Com S // Cpr E // Math 5250

Numerical Analysis of High-Performance Computing

Instructor: Songting Luo

Lecture 10: Review of Pointers, Structs in C, Header Files & Compiling Multiple Files

Outline

1. Quick Review of Pointers in C
2. Structs in C
3. Header Files & Compiling Multiple Files into a Single Executable

Quick Review of Pointers in C

Ints/Floats/Chars are passed by value

$ISUHPC/lectures/lecture10/codes/intfunc.c:

```
1  int main()
2  {
3     int a =  2;
4     int b = -5;
5
6     printf("\n Before function call:\n");
7     printf("%3i %3i \n",a,b);
8
9     void somefunc(int a, int b);
10    somefunc(a,b);
11
12    printf("\n After function call:\n");
13    printf("%3i %3i \n\n",a,b);
14 }
15
16 void somefunc(int a, int b)
17 {
18    a = -7; b = 11;
19
20    printf("\n During function call:\n");
21    printf("%3i %3i \n",a,b);
22 }
```

Ints/Floats/Chars are passed by value

```
$ gcc intfunc.c
$ ./a.out

 Before function call:
  2  -5

 During function call:
 -7  11

 After function call:
  2  -5
```

Arrays are passed by reference

$ISUHPC/lectures/lecture10/codes/arrfunc.c:

```
1  int main()
2  {
3     int somearray[] = {-1,-2,-3,0,2,4,6};
4
5     printf("\n Before function call:\n");
6     for (int i=0; i<7; i++)
7     {
8        if (i>0){ printf(","); }
9        printf(" %3i",somearray[i]);
10    }
11    printf("\n");
12
13    void somefunc(int somearray[]);
14    somefunc(somearray);
15
16    printf("\n After function call:\n");
17    for (int i=0; i<7; i++)
18    {
19       if (i>0){ printf(","); }
20       printf(" %3i",somearray[i]);
21    }
22    printf("\n\n");
23 }
```

Arrays are passed by reference

$ISUHPC/lectures/lecture10/codes/arrfunc.c:

```
1  void somefunc(int somearray[])
2  {
3     for (int i=0; i<7; i++)
4     {
5        somearray[i] = -2*somearray[i];
6     }
7  }
```

```
$ gcc arrfunc.c
$ ./a.out

 Before function call:
  -1,  -2,  -3,   0,   2,   4,   6

 After function call:
   2,   4,   6,   0,  -4,  -8, -12
```

Memory address

Every declared variable in C is stored somewhere in memory. You can use the & (ampersand) operator to obtain the **memory address** of that variable.

$ISUHPC/lectures/lecture10/codes/simple1.c:

```c
#include <stdio.h>

int main ()
{
    int  var1;
    char var2[10];

    printf("Address of var1 variable: %p\n", &var1 );
    printf("Address of var2 variable: %p\n", &var2 );

    return 0;
}
```

```
$ gcc simple1.c
$ ./a.out
Address of var1 variable: 0x7fff55da2784
Address of var2 variable: 0x7fff55da278e
```

Pointers

A **pointer** is a variable whose value is the address of another variable, i.e., direct address of the memory location.

$ISUHPC/lectures/lecture10/codes/simple2.c:

```c
int main ()
{
   int  var = -11;  /* actual variable declaration */
   int  *ip;        /* pointer variable declaration */
   ip = &var;  /* store address of var in pointer variable*/

   printf("       Address of var variable: %p\n", &var );
   printf(" Address stored in ip variable: %p\n", ip   );
   printf("         Value of *ip variable: %d\n", *ip  );

   return 0;
}
```

```
$ gcc simple2.c
$ ./a.out
       Address of var variable: 0x7fff5b09a798
 Address stored in ip variable: 0x7fff5b09a798
         Value of *ip variable: -11
```

Passing Ints/Floats/Chars by reference

$ISUHPC/lectures/lecture10/codes/intfunc_byref.c:

```
1  int main()
2  {
3     int a =  2;
4     int b = -5;
5
6     printf("\n Before function call:\n");
7     printf("%3i %3i \n",a,b);
8
9     void somefunc(int* a, int* b);
10    somefunc(&a,&b);
11
12    printf("\n After function call:\n");
13    printf("%3i %3i \n\n",a,b);
14 }
15
16 void somefunc(int* a, int* b)
17 {
18    *a = -7; *b = 11;
19
20    printf("\n During function call:\n");
21    printf("%3i %3i \n",*a,*b);
22 }
```

Passing Ints/Floats/Chars by reference

NOTE: To change the value of the variable to which the pointer is pointing to, you need to **dereference** the pointer and then treat as a regular variable:

```
1     *a = -7; *b = 11;
2
3     printf("\n During function call:\n");
4     printf("%3i %3i \n",*a,*b);
```

```
$ gcc intfunc_byref.c
$ ./a.out

 Before function call:
  2   -5

 During function call:
 -7   11

 After function call:
 -7   11
```

Structs in C

Structs in C

struct

A `struct` in the C programming language is a complex data type declaration that defines a physically grouped list of variables to be placed under one name in a block of memory, allowing the different variables to be accessed via a single pointer.

