Chapter 5

Functions

5.1 *Functions*(Page204)

- The purpose of a C function, whether library or user-written, is to receive data, operate on the data, and directly return at most a single value
- As we have already seen with the *printf*() and *scanf* () functions, a function is called, or used, by giving the function's name and passing any data to it in the parentheses

5.1 *Functions*(Page204)

- We have written programs with just one function – **main()**
- Program with only one function is difficult to maintain
- To define and make use of a programmer-defined function, three things are needed
  - (1)function definition
  - (2)function call
  - (3)function prototype

```
void func1(void) ;
main()
{
  printf("This is in main_1\n");
  func1();
  printf("This is in main_2\n");
}
void func1(void)
{
  printf("This is in func1\n");
}
```

(1)Functions Definition

- **ReturnType FunctionName(ArgumentType ArgumentName,. . .)**
  {
  Statements
  }
- First line called ***function header***
- Function body
- **void** meaning no return value

```
void func2 (int n, double x);
void main (void)
{
   int m;
   double y;
   m = 15;   y = 3.14;
   printf ("main_m=%d\n", m);
   func2 (m, y);
   printf ("main_m=%d\n", m);
}
void func2 (int n, double x)
{
   printf ("n=%d\n",n);
   printf ("x=%lf\n",x);
}
```

(2)Function Call

- To call (or invoke) a function, write function name followed by arguments enclosed in parentheses

  func2(m,y);

- Transfers program control to the function
- Control goes back to location at which the function was called

```
void func2 (int n, double x);
void main (void)
{
   int m;
   double y;

   m = 15;
   y = 3.14;
   printf ("main_m=%d\n", m);
   func2 (m, y);
   printf ("main_m=%d\n", m);
}
void funct2 (int n, double x)
{
   printf ("n=%d\n",n);
   printf ("x=%lf\n",x);
}
```

Parameters in Function

- To pass any information from main to called func2, we created the variable names **n** and **x**

void func2(int n, double x)

```
void func2 (int n, double x);
void main (void)
{
   int m;
   double y;

   m = 15;
   y = 3.14;
   printf ("main_m=%d\n", m);
   func2 (m, y);
   printf ("main_m=%d\n", m);
}
void func2 ( int n, double x )
{
   printf ("n=%d\n",n);
   printf ("x=%lf\n",x);
}
```

(3)Function Prototype

- A declaration indicates that the function exists
- Use the function header and ;
- If a function is defined before called, then Function Prototype does not needed.

```
void func2 (int n, double x);
void main (void)
{
    int m;
    double y;

    m = 15;
    y = 3.14;
    printf ("main_m=%d\n", m);
    func2 (m, y);
    printf ("main_m=%d\n", m);
}
void func2 (int n, double x)
{
    printf ("n=%d\n",n);
    printf ("x=%lf\n",x);
}
```

5.2Functions Return Just One Value

- function that returns a value must have a ***type*** that reflects the type of value returned

  long GetFactorial(int m);

- In doing this, a return statement must appear in the body of the function

  return lProduct;

- form of the return

  return expression;

- or

  return (expression);

- Omitting the type, by default, defines the function's return value to be of type ***int***.

```
#include <stdio.h>
long GetFactorial(int m);
int main (void)
{
int        n;
long       lFactorial;
double     f;
  scanf("%d", &n);
  lFactorial=GetFactorial(n);
  f=1.0/lFactorial;
  printf ("1.0/%d=%e", n, f);
  return (0);
}
long GetFactorial(int m)
{
int        i;
long       lProduct;
  lProduct=1;
  for (i=m; i>=1; i--)
    lProduct*=i;
  return lProduct;
}
```

