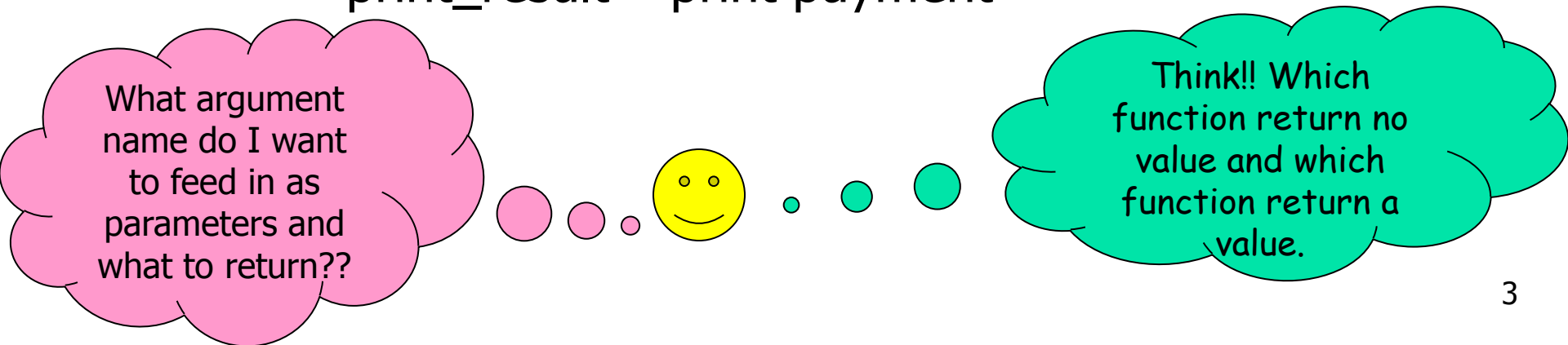
Functions (2)

Outline

- Recall - sample application
  - functions that return no value
  - functions that return a value
- Recall – global variable vs. local variable
- Recall – pass by value
- Functions that “return” more than one value
- Recursive function

Sample application

- Write a C program that reads item code and quantity, then calculate the payment.Use functions:
  - menu – print item code menu
  - determine_price – determine price based on item code
  - calc - calculate payment
  - print_result – print payment



Sample application-cont

```
#include <stdio.h>
void menu();
float determine_price(int);
float calc(float,int);
void print_result(float);
int main()
{    int code,qty;float price,pay;
     menu();
     printf("Enter item code and quantity:");
     scanf("%d %d", &code,&qty);
     price= determine_price(code);
     pay=calc(price,qty);
     print_result(pay);
     return 0;
}
```

Sample application-cont

```
void menu( )
{
        printf("Code\tItem\tPrice\n");
        printf("1\tPapaya\t1.00\n");
        printf("2\tMelon\t2.00\n");
        printf("3\tDurian\t3.00\n");
        printf("\tOthers\t4.00\n");
}
float determine_price(int item_code)
{       float pricing;
        switch(item_code)
        {
                case 1:pricing=1.00;break;
                case 2:pricing=2.00;break;
                case 3:pricing=3.00;break;
                default:pricing=4.00;
        }
        return(pricing);
}
float calc(float item_price,int quantity)
{       float answer;
        answer=item_price*quantity;
        return(answer);
}
void print_result(float payment)
{       printf("Payment is %.2f\n", payment);
}
```

```
[yasmin@localhost week5]$ gcc testing.c
[yasmin@localhost week5]$ ./a.out
Code    Item    Price
1       Papaya  1.00
2       Melon   2.00
3       Durian  3.00
        Others  4.00
Enter item code and quantity:1 3
Payment is 3.00
[yasmin@localhost week5]$ ./a.out
Code    Item    Price
1       Papaya  1.00
2       Melon   2.00
3       Durian  3.00
        Others  4.00
Enter item code and quantity:9 3
Payment is 12.00
```

