابع

تعریف رکورد و متغیر در خارج از همه‌ی توابع و استفاده‌ی مشترک در سرتاسر فایل (تعاریف سراسری)

تعاریف سراسری در یک فایل سرآمد و استفاده از فرمان `#include` و یک زوج علامت نقل قول

پارامتر و آرگومان با ساختمان رکورد

تعریف پارامترهایی با ساختمان رکورد و ارسال آرگومان‌هایی دقیقاً از همان ساختمان‌ها

ارسال آرگومان‌ها به صورت انتقال مقدار (و نه انتقال آدرس) به برنامه‌ی احضار شونده

برنامه‌ی ۹-۲: ساختاری برای تعریف نقطه روی صفحه‌ی مختصات تعریف کنید و سپس مجموعه توابعی بنویسید که مختصات دو نقطه‌ی مختلف را از ورودی بخواند و فاصله‌ی بین دو نقطه‌ی مزبور و همچنین مساحت مستطیلی که این دو نقطه دو گوشه‌ی سمت چپ ضلع بالا و سمت راست ضلع پایین آن را نشان می‌دهد، را محاسبه نموده همراه با توضیح مناسب چاپ نماید (فرمول محاسبه‌ی فاصله‌ی دو نقطه در برنامه‌ی نمونه‌ی ۷-۳ در فصل هفتم ارائه شده است).

```
#include <stdio.h>
#include <math.h>
#define TRUE ۱                          /* وضعیت مبنی بر شرط همیشه درست */

struct point                            /* تعریف رکورد نقطه به‌صورت سراسری */
{  float  x;
   float  y;
};

                                /* تابعی برای محاسبه‌ی فاصله‌ی دونقطه و مساحت مستطیل ساخته شده به‌وسیله‌ی دو نقطه */
float process_point(struct point first, struct point second,
                    float *surface)
{  float distance;

   distance = sqrt(pow(first.x - second.x, ۲) +/* محاسبه‌ی فاصله در یک متغیر محلی */
                   pow(first.y - second.y, ۲));
                                /* محاسبه‌ی مساحت مستطیل در یک پارامتر که آدرسش ارسال شده است */
   *surface = abs((first.x - second.x) * (first.y - second.y));
   return distance;             /* بازگرداندن فاصله‌ی بین دو نقطه به عنوان نتیجه‌ی تابع */
}

main()
{  struct point p, q;              /* تعریف دو متغیر از نوع رکورد نقطه برای ذخیره‌ی مختصات دو نقطه */
   int i, input_state;
   float point_dist, rect_surface;

   while (TRUE)                    /* حلقه‌ی تکرار خواندن مختصات هر جفت نقطه */
   {  printf("Enter x,y for the next two points:");
      input_state = scanf("%f%f%f%f", &p.x, &p.y, &q.x, &q.y);
      if (input_state == EOF)      /* تشخیص خاتمه‌ی داده‌ها و ترک حلقه‌ی تکرار */
         break;
                        /* احضار تابع، ارسال مختصات یک جفت نقطه به آن وبه‌دست آوردن فاصله و مساحت مستطیل */
      point_dist = process_point(p, q, &rect_surface);
      printf("The two points are: (%f, %f) and (%f, %f)\n",/* چاپ نتایج */
             p.x, p.y, q.x, q.y);
      printf("Distance between these points is: %f\n", point_dist);
      printf("Rectangle surface is: %f\n", rect_surface);
   }
   return ۰;
}
```

شکل ۹-۸: متن برنامه‌ی ۹-۲ انجام محاسبات روی مختصات دو نقطه.