Functions Return One Value

- Where does the value of the variable **lProduct** (in example) return?
  - causes execution to transfer **unconditionally** to calling statement
  - the value of **lProduct** is passed to the variable in front of calling function

    lFactorial = GetFactorial (n);
- ***return* statement can appear *anywhere* in the function body**

```
#include <stdio.h>
long GetFactorial(int m);
int main (void)
{
int         n;
long        lFactorial;
double      f;
  scanf("%d", &n);
  lFactorial=GetFactorial(n);
  f=1.0/lFactorial;
  printf ("1.0/%d=%e", n, f);
  return (0);
}
long GetFactorial(int m)
{
int         i;
long        lProduct;
  lProduct=1;
  for (i=m; i>=1; i--)
    lProduct*=i;
  return lProduct;
}
```

Try

- Write a function "FindMax" with two parameters of ***int x*** and ***int y***. The function is to find maximum of two input parameters ***x*** and ***y*** and return this maximum number.
- Given values of 5 ***int*** variables, try to find the maximum of these 5 variables by calling the above function "FindMax"

```
int FindMax(int x,int y);
int _tmain(int argc, _TCHAR* argv[])
{
int    a,b,c,d,e,nMax;
  a=2;b=5;c=4;d=8;e=10;
  nMax=FindMax(a,b);
  nMax=FindMax(nMax,c);
  nMax=FindMax(nMax,d);
  nMax=FindMax(nMax,e);
  printf("Maximum=%d",nMax);
  return 0;
}
int FindMax(int x,int y)
{
int nMax;
  nMax=(x>y)?x:y;
  return nMax;
}
```

Local Variables

(Page229)

- The variables created **inside** a function are said to be ***local*** to the function, or ***local*** variables.
- ***Local*** variables are available only to the function itself
- The same variable name can be declared and used in more than one function

```
#include "stdafx.h"
long GetFactorial(int m);
int _tmain(int argc, _TCHAR* argv[])
{
int  n;  long  lFactorial;  double   i;
  scanf("%d", &n);
  lFactorial=GetFactorial(n);
  i=1.0/lFactorial;
  printf ("1.0/%d=%e", n, i);
  return 0;
}
long GetFactorial(int m)
{
long   lFactorial;
  lFactorial =1;
  for (int i=m; i>=1; i--)
  lFactorial*=i;
  return (lFactorial);
}
```

Global Variables

(Page229)

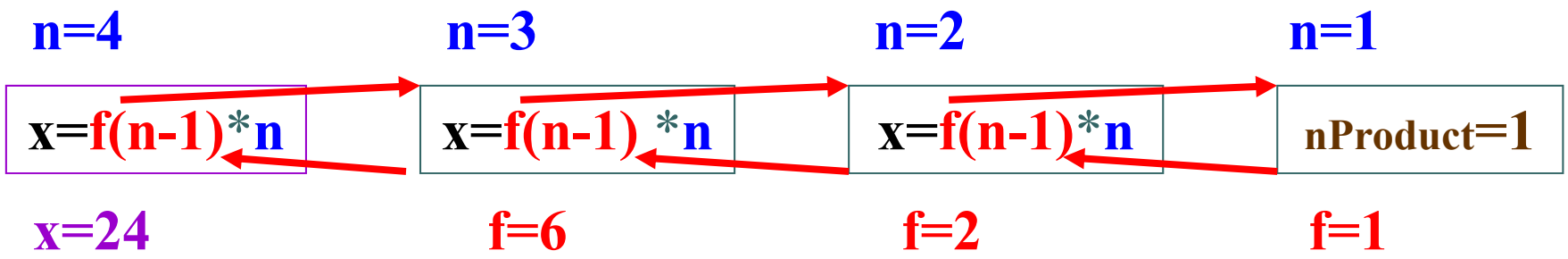
- Global variable is declared **outside** any function.
- Global variables can be used by all functions in a program that are physically placed **after** the global variable declaration.

```
#include "stdafx.h"
void GetFactorial(int m);
long   lFactorial;
int _tmain(int argc, _TCHAR* argv[])
{
int                    n;
double i;
  scanf("%d", &n);
  GetFactorial(n);
  i=1.0/lFactorial;
  printf ("1.0/%d=%e", n, i);
  return 0;
}
void GetFactorial(int m)
{
int                    i;
    lFactorial=1;
    for (i=m; i>=1; i--)
        lFactorial*=i;
}
```

