Today:

Functions

1

Some General Tips on Programming

- Write your code modularly
- Compile + test functionality in the process

2

Some General Tips on Programming (Cont.)

- Use Debugger or printf to follow your execution flow and find what went wrong
- Understanding is good but not enough – you must practice!

3

Example: Factorial

```
#include <stdio.h>

int factorial(int n)
{
    int i, fact = 1;

    for (i=2; i<=n; i++)
      fact *= i;

    return fact;
}
```

```
int main(void)
{
   int num;

   printf("enter a number\n");
   scanf("%d",&num);

   printf("%d!=%d\n",num,factorial(num));

   return 0;
}
```

4

Exercise: Fibonacci (@class)

- Input:
  - An integer n
- Output:
  - The n'th fibonacci number
- Note –
  - Use an appropriately defined function

5

Solution

fibonacci_func.c

6

A Detailed Example

Write a program that receives a nominator and a denominator from the user, and displays the reduced form of the number.

For example, if the input is 6 and 9, the program should display 2/3.

7

Example – solution (step I)

```
#include <stdio.h>

int main(void)
{
    int n, d;

    printf("Please enter nominator and denominator: ");
    scanf("%d%d", &n, &d);

    Calculate n's and d's Greatest Common Divisor

    printf("The reduced form of %d/%d is %d/%d", n, d, n/gcd, d/gcd);

    return 0;
}
```

8

Example – solution (step II)

```
#include <stdio.h>

int main(void)
{
    int n, d, g;

    printf("Please enter nominator and denominator: ");
    scanf("%d%d", &n, &d);

    g = gcd(n, d);

    printf("The reduced form of %d/%d is %d/%d", n, d, n/g, d/g);

    return 0;
}
```

9

Example – solution (step III)

```
/* Returns the greatest common divisor of its
   two parameters. Assumes both are positive.
   */
int gcd(int x, int y)
{
   int i;

   for (i=x; i>0; i--)
        if (x%i == 0 && y%i ==0)
                return i;
}
```

10

GCD – step by step

```
int main(void)
{
    int n, d, g;

    printf("Please enter … : ");
    scanf("%d%d", &n, &d);

    g = gcd(n, d);

    printf("The reduced form…", n, d, n/g, d/g);

    return 0;
}
```

n	d	g
6	9	---

11

GCD – step by step

```
int main(void)
{
    int n, d, g;

    printf("Please enter … : ");
    scanf("%d%d", &n, &d);

    g = gcd(n, d);

    printf("The reduced form…", n, d, n/g, d/g);

    return 0;
}
```

n	d	g
6	9	---

12

GCD – step by step

```
int gcd(int x, int y)
{
  int i;

  for (i=x; i>0; i--)
      if (x%i == 0 && y%i ==0)
            return i;
}
```

x: 6 y: 9 i: ---

13

GCD – step by step

```
int gcd(int x, int y)
{
  int i;

  for (i=x; i>0; i--)
      if (x%i == 0 && y%i ==0)
            return i;
}
```

x: 6 y: 9 i: 6

14

GCD – step by step

```
int gcd(int x, int y)
{
  int i;

  for (i=x; i>0; i--)
      if (x%i == 0 && y%i ==0)
            return i;
}
```

x: 6 y: 9 i: 6

15

GCD – step by step

```
int gcd(int x, int y)
{
  int i;

  for (i=x; i>0; i--)
      if (x%i == 0 && y%i ==0)
            return i;
}
```

x: 6 y: 9 i: 5

16

GCD – step by step

```
int gcd(int x, int y)
{
  int i;

  for (i=x; i>0; i--)
      if (x%i == 0 && y%i ==0)
            return i;
}
```

x: 6 y: 9 i: 5

17

GCD – step by step

```
int gcd(int x, int y)
{
  int i;

  for (i=x; i>0; i--)
      if (x%i == 0 && y%i ==0)
            return i;
}
```

x: 6 y: 9 i: 4

18

GCD – step by step

```
int gcd(int x, int y)
{
  int i;

  for (i=x; i>0; i--)
      if (x%i == 0 && y%i ==0)
            return i;
}
```

x	y	i
6	9	4

19

GCD – step by step

```
int gcd(int x, int y)
{
  int i;

  for (i=x; i>0; i--)
      if (x%i == 0 && y%i ==0)
            return i;
}
```

x	y	i
6	9	3

20

GCD – step by step

```
int gcd(int x, int y)
{
  int i;

  for (i=x; i>0; i--)
      if (x%i == 0 && y%i ==0)
            return i;
}
```

x	y	i
6	9	3

21

GCD – step by step

```
int gcd(int x, int y)
{
  int i;

  for (i=x; i>0; i--)
      if (x%i == 0 && y%i ==0)
            return i;
}
```

x	y	i
6	9	3

22

GCD – step by step

```
int main(void)
{
   int n, d, g;

   printf("Please enter … : ");
   scanf("%d%d", &n, &d);

   g = gcd(n, d);

   printf("The reduced form…", n, d, n/g, d/g);

   return 0;
}
```

n	d	g
6	9	3

23

Example

- Newton was the first to notice that for any positive n, and when x_0=1, the following series converges to sqrt(n) –

$$x_{k+1} = \frac{1}{2}\left(x_k + \frac{n}{x_k}\right)$$

- Use this fact to write a program that accepts a positive number and outputs its square root
  - Hint – the thousandth element of Newton's series is a good-enough approximation