بازگرداندن رکورد به‌عنوان نتیجه‌ی تابع

برنامه‌ی ۹-۳: برنامه‌ی ۹-۲ را در نظر بگیرید و فرض کنید بخواهیم علاوه بر موارد خواسته شده در آن برنامه مختصات حاصل از مجموع هر دو نقطه‌ی خوانده شده از ورودی را نیز محاسبه نموده و چاپ نماییم. همچنین می‌خواهیم برای هر یک از موارد خواندن مختصات یک نقطه از ورودی، محاسبه‌ی فاصله‌ی دو نقطه، محاسبه‌ی مساحت مستطیل ایجاد شده توسط دو نقطه و محاسبه‌ی جمع دو نقطه یک تابع مجزا بنویسیم.

```
                                        /*  تابعی برای خواندن مختصات یک نقطه و تشخیص خاتمه‌ی داده‌های ورودی  */
struct point read_point(int *ok)
{  struct point temp;

   printf("Enter the next point coordinates as x y:");
   if (scanf("%f%f", &temp.x, &temp.y) == EOF)
      *ok = temp.x = temp.y = ۰; /*تشخیص خاتمه‌ی داده‌ها و ثبت صفر در مختصات و پارامتر وضعیت*/
   else
      *ok = ۱;
   return temp;                         /*  بازگرداندن نقطه‌ای که مختصات آن خوانده شده به عنوان نتیجه‌ی تابع  */
}
                                        /*  تابعی برای محاسبه‌ی فاصله‌ی بین دونقطه  */
float distance(struct point first, struct point second)
{  float distance;

   distance = sqrt(pow(first.x - second.x, ۲) +/*محاسبه‌ی فاصله در یک متغیر محلی*/
                   pow(first.y - second.y, ۲));
   return distance;                     /*  بازگرداندن فاصله‌ی بین دو نقطه به عنوان نتیجه‌ی تابع  */
}
                                        /*  تابعی برای محاسبه‌ی مساحت مستطیل ساخته شده به‌وسیله‌ی دو نقطه  */
float surface(struct point first, struct point second)
{  float surface;
                                        /*  محاسبه‌ی مساحت مستطیل در یک متغیر محلی  */
   surface = abs((first.x - second.x) * (first.y - second.y));
   return surface;                      /*  بازگرداندن مساحت مستطیل به عنوان نتیجه‌ی تابع  */
}
                                        /*  تابعی برای محاسبه‌ی حاصل‌جمع دو نقطه  */
struct point add_points(struct point first, struct point second)
{                       /*  ذخیره‌ی حاصل‌جمع دو نقطه در یکی از پارامترها بدون هیچ تأثیری روی آرگومان متناظر  */
   first.x += second.x;
   first.y += second.y;
   return first;        /*  بازگرداندن پارامتر اول که حاوی حاصل‌جمع دو نقطه است به عنوان نتیجه‌ی تابع  */
}
```

شکل ۹-۹ ب: متن توابع مربوط به برنامه‌ی ۹-۳ انجام محاسبات روی مختصات دو نقطه.

```
#include <stdio.h>
#include <math.h>
#define TRUE ۱                          /*  وضعیت مبنی بر شرط همیشه درست  */

struct point                            /*  تعریف رکورد نقطه به‌صورت سراسری  */
{  float  x;
   float  y;
};

          /*  . . . . . محل قرار گرفتن چهار تابع نوشته شده در شکل ۹-۹ ب . . . . .  */

main()
{  struct point p, q, sum;/*تعریف سه متغیر از نوع رکورد نقطه برای ذخیره‌ی مختصات دو نقطه و حاصل‌جمع آن‌ها*/
   int i, input_ok;
   float point_dist, rect_surface;

   while (TRUE)                         /*  حلقه‌ی تکرار خواندن مختصات هر جفت نقطه  */
   {              /*  احضار تابع مربوطه برای خواندن مختصات نقطه‌ی اول و تخصیص نتیجه به یک متغیر از نوع رکورد نقطه  */
      p = read_point(&input_ok);
      if (!input_ok)                    /*  تشخیص خاتمه‌ی داده‌ها و ترک حلقه‌ی تکرار  */
         break;
      q = read_point(&input_ok);/*  احضار تابع مربوطه برای خواندن مختصات نقطه‌ی دوم  */
      if (!input_ok)                    /*  تشخیص خاتمه‌ی داده‌ها و ترک حلقه‌ی تکرار  */
         break;
      point_dist = distance(p, q); /*  احضار تابع محاسبه‌ی فاصله و ارسال مختصات دو نقطه به آن  */
      rect_surface = surface(p, q);/*  احضار تابع محاسبه‌ی مساحت و ارسال مختصات دو نقطه به آن  */
                        /*  احضار تابع محاسبه‌ی حاصل‌جمع دو نقطه و تخصیص نتیجه به یک متغیر از نوع رکورد نقطه  */
      sum = add_points(p, q);
                                        /*  چاپ نتایج  */
      printf("The two points are: (%.۲f, %.۲f) and (%.۲f, %.۲f)\n",
             p.x, p.y, q.x, q.y);
      printf("Distance between these points is: %.۲f\n",
             point_dist);
      printf("Rectangle surface is: %.۲f\n", rect_surface);
      printf("The sum of two points is: (%.۲f, %.۲f)\n",
             sum.x, sum.y);
   }
   return ۰;
}
```