Some examples of possible structs:

```c
struct employee
{
    int age;
    int salary;
    char name[12];
    char address[6][20];
};
```

```c
struct fraction
{
    int numer;
    int denom;
};
```

Example: points and triangles

$ISUHPC/lectures/lecture10/codes/triangle.c:

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <math.h>
4
5  typedef struct point point;
6  struct point
7  {
8     double x;
9     double y;
10 };
11
12 typedef struct triangle triangle;
13 struct triangle
14 {
15    point node1;
16    point node2;
17    point node3;
18    double area;
19 };
```

Example: points and triangles

$ISUHPC/lectures/lecture10/codes/triangle.c:

```
1  int main()
2  {
3     triangle t;
4     t.node1.x =  0.0;  t.node1.y =  0.0;
5     t.node2.x = -0.5;  t.node2.y =  0.5;
6     t.node3.x = -1.0;  t.node3.y =  2.5;
7
8     void compute_area(triangle* tri);
9     compute_area(&t);
10
11    printf("\n  triangle node #1: ( %12.5e, %12.5e )",t.node1.x
          ,t.node1.y);
12    printf("\n  triangle node #2: ( %12.5e, %12.5e )",t.node2.x
          ,t.node2.y);
```

Example: points and triangles

$ISUHPC/lectures/lecture10/codes/triangle.c:

```
    printf("\n  triangle node #3: ( %12.5e, %12.5e )",t.node3.x
        ,t.node3.y);
    printf("\n  area = %12.5e\n\n",t.area);
}

void compute_area(triangle* tri)
{
    double u[3],v[3],w[3];
    u[0] = tri->node3.x - tri->node1.x;
    u[1] = tri->node3.y - tri->node1.y;
    u[2] = 0.0;

    v[0] = tri->node2.x - tri->node1.x;
    v[1] = tri->node2.y - tri->node1.y;
    v[2] = 0.0;

    void cross_prod(double u[], double v[], double w[]);
```

Example: points and triangles

$ISUHPC/lectures/lecture10/codes/triangle.c:

```
    tri->area = 0.5*fabs(w[2]);
}

void cross_prod(double u[], double v[], double w[])
{
    w[0] = u[1]*v[2] - u[2]*v[1];
    w[1] = u[2]*v[0] - u[0]*v[2];
    w[2] = u[0]*v[1] - u[1]*v[0];
}
```

```
$ gcc triangle.c
$ ./a.out

  triangle node #1: (  0.00000e+00,  0.00000e+00 )
  triangle node #2: ( -5.00000e-01,  5.00000e-01 )
  triangle node #3: ( -1.00000e+00,  2.50000e+00 )
  area =  3.75000e-01
```

Example: fractions

$ISUHPC/lectures/lecture10/codes/fraction.c:

```
1  typedef struct fraction fraction;
2  struct fraction
3  {
4     int numer; int denom;
5  };
6
7  int main()
8  {
9     fraction a,b,sum;
10
11    a.numer = -1; a.denom = 10;
12    b.numer = -3; b.denom = 8;
13
14    void fraction_add(const fraction* a,
15                      const fraction* b,
16                      fraction* sum);
17    fraction_add(&a,&b,&sum);
18
19    printf("\n %i/%i  +  %i/%i  =  %i/%i\n\n",
20           a.numer,a.denom,b.numer,b.denom,
21           sum.numer,sum.denom);
22 }
```

Example: fractions

$ISUHPC/lectures/lecture10/codes/fraction.c:

```
1  void fraction_add(const fraction* a,
2                    const fraction* b,
3                    fraction* sum)
4  {
5      sum->denom = a->denom * b->denom;
6      sum->numer = a->numer * b->denom + b->numer * a->denom;
7
8      void fraction_reduce(fraction* sum);
9      fraction_reduce(sum);
10 }
```

Example: fractions

$ISUHPC/lectures/lecture10/codes/fraction.c:

```
1  void fraction_reduce(fraction* sum)
2  {
3     void get_prime_factors(int n,
4                            int prime_list[],
5                            int* num_primes);
6
7     if ( (sum->numer < 0) && (sum->denom < 0) )
8     { sum->numer = abs(sum->numer);
9       sum->denom = abs(sum->denom); }
10
11    int prime1[100]; int num_prime_1;
12    int msign1 = 1; if (sum->numer < 0) { msign1 = -1; }
13    sum->numer = abs(sum->numer);
14    get_prime_factors(sum->numer,prime1,&num_prime_1);
15
16    int prime2[100]; int num_prime_2;
17    int msign2 = 1; if (sum->denom < 0) { msign2 = -1; }
18    sum->denom = abs(sum->denom);
19    get_prime_factors(sum->denom,prime2,&num_prime_2);
20
21    int  i = 0; int  j = 0;
22    int z1 = prime1[i]; int z2 = prime2[j];
```

Example: fractions

$ISUHPC/lectures/lecture10/codes/fraction.c:

```
    while(i<num_prime_1 && j<num_prime_2)
    {
       if (z1==z2)
       {
          sum->numer = sum->numer/z1;
          sum->denom = sum->denom/z2;

          i  = i+1;
          j  = j+1;
          z1 = prime1[i];
          z2 = prime2[j];
       }
       else
       {
          if (z1>z2)
          {
             j = j+1;
             z2 = prime2[j];
          }
```

Example: fractions

$ISUHPC/lectures/lecture10/codes/fraction.c:

```
            else
            {
                i = i+1;
                z1 = prime1[i];
            }
        }
    }

    sum->numer = sum->numer*msign1;
    sum->denom = sum->denom*msign2;
}
```

Example: fractions

$ISUHPC/lectures/lecture10/codes/fraction.c:

```
1  void get_prime_factors(int n,
2                         int prime_list[],
3                         int* num_primes)
4  {
5      *num_primes = 0;
6
7      while (n%2==0)
8      {
9          prime_list[*num_primes] = 2;
10         *num_primes = *num_primes + 1;
11         n = n/2;
12     }
13
14     for (int i=3; i<=sqrt(n); i=i+2)
15     {
16         while (n%i==0)
17         {
18             prime_list[*num_primes] = i;
19             *num_primes = *num_primes + 1;
20             n = n/i;
21         }
22     }
```

Example: fractions

$ISUHPC/lectures/lecture10/codes/fraction.c:

```
1
2       if (n>2)
3       {
4           prime_list[*num_primes] = n;
5           *num_primes = *num_primes + 1;
6       }
7   }
```

```
$ gcc fraction.c

$ ./a.out

 1/10  +  3/8  =  19/40
```

Header Files & Compiling Multiple Files into a Single Executable

Header files

Header files (*.h)

A header file is a file with extension `.h` that contains C function declarations and macro definitions and to be shared between several source files.

`$ISUHPC/lectures/lecture10/codes/fraction/fraction.h`:

```
1  #ifndef __FRACTION_H__
2  #define __FRACTION_H__
3
4  typedef struct fraction fraction;
5  struct fraction
6  {
7      int numer;
8      int denom;
9  };
10
11 #endif
```

NOTE: the wrapper `#ifndef ... #endif` makes sure that fraction only gets defined once.

Multiple files # 1

$ISUHPC/lectures/lecture10/codes/fraction/main.c:

```
1  #include <stdio.h>
2  #include "fraction.h"
3
4  int main()
5  {
6     printf("\n I am in the **main** function.\n");
7
8     fraction a,b,sum;
9
10    a.numer = -1; a.denom = 10;
11    b.numer = -3; b.denom = 8;
12
13    void fraction_add(const fraction* a,
14                      const fraction* b,
15                      fraction* sum);
16    fraction_add(&a,&b,&sum);
17
18    printf("\n %i/%i  +  %i/%i  =  %i/%i\n\n",
19           a.numer,a.denom,b.numer,b.denom,
20           sum.numer,sum.denom);
21 }
```

Multiple files # 2

$ISUHPC/lectures/lecture10/codes/fraction/fraction_add.c:

```
1  #include <stdio.h>
2  #include "fraction.h"
3
4  void fraction_add(const fraction* a,
5                    const fraction* b,
6                    fraction* sum)
7  {
8     printf(" I am in the **fraction_add** function.\n");
9
10    sum->denom = a->denom * b->denom;
11    sum->numer = a->numer * b->denom + b->numer * a->denom;
12
13    void fraction_reduce(fraction* sum);
14    fraction_reduce(sum);
15 }
```

Multiple files # 3

$ISUHPC/lectures/lecture10/codes/fraction/fraction_reduce.c:

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include "fraction.h"
4
5  void fraction_reduce(fraction* sum)
6  {
7     printf(" I am in the **fraction_reduce** function.\n");
8
9     void get_prime_factors(int n,
10                           int prime_list[],
11                           int* num_primes);
12
13    int prime1[100]; int num_prime_1;
14    int msign1 = 1; if (sum->numer < 0) { msign1 = -1; }
15    sum->numer = abs(sum->numer);
16    get_prime_factors(abs(sum->numer),prime1,&num_prime_1);
17
18    int prime2[100]; int num_prime_2;
19    int msign2 = 1; if (sum->denom < 0) { msign2 = -1; }
20    sum->denom = abs(sum->denom);
21    get_prime_factors(abs(sum->denom),prime2,&num_prime_2);
```