24

sqrt - solution

```
/* This program uses the Newton method to calculate a number's
   square root */
#include <stdio.h>

double sqrt(double num) {
    int i;
    double result = 1;

    for (i=0; i<1000; i++)
          result = (result + num/result)/2;

    return result;
}
```

25

sqrt – solution cont.

```
int main () {
    double num;

    printf("Please enter positive number: ");
    scanf("%lf", &num);

    if (num < 0) {
          printf("There's no square root for a negative number!\n");
          return 1;
    }
  printf("The square root of %g is %g\n", num, sqrt(num));
  return 0;
}
```

26

Pass-By-Value: Example

```
int add_one(int b)
{
    b=b+1;
    return b;
}

int main(void)
{
    int a=34,b=1;

    a=add_one(b);

    printf("a = %d, b = %d\n", a, b);
    return 0;
}
```

27

add_one – step by step

```
int add_one(int b)
{
    b=b+1;
    return b;
}

int main(void)
{
    int a=34,b=1;

    a=add_one(b);

    printf("a = %d, b = %d\n", a, b);
    return 0;
}
```

Main() memory state

a	b
34	1

28

add_one – step by step

```
int add_one(int b)
{
    b=b+1;
    return b;
}

int main(void)
{
    int a=34,b=1;

    a=add_one(b);

    printf("a = %d, b = %d\n", a, b);
    return 0;
}
```

Main() memory state

a	b
34	1

29

add_one – step by step

```
int add_one(int b)
{
    b=b+1;
    return b;
}

int main(void)
{
    int a=34,b=1;

    a=add_one(b);

    printf("a = %d, b = %d\n", a, b);
    return 0;
}
```

Main() memory state

a	b
34	1

add_one memory state

b
1

30

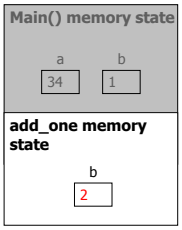
add_one – step by step

```
int add_one(int b)
{
   b=b+1;
   return b;
}

int main(void)
{
   int a=34,b=1;

   a=add_one(b);

   printf("a = %d, b = %d\n", a, b);
   return 0;
}
```



31

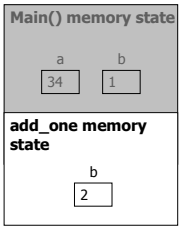
add_one – step by step

```
int add_one(int b)
{
   b=b+1;
   return b;
}

int main(void)
{
   int a=34,b=1;

   a=add_one(b);

   printf("a = %d, b = %d\n", a, b);
   return 0;
}
```



32

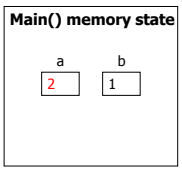
add_one – step by step

```
int add_one(int b)
{
   b=b+1;
   return b;
}

int main(void)
{
   int a=34,b=1;

   a=add_one(b);

   printf("a = %d, b = %d\n", a, b);
   return 0;
}
```



33

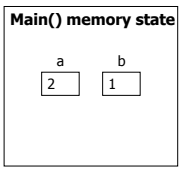
add_one – step by step

```
int add_one(int b)
{
   b=b+1;
   return b;
}

int main(void)
{
   int a=34,b=1;

   a=add_one(b);

   printf("a = %d, b = %d\n", a, b);
   return 0;
}
```



34

Example: Error in Scope

```
int add_one(int b)
{
   a=b+1;
   return a;
}

int main(void)
{
   int a=34,b=1;

   add_one(b);

   printf("a = %d, b = %d\n", a, b);
   return 0;
}
```

35

Example

Write a function that uses the formula

$\prod^2/6=1/1+1/4+1/9+1/16+\ldots+1/n^2$
(where n goes to infinity)

in order to approximate $\prod$. The function should accept an argument n which determines the number of terms in the formula. It should return the approximation of $\prod$.

Write a program that gets an integer n from the user, and approximate $\prod$ using n terms of the above formula.

36

pi - solution

```
#include <math.h>

/* Returns an approximation of pi using num terms */
double pi(int num) {
  int n;
  double result = 0.0;
  for (n = 1; n <= num; n++)
    result += 1.0 / (n * n);

  return sqrt(6*result);
}
```

37

pi – solution cont.

```
/*This program approximates pi*/
#include <stdio.h>

int main(void) {
  int num;

  printf("enter number of terms\n");
  scanf("%d",&num);

  printf("The calculated approximation of pi is: %g\n",pi(num));

  return 0;
}
```

38

Example

Modify the previous function that approximates $\prod$. The function should accept an argument specifying the desired accuracy, and keep adding terms until the contribution of the next term drops below this level.

Write a program that gets a (small) double, epsilon, from the user, and approximates $\prod$ within this function.

39

pi (advanced)- solution

```
#include <math.h>
double pi(double epsilon) {
  double n=1, result = 0, term = 1;

  while(sqrt(term*6) >= epsilon)  {
        result += term;
        n=n+1;
        term = 1.0/(n*n);
  }

  return sqrt(result*6);
}
```

40

pi (advanced)- solution cont.

```
/*This program approximates pi*/
#include <stdio.h>

int main(void) {
  double epsilon;

  printf("enter approximation factor\n");
  scanf("%lf",&epsilon);

  printf("The calculated approximation of pi is: %g\n",pi(epsilon));

  return 0;
}
```

41