شکل ۹-۹ پ: متن تابع اصلی مربوط به برنامه‌ی ۹-۳ انجام محاسبات روی مختصات دو نقطه.

اشاره‌گر به رکورد به عنوان پارامتر

دستیابی و پردازش مستقیم محتویات عضوهای یک رکورد از تابع احضارکننده در تابع احضارشونده

ارسال آدرس متغیر مزبور به تابع احضارشونده

نحوه‌ی دسترسی به عضوهای این‌گونه پارامترها در تابع احضارشونده

مثلاً پارامتر p اشاره‌گر به یک رکورد و حاوی آدرس یک رکورد از آن نوع و a نام یکی از عضوهای رکورد p* خود آن رکورد و عبارت (*p).a عضوهای رکورد مزبور

قراردادن عملگر * در داخل زوج پرانتز به این دلیل تقدم

آسان نمودن کار این‌گونه دستیابی‌ها با عملگر خاصی به شکل <- (دستیابی غیرمستقیم به عضوهای رکورد)

معادل بودن دو عبارت p->a و (*p).a

قرار ندادن فاصله‌ی خالی در دوطرف این عملگر

برنامه‌ی ۹-۴: توابع read_point، distancd، surface و add_points را که در برنامه‌ی ۹-۳ نوشته شدند و در آن‌ها خود رکوردها به عنوان پارامتر تعریف شده اند، این بار طوری بازنویسی نمایید که به‌جای خود رکوردها آدرس آن‌ها به توابع مزبور ارسال گردد.

```
                                    /*  تابعی برای خواندن مختصات یک نقطه و تشخیص خاتمه‌ی داده‌های ورودی  */
struct point *read_point(struct point *p)
{
   printf("Enter the next point coordinates as x y:");
   if (scanf("%f%f", &p->x, &(*p).y) == EOF)
   {  p->x = p->y = ۰; /*  تشخیص خاتمه‌ی داده‌ها، ثبت صفر در مختصات نقطه‌ی جواب  */
      return NULL;          /*          و بازگرداندن اشاره‌گر تهی به‌عنوان نتیجه‌ی تابع  */
   }
   else
      return p;    /*  بازگرداندن آدرس رکوردی که مختصات خوانده شده در آن قرار داده شده است به عنوان نتیجه‌ی تابع  */
}
                                    /*  تابعی برای محاسبه‌ی فاصله‌ی بین دونقطه  */
float distance(struct point *first, struct point *second)
{  float distance;

   distance = sqrt(pow(first->x - second->x, ۲)+/*محاسبه‌ی فاصله در یک متغیر محلی*/
                   pow(first->y - second->y, ۲));
   return distance;          /*  بازگرداندن فاصله‌ی بین دو نقطه به عنوان نتیجه‌ی تابع  */
}
                                    /*  تابعی برای محاسبه‌ی مساحت مستطیل ساخته شده به‌وسیله‌ی دو نقطه  */
float surface(struct point *first, struct point *second)
{  float surface;
                                    /*  محاسبه‌ی مساحت مستطیل در یک متغیر محلی  */
   surface = abs((first->x - second->x) * (first->y - second->y));
   return surface;          /*  بازگرداندن مساحت مستطیل به عنوان نتیجه‌ی تابع  */
}
                                    /*  تابعی برای محاسبه‌ی حاصل‌جمع دو نقطه  */
void add_points(struct point *first, struct point *second,
                struct point *sum)
{                    /*  ذخیره‌ی حاصل‌جمع دو نقطه (پارامترهای اول و دوم) در پارامتر سوم که آدرس یک نقطه است  */
   sum->x = first->x + second->x;
   sum->y = first->y + second->y;
   return;
}
```

