Linux C 7 结构体

Subtitle

2022/10/05

Table of Contents

Linux C 7 结构体

数据抽象

复数运算

1、在本节的基础上实现一个打印复数的函数，打印的格式是x+yi，如果实部或虚部为0则省略，例如：1.0、-2.0i□-1.0+2.0i□1.0-2.0i。最后编写一个main函数测试本节的所有代码。想一想这个打印函数应该属于上图中的哪一层？

complex.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

struct complex_struct {
  double r, A;
};

double real_part(struct complex_struct z)
{
  return z.r * cos(z.A);
}

double img_part(struct complex_struct z)
{
  return z.r * sin(z.A);
}

double magnitude(struct complex_struct z)
{
  return z.r;
}

double angle(struct complex_struct z)
{
  return z.A;
}

struct complex_struct make_from_real_img(double x, double y)
{
  struct complex_struct z;
   z.A = atan2(y, x);
   z.r = sqrt(x * x + y * y);
  return z;
}

struct complex_struct make_from_mag_ang(double r, double A)
{
  struct complex_struct z;
   z.r = r;
```

```
    z.A = A;
  return z;
}

struct complex_struct add_complex(struct complex_struct z1, struct complex_struct z2)
{
  return make_from_real_img(real_part(z1) + real_part(z2),
              img_part(z1) + img_part(z2));
}

struct complex_struct sub_complex(struct complex_struct z1, struct complex_struct z2)
{
  return make_from_real_img(real_part(z1) - real_part(z2),
              img_part(z1) - img_part(z2));
}

struct complex_struct mul_complex(struct complex_struct z1, struct complex_struct z2)
{
  return make_from_mag_ang(magnitude(z1) * magnitude(z2),
              angle(z1) + angle(z2));
}

struct complex_struct div_complex(struct complex_struct z1, struct complex_struct z2)
{
  return make_from_mag_ang(magnitude(z1) / magnitude(z2),
              angle(z1) - angle(z2));
}

//打印复数 属于复数运算层，调用复数表示层
void printf_complex(struct complex_struct z)
{
  if(abs(real_part(z))<0.1&&abs(img_part(z))>=0.1)  //浮点型不能直接判断是否为0，这里因为输出只保留小数点后1位，所以认为小于0.1的就为0
      printf("%.1fi\n",img_part(z));
  else if(abs(real_part(z))>=0.1&&abs(img_part(z))<0.1)
      printf("%.1f\n",real_part(z));
  else if(abs(real_part(z))<0.1&&abs(img_part(z))<0.1)
      printf("0\n");
  else
    printf("%.1f+%.1fi\n",real_part(z),img_part(z));
}
int main(int argc, char* argv[])
{
  struct complex_struct z1=make_from_real_img(0,2);
  struct complex_struct z2=make_from_real_img(0,1);
  printf("z1+z2=");   printf_complex(add_complex(z1,z2));
  printf("z1-z2=");   printf_complex(sub_complex(z1,z2));
  printf("z1*z2=");   printf_complex(mul_complex(z1,z2));
  printf("z1/z2=");   printf_complex(div_complex(z1,z2));

  system("pause");
}
```

测试输出：

```
z1+z2=4.0+3.0i
z1-z2=-2.0+1.0i
z1*z2=1.0+7.0i
z1/z2=0.5+0.5i
```

分数运算

2、实现一个用分子分母的格式来表示有理数的结构体rational以及相关的函数，rational结构体之间可以做加减乘除运算，运算的结果仍然是rational。测试代码如下：

```
int main(void)
{
    struct rational a = make_rational(1, 8); /* a=1/8 */
    struct rational b = make_rational(-1, 8); /* b=-1/8 */
    print_rational(add_rational(a, b));
    print_rational(sub_rational(a, b));
    print_rational(mul_rational(a, b));
    print_rational(div_rational(a, b));

    return 0;
}
```

注意要约分为最简分数，例如1/8和-1/8相减的打印结果应该是1/4而不是2/8，可以利用第 3 节“ 递归 ”练习题中的Euclid算法来约分。在动手编程之前先思考一下这个问题实现了什么样的数据抽象，抽象层应该由哪些函数组成。

rational.c

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

struct rational {
  int a,b;
};

struct rational make_rational(int a,int b)
{
  struct rational z;
   z.a=a;
   z.b=b;
  return z;
}

struct rational add_rational(struct rational a,struct rational b)
{
  struct rational c;
   c.a=a.a*b.b+a.b*b.a;
   c.b=a.b*b.b;
  return c;
}

struct rational sub_rational(struct rational a,struct rational b)
```

```
{
  struct rational c;
   c.a=a.a*b.b-a.b*b.a;
   c.b=a.b*b.b;
  return c;
}

struct rational mul_rational(struct rational a,struct rational b)
{
  struct rational c;
   c.a=a.a*b.a;
   c.b=a.b*b.b;
  return c;
}

struct rational div_rational(struct rational a,struct rational b)
{
  struct rational c;
   c.a=a.a*b.b;
   c.b=a.b*b.a;
  return c;
}
int gcd (int a,int b)
{
  if(b==0)
    return abs(a);
  else
    return gcd(b,a%b);
}

void print_rational(struct rational a)
{
  int ra_gcd=gcd(a.a,a.b);
   a.a=a.a/ra_gcd;
   a.b=a.b/ra_gcd;
  if(a.a==0)
    printf("0\n");
  else if(abs(a.b)==1)
    printf("%d\n",a.a/a.b);
  else
    printf("%d/%d\n",a.a,a.b);
}
int main(int argc, char* argv[])
{
  struct rational a = make_rational(-1, 8);/* a=1/8 */
  struct rational b = make_rational(-1, 8);/* b=-1/8 */
   print_rational(add_rational(a, b));
   print_rational(sub_rational(a, b));
   print_rational(mul_rational(a, b));
   print_rational(div_rational(a, b));
  return 0;
  system("pause");
```

```
}
```

数据类型标志

1、本节只给出了make*from*real*img*和make*from*mag*ang函数的实现，请读者自己实现real*part*、*img*part、magnitude、angle这些函数。

2、编译运行下面这段程序：

```c
#include <stdio.h>

enum coordinate_type { RECTANGULAR = 1, POLAR };

int main(void)
{
  int RECTANGULAR;
  printf("%d %d\n", RECTANGULAR, POLAR);
  return 0;
}
```

结果是什么？并解释一下为什么是这样的结果。

> 结果是 8 2.
> 枚举的成员名和变量名在同一命名空间中，会出现命名冲突。

Printed on: 2022/10/05 17:12