Global variable vs. local variable

modification

```
#include <stdio.h>
void menu();
float determine_price(int);
float calc(float,int);
void print_result(float);
int code,qty;float price,pay;
int main()
{
        menu();
        printf("Enter item code and quantity:");
        scanf("%d %d", &code,&qty);
        price= determine_price(code);
        pay=calc(price,qty);
        print_result(pay);
        return 0;
}
void menu( )
{
        printf("Code\tItem\tPrice\n");
        printf("1\tPapaya\t1.00\n");
        printf("2\tMelon\t2.00\n");
        printf("3\tDurian\t3.00\n");
        printf("\tOthers\t4.00\n");
}
```

```
float determine_price(int code)
{
        code--;
        switch(code)
        {
                case 1:price=1.00;break;
                case 2:price=2.00;break;
                case 3:price=3.00;break;
                default:price=4.00;
        }
        return(price);
}
float calc(float price,int quantity)
{
        pay=pay+1;
        pay=price*quantity;
        return(pay);
}
void print_result(float pay)
{       printf("Payment is %.2f\n", pay);
}
```

```
[yasmin@localhost yasmin]$ gcc testing2.c
[yasmin@localhost yasmin]$ ./a.out
Code    Item    Price
1       Papaya  1.00
2       Melon   2.00
3       Durian  3.00
        Others  4.00
Enter item code and quantity:1 4
Payment is 16.00
[yasmin@localhost yasmin]$ ./a.out
Code    Item    Price
1       Papaya  1.00
2       Melon   2.00
3       Durian  3.00
        Others  4.00
Enter item code and quantity:3 1
Payment is 2.00
```

However, sometimes we need to do some modification from inside a function, using global variable will make things worse!!!

Pass by Value

- If a parameter is passed by value, then the value of the original data is copied into the function's parameter *(scope: local variable(s))*
- In other words, it (i.e. local variable) has its own copy of the data
- changes to copy **do not change** original data
- During program execution, it (i.e. local variable) will manipulate the data stored in its own memory space

Pass by Value (Example)

```
#include <stdio.h>
void fun1(int, int);  //function prototype
int main(void)
{
      int a=5, b=10;
      printf("Before fun 1\n‟);
       printf(" a = %d  b = %d\n”, a, b);
       fun1(a, b);  //function call
       printf("\nAfter fun 1\n‟);
       printf(" a = %d  b = %d\n”, a, b);
       return 0;
}
void fun1(int aa, int bb)  //function definition
{
      aa++;
       bb--;
       printf("\n\nInside fun 1\n)";
       printf("aa = %d  bb = %d\n”, aa, bb);
}
```

Output

```
Before fun 1
a = 5 b = 10


Inside fun 1
aa = 6 bb = 9


After fun 1
a = 5 b = 10
```

Functions that "return" more than one value

- When we talk about functions that "return" more than one value it also means that we want to pass arguments by reference
  - pass **addresses** (references), NOT value/data
  - allows **direct manipulation**
  - changes **will affect** original data

Functions that “return” more than one value

- There are cases where you need to manipulate the value of an external variable from inside a function, thus we pass the values by reference

Sample application

- Write a C program that calculates and print average of 2 test marks.
- Your program should have function:
  - read – read 2 test marks
  - calc_avg –calculate average of two test marks
  - print-print average

Sample application

Functions that "return" more than one value i.e. arguments are pass by reference

```
#include <stdio.h>
void read_marks(float*, float*);
float calc_avg(float, float);
void print(float);
int main(void)
{
     float marks1, marks2, avg;

     read_marks(&marks1, &marks2);
     avg = calc_avg(marks1, marks2);
     print(avg);
     return 0;
}
```

```
void read_marks(float *m1, float *m2)
{

    printf("Enter marks for test1 and test2 : ");
    scanf("%f %f", m1,m2); //notice no &
}
float calc_avg(float m1, float m2)
{
        return((m1 + m2)/2);
}
void print(float average)
{
 printf("\nAverage marks are :%.2f\n",average);
}
```

Output
Enter marks for test1 and test2 : 70 80
Average marks are : 75.00

Pass by Reference

- A function's parameter that receives the location (memory address) of the corresponding actual variables
- When we attach * (star) after the *arg_type* in the parameter list of a function, then the variable following that *arg_type* is passed by reference
- It **stores the address** of the actual variable, NOT the value
- During program execution to manipulate the data, the address stored will direct control to the memory space of the actual variable
- Syntax
  - In function protoype and function definition, put the * (star) after the data type
  - In function call, put the &(ampersand) before the argument name to be passed by reference

Pass by Reference (cont.)

- Pass by Reference are useful in two situations:
  - when you want to return more than one value from a function
  - when the value of the actual parameter needs to be changed

Sample application

- Write a C program that reads character and calculates numbers of vowel and consonant
- Your program should have function:
  - read – read character
  - find_count_vc –determine and calculate number of vowel or consonant
  - print-print number of vowel or consonant