شکل ۹-۱۰ الف: متن توابع مربوط به برنامه‌ی ۹-۴ تعریف پارامتر از نوع آدرس رکورد و پردازش عضوها.

```
#include <stdio.h>
#include <math.h>
#define TRUE ۱                    /* وضعیت مبنی بر شرط همیشه درست */

struct point                      /* تعریف رکورد نقطه به‌صورت سراسری */
{  float  x;
   float  y;
};
          /* . . . . . محل قرار گرفتن چهار تابع نوشته شده در شکل ۹-۱۰ الف . . . . . */

main()
{  struct point p, q, sum;/* تعریف سه متغیر از نوع رکورد نقطه برای ذخیره‌ی مختصات دو نقطه و حاصل‌جمع آن‌ها */
   struct point *pp;              /* تعریف یک متغیر از نوع اشاره‌گر به رکورد نقطه برای ذخیره‌ی نتیجه‌ی تابع خواندن */
   int i, input_ok;
   float point_dist, rect_surface;

   while (TRUE)                   /* حلقه‌ی تکرار خواندن مختصات هر دفعت نقطه */
   {                        /* احضار تابع مربوطه برای خواندن مختصات نقطه‌ی اول و تخصیص نتیجه به یک متغیر از نوع رکورد نقطه */
      pp = read_point(&p);
      if (pp == NULL)             /* تشخیص خاتمه‌ی داده‌ها و ترک حلقه‌ی تکرار */
         break;
      pp = read_point(&q);/* احضار تابع مربوطه برای خواندن مختصات نقطه‌ی دوم */
      if (pp == NULL)             /* تشخیص خاتمه‌ی داده‌ها و ترک حلقه‌ی تکرار */
         break;
      point_dist = distance(&p, &q); /* احضار تابع محاسبه‌ی فاصله و ارسال آدرس دو نقطه به آن */
      rect_surface = surface(&p, &q);/* احضار تابع محاسبه‌ی مساحت و ارسال آدرس دو نقطه به آن */
                        /* احضار تابع محاسبه‌ی حاصل‌جمع و ارسال آدرس دو نقطه و آدرس نقطه‌ی حاصل‌جمع به آن */
      add_points(&p, &q, &sum);
      printf("The two points are: (%.۲f, %.۲f) and (%.۲f, %.۲f)\n",
             p.x, p.y, q.x, q.y);       /* چاپ نتایج */
      printf("Distance between these points is: %.۲f\n",point_dist);
      printf("Rectangle surface is: %.۲f\n", rect_surface);
      printf("The sum of two points is: (%.۲f, %.۲f)\n",
             sum.x, sum.y);
   }
   return ۰;
}
```

شکل ۹-۱۰ ب: متن تابع اصلی مربوط به برنامه‌ی ۴-۹ ارسال آدرس رکوردها به‌عنوان آرگومان به توابع.

۹-۲-۴ عملگرهای مرتبط با رکوردها

زیرنویس یا []، عضو رکورد یا .، دستیابی غیرمستقیم به عضو رکورد یا <- دارای بالاترین تقدم

عملگرهای دستیابی غیر مستقیم یا * و استخراج آدرس یا & دارای تقدم دوم

```
struct point
{  float x;
   float y;
};
struct line
{  struct point start;
   struct point end;
   char *name;
} a = {۱, ۱, ۱۰, ۲۰, "a b"};
struct line *pa, *pm,
       m[] = {{۲, ۳, ٤, ٥, "c d"},
              {٤, ٦, ۸, ۱, "m n"},
              {۸, ٥, ٤, ۲, "x y"}};
pa = &a;
pm = &m[۱];
```

نحوه‌ی مقداردهی اولیه‌ی آرایه‌ای از رکوردها

ترتیب اجرای دو عملگر نقطه و <- از چپ به راست است

۱- معادل بودن عبارات زیر

`a.start.x  pa->start.x  (a.start).x  (pa->start).x  (*pa).start.x`

۲- معادل بودن دو عبارت `m[۱].end.y` **و** `pm->end.y`

۳- معادل بودن دو عبارت `m[۰].start.y` **و** `(pm-۱)->start.y`

توجه به افزودن یا کاستن عدد به اشاره‌گری به خانه‌ای از یک آرایه با ساختمان رکورد

۴- عمل افزایش روی عضو x در هر دو عبارت `a.start.x++` **و** `++pa->start.x`

تقدم بالاتر عملگرهای نقطه و <-

۵- افزایش pm به اندازه‌ی یک رکورد و ذخیره ۵ در x مربوط به عضو start (از سطر دوم آرایه‌ی m)