Multiple files # 3

$ISUHPC/lectures/lecture10/codes/fraction/fraction_reduce.c:

```
1     int  i = 0; int  j = 0;
2     int z1 = prime1[i]; int z2 = prime2[j];
3     while(i<num_prime_1 && j<num_prime_2)
4     {
5        if (z1==z2)
6        {
7           sum->numer = sum->numer/z1;
8           sum->denom = sum->denom/z2;
9
10          i  = i+1; j  = j+1;
11          z1 = prime1[i]; z2 = prime2[j];
12       }
13       else
14       {
15          if (z1>z2)
16          {  j = j+1; z2 = prime2[j];  }
17          else
18          {  i = i+1; z1 = prime1[i];  }
19       }
20    }
21    sum->numer = sum->numer*msign1;
22    sum->denom = sum->denom*msign2;
23 }
```

Multiple files # 4

$ISUHPC/lectures/lecture10/codes/fraction/get_prime_factors.c:

```
1  #include <stdio.h>
2  #include <math.h>
3
4  void get_prime_factors(int n, int prm_list[], int* num_prms)
5  {
6     printf(" I am in the **get_prime_factors** function.\n");
7     *num_prms = 0;
8
9     while (n%2==0)
10    { prm_list[*num_prms] = 2; *num_prms=*num_prms+1; n=n/2; }
11
12    for (int i=3; i<=sqrt(n); i=i+2)
13    {
14       while (n%i==0)
15       {
16          prm_list[*num_prms] = i;
17          *num_prms = *num_prms + 1;
18          n = n/i;
19       }
20    }
21    if (n>2)
22    { prm_list[*num_prms] = n; *num_prms = *num_prms + 1; }
23 }
```

Compiling multiple functions

Compile all four files and link together into single executable:

```
$ gcc fraction_add.c fraction_reduce.c \
get_prime_factors.c main.c \
-o main.exe
```

Run the executable:

```
$ ./main.exe

 I am in the **main** function.
 I am in the **fraction_add** function.
 I am in the **fraction_reduce** function.
 I am in the **get_prime_factors** function.
 I am in the **get_prime_factors** function.

 -1/10  +  -3/8  =  -19/40
```

Compiling multiple functions

Can split into separate compile ...:

```
$ gcc -c fraction_add.c fraction_reduce.c \
get_prime_factors.c main.c

$ ls *.o
fraction_add.o   fraction_reduce.o   get_prime_factors.o   main.o
```

... and link steps:

```
$ gcc fraction_add.o fraction_reduce.o get_prime_factors.o main.o \
-o main.exe

$ ./main.exe > output.txt
```

NOTE: redirected output to a text file.

Compiling multiple functions

Advantage: If we modify `fraction_add.c` to print "Luke, I am your father!", we only need to recompile this piece.

```
$ gcc -c fraction_add.c

$ gcc fraction_add.o fraction_reduce.o get_prime_factors.o main.o \
-o main.exe

$ main.exe

 I am in the **main** function.
 Luke, I am your father!
 I am in the **fraction_reduce** function.
 I am in the **get_prime_factors** function.
 I am in the **get_prime_factors** function.

 -1/10  +  -3/8  =  -19/40
```

When working on a big code (e.g., 100,000 lines split between 200 files) this can make a big difference!

Compiling multiple functions

Advantage: If we modify `fraction_add.c` to print "Luke, I am your father!", we only need to recompile this piece.

```
$ gcc -c fraction_add.c

$ gcc fraction_add.o fraction_reduce.o get_prime_factors.o main.o \
-o main.exe

$ main.exe

 I am in the **main** function.
 Luke, I am your father!
 I am in the **fraction_reduce** function.
 I am in the **get_prime_factors** function.
 I am in the **get_prime_factors** function.

 -1/10  +  -3/8  =  -19/40
```

When working on a big code (e.g., 100,000 lines split between 200 files) this can make a big difference!

Use of Makefiles greatly simplifies this.

Lab assignment: Computing info of a quadrilateral

Develop source codes **main.c** (with **main()**) for computing area, perimeter and inner angles of an input quadrilateral, use/declare **struct** in a header file, define functions for computing area, perimeter and inner angles in separated files (**quad_perimeter.c**, **quad_area.c**, and **quad_angles.c**), and call them in **main()**.

1. Try compile multiple files together to single executable (main.exe)
2. Try compile multiple files to Objects (*.o), and link them to single executable (main.exe)
3. Update Git Repository
4. submit source codes and screenshots.