Sample application

```
Enter character : f
Do you want to continue?y
Enter character : I
Do you want to continue?y
Enter character : k
Do you want to continue?n
Number of vowel : 1
Number of consonant : 2
```

```
#include <stdio.h>
#include <string.h>
char read();
void find_count_vc(char, int*, int*);
void print(int,int);
int main()
{   char ch, choice; int count_v=0,count_c=0;
        do
        {       ch = read();
                find_count_vc(ch, &count_v, &count_c);
                printf("Do you want to continue?");
                scanf("%c", &choice);
                getchar();
        }while((choice == 'y') ||(choice =='Y'));
        print(count_v,count_c);
        return 0;
}
char read()
{       char ch1;
        printf("Enter character : ");
        scanf("%c", &ch1);
        getchar();
        return(ch1);
}
```

```
void find_count_vc(char ch1, int *vowel, int *consonant)
{
        switch(ch1)
        {       case 'A':
                case 'a':
                case 'E':
                case 'e':
                case 'I':
                case 'i':
                case 'O':
                case 'o':
                case 'U':
                case 'u': *vowel = *vowel +1;break;
                default: *consonant = *consonant + 1;
        }
}
void print(int vowel, int consonant)
{
        printf("Number of vowel : %d\n", vowel);
        printf("Number of consonant : %d\n", consonant);
}
```

Functions that "return" more than one value i.e. arguments are passed by ref

Pass by Reference (Example)

```
#include <stdio.h>
void fun1(int, int*);  //function prototype
int main(void)
{
        int a=5, b=10;
        printf("Before fun 1\n");
        printf(" a = %d b = %d",a, b);
        fun1(a, &b);  //function call
        printf("\n\nAfter fun 1\n");
       printf("a = %d  b = %d\n",a,b);
        return 0;
}
void fun1(int aa, int * bb)  //function definition
{
        aa++;
        *bb--;
        printf("\n\nInside fun 1\n");
        printf("aa = %d  bb = %d",aa,bb);
}
```

Output
Before fun 1
a=5 b = 10

Inside fun 1
aa = 6 bb = 9

After fun 1
a = 5 b = 9

Recursive Functions

- Recursion is a term describing functions which are called by themselves (functions that calls themselves)
- Recursive function has two parts i.e. base case and not base case
- If not base case, the function breaks the problem into a slightly smaller, slightly simpler, problem that resembles the original problem and
  - Launches a new copy of itself to work on the smaller problem, slowly converging towards the base case
  - Makes a call to itself inside the `return` statement
- Eventually the base case gets solved and then that value works its way back up to solve the whole problem
- Recursion is very useful in mathematical calculations and in sorting of lists

Recursive Functions (cont.)