در عبارت `(++pm)->start.x=۵`

۶- عبارت مزبور به شکل `(pm++)->start.x=۵`

ذخیره ۵ در x مربوط به رکوردی که آدرس آن در متغیر pm است (x از عضو start از سطر یکم آرایه‌ی m) و بعد افزایش pm به اندازه‌ی یک رکورد (حذف پرانتز نیز تأثیری ندارد)

۷- عبارت pa->name* کاراکتر مورد اشاره به‌وسیله‌ی name از رکورد مورد اشاره توسط pa

۸- pa->name مبین کل رشته‌ی مورد اشاره به‌وسیله‌ی name از رکورد مورد اشـــــاره توسط pa

۹- عبارت pa->name++*، دستیابی به محتویات کاراکتر مورد اشاره توسط name بـا خصوصـیات فـوق، افزودن یک واحد به اشاره‌گر موجود در آن، اشاره‌گر name عضو رکورد a (مورد اشاره توسط pa) حـاوی آدرس فاصله‌ی خالی بعد از حرف a و قبل از حرف b در رشته‌ی اولیه‌ی مورد اشـاره توسـط name اشـاره خواهد نمود (تقدم ++ در حالت پسوند بالاتر از تقدم * است یعنی روی pa->name عمل می‌کند ولـی اثـر آن بعد از عمل * ظاهر می‌شود).

۱۰- قرار دادن پرانتز ++(pa->name*) تـأثیر عمـل ++ روی حاصـل عبـارت داخـل زوج پرانتـز یعنـی کاراکتر مورد اشاره توسط عضو name از رکورد a (مورد اشاره توسط pa) و افزایش مقدار آن به انـدازه‌ی یک واحد (تبدیل حرف a به حرف b و درنتیجه رشته‌ی "a b" به "b b").

۱۱- عبارت pm++->name* دستیابی به محتویات name به شرح فوق، افزایش یک واحد به آدرس موجـود در اشاره‌گر pm (تقدم ++ در حالت پسـوند مسـاوی تقـدم -> و بـالاتر از تقـدم * و اثـر روی pm ولـی بعـد از عمل‌های * و -> نتیجه قابل استفاده است).

تعداد بایت مصرف شده برای رکورد point در عبارت sizeof(struct point)

استخراج اندازه‌ی رکورد line در هر دو عبارت sizeof(struct line) یا sizeof a

آرایه‌ی m که تعداد خانه‌هایش صریحاً مشخص نشده وعبارت sizeof(m)/sizeof(m[۰]) فرمانی به شکل زیر

```
#define MSIZE (sizeof m / sizeof m[۰])
```