Embeded Calling

4!=	1 *2 *3 *4
4! n=4	3!*4
3! n=3	2!*3 *4
2! n=2	1!*2 *3 *4
1! n=1	1* 2 *3* 4

```
int f(int n)
{
int  nProduct;
  if(n<=1)
     nProduct=1;
  else
     nProduct=f(n-1)*n;
  return nProduct;
}
int _tmain(int argc, _TCHAR* argv[])
{
int   nFactorial;
  nFactorial=f(4);
  return 0;
}
```



Try

- **Write a embeded function “GetSum” to calculate 1+2+3+4**

Multiple Functions

```
int GetFactorial(int n)
{
int  nProduct;
  if(n<=1)
     nProduct=1;
  else
     nProduct=GetFactorial(n-1)*n;
  return nProduct;
}
int GetSumFactorial(int nMax)
{
int  n,nSum=0,nFactorial;
  for(n=1;n<nMax;n++)
  {
     nFactorial=GetFactorial(n);
     nSum+=nFactorial;
  }
  return nFactorial;
}
int _tmain(int argc, _TCHAR* argv[])
{
int   nSumFactorial,nFactorial;
  nFactorial=GetFactorial(4);
  nSumFactorial=GetSumFactorial(5);
  return 0;
}
```

5.5 Address , Pointer (Page235)

- "Address of" operator & was used to ***retrieve*** the address of a variable
- A variable that stores address is called Pointer.
- **In declaration**, * is used to declared a Pointer:

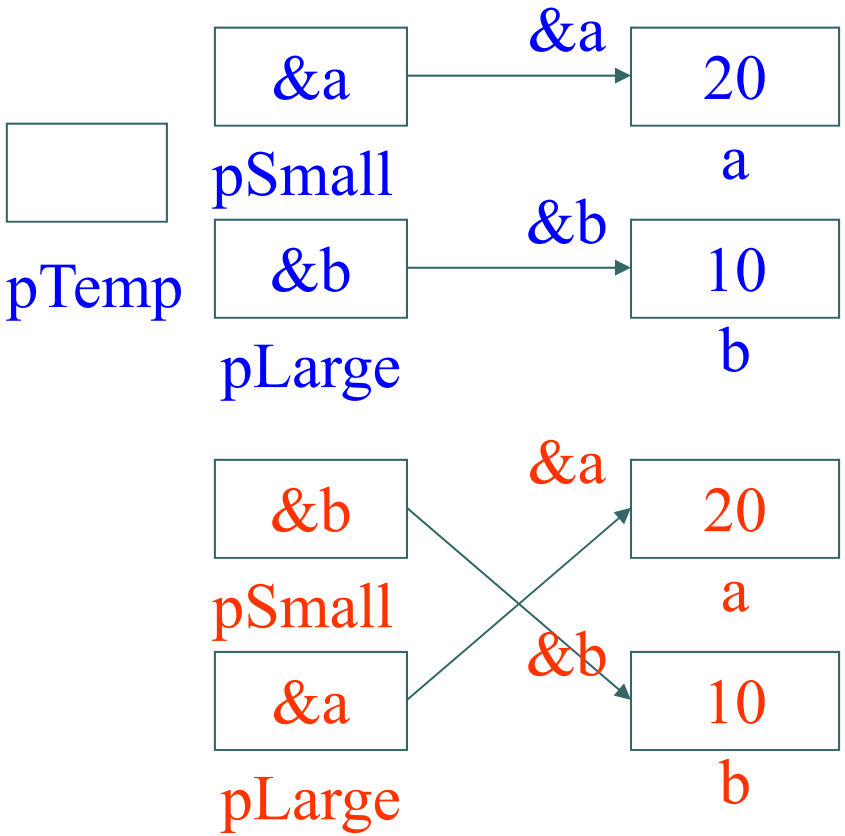
```
int   *pPointer;
```

- **In statements**, * is used to get value pointed by pointer

```
int   x,y;
int   *pPointer;
   pPointer =&x;
   y=*pPointer;
```