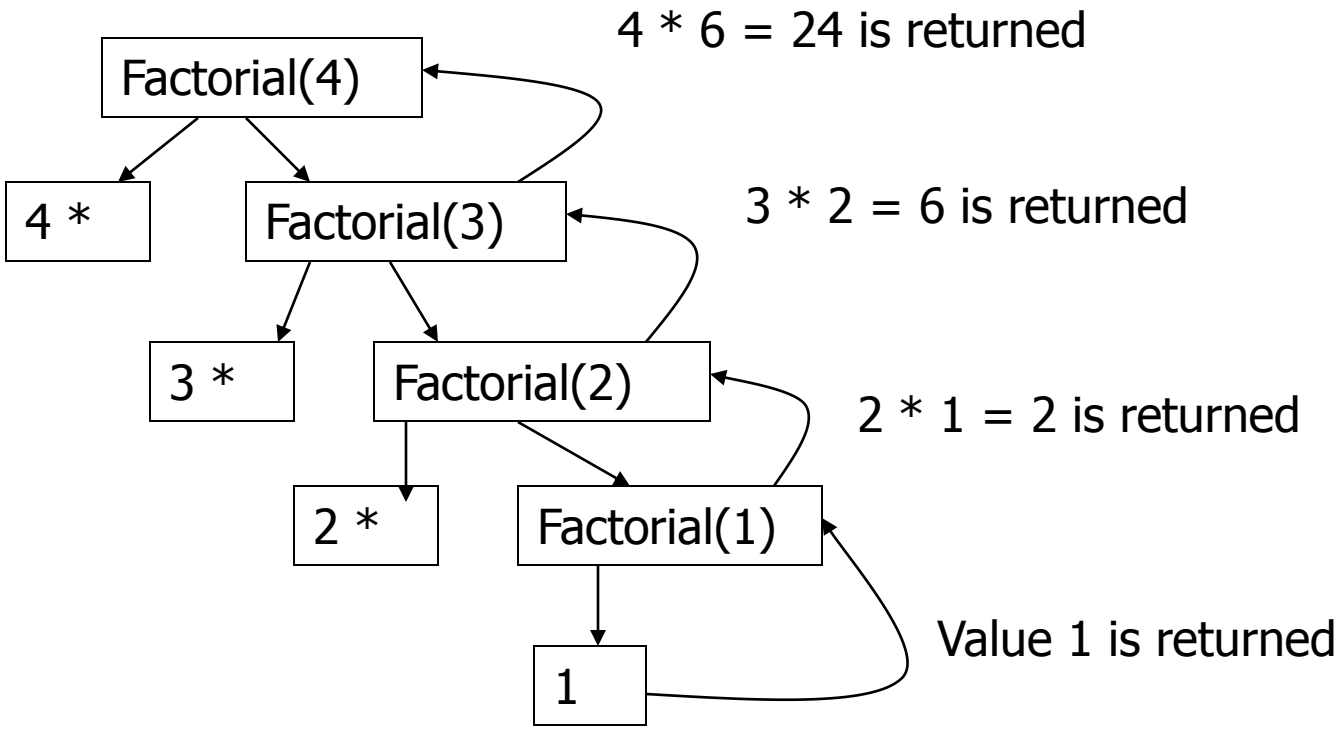
- Example: factorial

$$n! = n * (n - 1) * (n - 2) * \ldots * 1$$

- Recursive relationship:
  - $(n! = n * (n - 1)!)$
  - $5! = 5 * 4!$
  - $4! = 4 * 3!\ldots$
- Base case ($1! = 0! = 1$)

Recursive Functions(Example)

- Factorial



Recursive Functions(Example)

```
#include <stdio.h>
int Factorial(int n)
{
    if(n <= 1)
        return 1;
    else
        return ( n * Factorial(n-1));
        }
        void main()
        {
            int n=4;
            printf("Factorial %d  is %d",n, Factorial(n));
        }
```

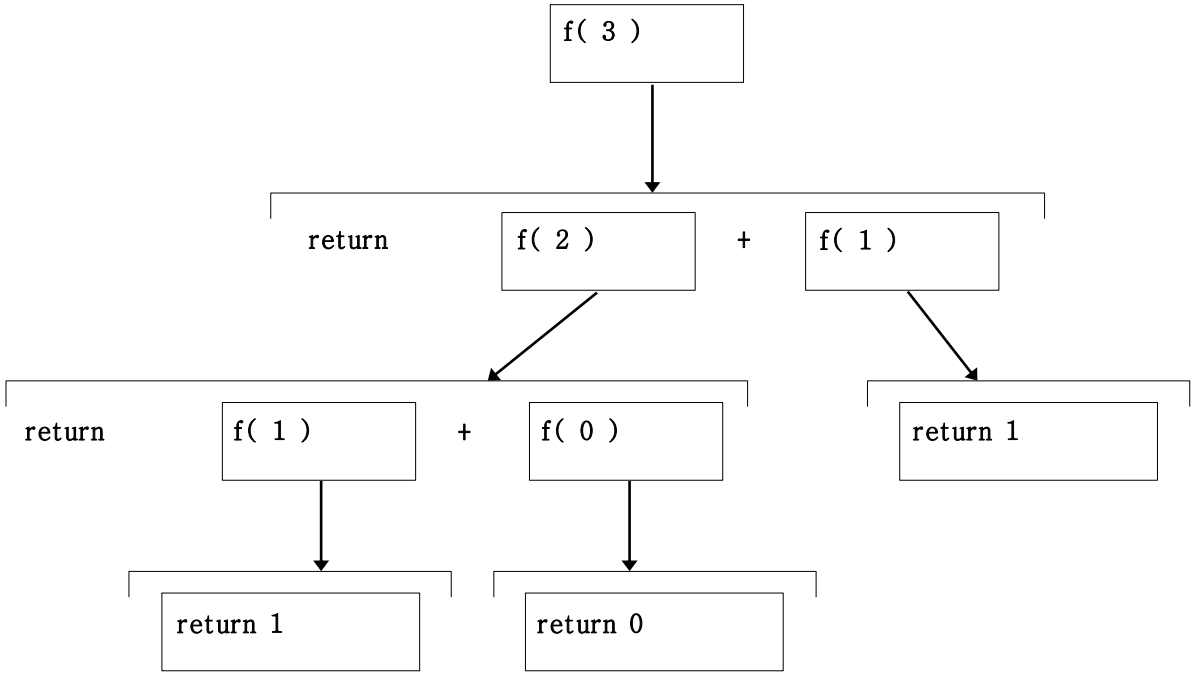
Recursive Functions (Example)

- Fibonacci series: 0, 1, 1, 2, 3, 5, 8…
  - Each number sum of two previous ones
  - Example of a recursive formula:

```
fib(n) = fib(n-1) + fib(n-2)
```

Recursive Functions (Example)

- Diagram of Fibonacci function:



Recursive Functions (Example)

- Sample code for **`fibonacci`** function

```
long fibonacci( long n )
{
 if ( n == 0 || n == 1 ) //base case
     return n;
 else
     return fibonacci( n - 1 ) +
            fibonacci( n - 2 );
}
```