از اول ۹-۲-۵ تعریف متغیرها براساس مجموعه‌ی ثابت‌های شمارشی

تا آخر ۹-۲-۷ تعریف متغیرها با تعداد بیت خاص برای مطالعه‌ی شخصی

۹-۲-۸ رکوردها و متغیرهای سراسری

- تعریف رکوردها در خارج از همه توابع برای ایجاد امکان استفاده از آن تعریف در توابع
- امکان تعریف خود رکوردها، آرایه ها و حتی متغیرهای عادی به صورت سراسری و در شروع فایل قبل از تعریف توابع
- امکان دسترسی کلیه توابع به متغیرهای سراسری از هر نوع بدون نیاز به ارسال به عنوان آرگومان
- برنامه نمونه ۹-۵: تعریف کلیه متغیرها به صورت سراسری، عدم تعریف پارامتر در توابع و عدم ارسال آرگومان در زمان احضار

```
void read_data()                          /* تابعی برای خواندن داده های دانشجویان و ذخیره در آرایه ای از رکوردها */
{  int i;
    scanf("%d", &nofst);
   for (i = ۰; i < nofst; i++)            /* حلقه تکرار خواندن داده های دانشجویان */
      scanf("٪۷۱d٪۲۰s٪۳۰s٪۵f"
            &std[i].std_numb, std[i].std_first,
            std[i].std_last, &std[i].gpa);
   return;
}

void sort_records()                       /* تابعی برای مرتب سازی رکوردهای دانشجویان، ذخیره شده در آرایه ای از رکوردها */
{  struct std_rec tmp;
    int i, j = ۰, sort_stat = !SORTED;

    While (sort_stat != SORTED)           /* حلقه های تکرار مرتب سازی رکوردها */
        for (j++, sort_stat = SORTED, i = ۰; i < nofst - j; i++)
            if (strcmp(std[i].std_last, std[i + ۱].std_last) > ۰)
            {   tmp = std[i];             /* جابه جا کردن دو رکورد متوالی */
                std[i] = std[i + ۱];
                std[i + ۱] = tmp;
                sort_stat = !SORTED;
            }
    return;
}

void print_report()                       /* تابعی برای چاپ گزارش از رکوردهای مرتب شده */
{   int i;

    memset(dash, '-', LINEWIDTH);         /* ساختن خط افقی در حافظه جهت چاپ در ای‌ست */
    dash[LINEWIDTH] = '\۰'
    for (i = ۰; i < nofst; i++)           /* حلقه‌ی تکرار چاپ داده های مرتب شده */

    {   if (!(i % LINEPPG))
        {   printf("\f\n    seq    Student_no First Name          ");
            Printf("    Last Name                                GPA\n");
            Printf("%s\n", dash);         /* چاپ خط افقی */

        }
        Printf('٪۵d ٪۷ld     ٪-۲۰s     ٪-۳۰ ٪۵.۲f\n\n",
            i + ۱, std[i].std_numb, std[i].std_first,
```

شکل ۹-۱۱:الف: متن برنامه‌ی۹-۵، توابع مربوط به پردازش رکوردهای دانشجویان با تعاریف سراسری

```
#include <stdio.h>
#include <string.h>
#define MAXNOS ۱۰۰                    /* حداکثر تعداد دانشجو */

#define LINEPPG ۲۵                    /* تعداد سطر مورد چاپ در یک صفحه */

#define SORTED ۱                      /* وضعیت مبنی بر مرتب بودن داده ها */

#define LINEWIDTH ۷۸                  /* عرض سطر مورد چاپ */


Struct std_rec                        /* تعریف ساختمان رکورد دانشجو */
{  long std_numb;
   char std_first[۲۱];
   char std_last[۳۱];
   float gpa;
};
Struct std_rec std[MAXNOS];           /* تعریف آرایه سراسری برای ذخیره و مرتب سازی داده ها */
int nofst;                            /* تعریف متغیر سراسری برای ذخیره تعداد رکوردهای خوانده شده */
char dash[LINEWIDTH + ۱]              /* تعریف آرایه‌ی حاوی خط افقی */

                                      /* .... محل قرارگرفتن سه تابع نوشته شده در شکل ۹-۱۱ الف .... */

main()
{
   read_data();                       /* احضار تابع برای خواندن داده ها به داخل آرایه ای از رکوردها */
   sort_records();                    /* احضار تابع برای مرتب سازی رکوردهای موجود در آرایه */
   print_report();                    /* احضار تابع برای چاپ محتویات رکوردهای مرتب شده */
   return ۰;
}
```

شکل ۹-۱۱:ب: متن برنامه‌ی۹-۵، تعاریف سراسری و تابع اصلی برای پردازش رکوردهای دانشجویان

۹-۲-۹ تلفیق رکوردها و آرایه ها

- **توجه به تعریف رکورد دانشگاه در شکل ۹-۵ به صورت تودرتو**

```
#define MAX_COURSES   ۱۰۰          /* تعریف حداکثر تعداد درس برای هر دانشجو */
#define MAX_STUDENTS  ۵۰           /* تعریف حداکثر تعداد دانشجو برای هر رشته */
#define MAX_MAJORS    ۲۰           /* تعریف حداکثر تعداد رشته در دانشگاه */

struct univ_rec                    /* تعریف رکورد حاوی داده‌های یک دانشگاه */
{  char university_name[۵۱];       /* نام دانشگاه */
   struct major_rec                /* تعریف رکورد رشته در داخل رکورد دانشگاه */
   {  char major_name[۲۱];         /* نام رشته‌ی تحصیلی آرایه‌ی ۲۱ کاراکتری */
      float major_gpa;             /* متوسط معدل دانشجویان رشته */
      struct std_rec               /* تعریف رکورد دانشجو در داخل رکورد رشته */
      {  long  std_number;         /* شماره‌ی دانشجویی */
         char  first_name[۲۱];     /* نام دانشجو */
         char  last_name[۳۱];      /* نام خانوادگی دانشجو */
         struct course_rec         /* تعریف رکورد درس در داخل رکورد دانشجو */
         {  long course_number;    /* شماره‌ی درس */
            char course_name[۱۶];  /* نام درس */
            char instructor[۲۱];   /* نام مدرس */
         } courses[MAX_COURSES];   /* تعریف آرایه‌ی حاوی ۱۰۰ رکورد درس در هر رکورد دانشجو
*/
         float gpa;                /* معدل کل دانشجو */
         int   units_passed;       /* تعداد واحد گذرانده */
      } students[MAX_STUDENTS];    /* تعریف آرایه‌ی حاوی ۵۰ رکورد دانشجو در هر رکورد رشته */
   } majors[MAX_MAJORS];           /* تعریف آرایه‌ی حاوی ۲۰ رکورد رشته در هر رکورد دانشگاه */
   float university_gpa;           /* متوسط معدل دانشجویان دانشگاه */
} university;                      /* تعریف نام رکورد دانشگاه */
```

شکل ۹-۵: نمونه‌ای از تعاریف تودرتو برای آرایه‌ای از رکوردها، رکورد در داخل رکورد و آرایه در داخل رکورد.

- **امکان تعریف رکوردها به صورت مجزا و استفاده از آن به صورت تودرتو**

```
#define MAX_COURSES   ۱۰۰      /* تعریف حداکثر تعداد درس برای هر دانشجو */
#define MAX_STUDENTS  ۵۰       /* تعریف حداکثر تعداد دانشجو برای هر رشته */
#define MAX_MAJORS    ۲۰       /* تعریف حداکثر تعداد رشته در دانشگاه */

struct course_rec                          /* تعریف رکورد درس به صورت جداگانه */
{  long course_number;                     /* شماره درس */
   char course_name[۱۶];                   /* نام درس */
   char instructor[۲۱];                    /* نام مدرس */
};

Struct std_rec                             /* تعریف رکورد دانشجو به صورت جداگانه */
{  long   std_number;                      /* شماره‌ی دانشجویی */
   char   first_name[۲۱];                  /* نام دانشجو */
   char   last_name[۳۱];                   /* نام خانوادگی دانشجو */
   struct course_rec courses[MAX_COURSES]; /* آرایه حاوی ۱۰۰ رکورد درس در رکورد دانشجو */
   float  gpa;                             /* معدل کل دانشجو */
   int    passed_units;                    /* تعداد واحد گذارنده */
};

struct major_rec                           /* تعریف رکورد درس به صورت جداگانه */
{  char major_name[۲۱];                    /* نام رشته تحصیلی آرایه‌ی ۲۱ کاراکتری */
   float major_gpa;                        /* متوسط معدل دانشجویان رشته */
   struct std_rec students[MAX_STUDENTS];  /* آرایه ۵۰ رکوردی دانشجو در رکورد رشته */
};

Struct univ_rec                            /* تعریف رکورد حاوی داده های یک دانشگاه */
{  char   university_name[۵۱];             /* نام دانشگاه */
   struct major_rec majors[MAX_MAJORS];    /* آرایه حاوی ۲۰ رکورد رشته در رکورد دانشگاه */
   float  university_gpa;                  /* متوسط معدل دانشجویان دانشگاه */
}  university;                             /* تعریف نام رکورد دانشگاه */
```

شکل ۹-۶: نمونه‌ای از تعاریف تودرتو برای آرایه‌ای از رکوردها، رکورد تودرتو و آرایه در داخل رکورد.