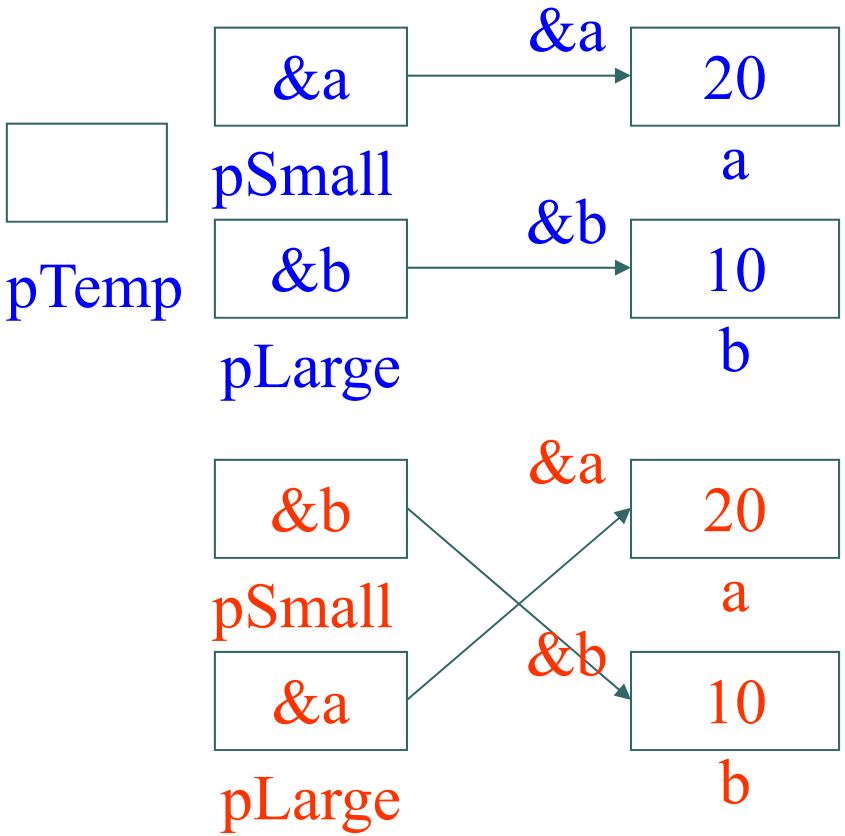
Different meaning!!!

Compare two variables by their pointers and print out them in order by pointers *pSmall* and *pLarge*




```
#include "stdafx.h"
int _tmain(int argc, _TCHAR* argv[])
{
int   a=20,b=10;
int   ??;
  pSmall=??;
  pLarge=??;
  if(pSmall>pLarge) //??
  {
     pTemp=pSmall;
     pSmall=pLarge;
     pLarge=pTemp;
  }
  printf("Small=%d",??);
  printf(",Large=%d\n",??);
  return 0;
}
```

Compare two variables by their pointers and print out them in order by pointers *pSmall* and *pLarge*




```
#include "stdafx.h"
int _tmain(int argc, _TCHAR* argv[])
{
int   a=20,b=10;
int   *pSmall,*pLarge,*pTemp;
  pSmall=&a;
  pLarge=&b;
  if(*pSmall>*pLarge)
  {
    pTemp=pSmall;
    pSmall=pLarge;
    pLarge=pTemp;
  }
  printf("Small=%d",*pSmall);
  printf(",Large=%d\n",*pLarge);
  return 0;
}
```

5.5 “Return” More Values

```
#include "math.h"
void Complex(float x,float y,double *pModulus,double *pAngle)
{
double   dSqureRoot,dAngle;
  dSqureRoot=sqrt(x*x+y*y);          dAngle=atan(y/x);
  *pModulus=dSqureRoot;              *pAngle=dAngle*180.0/3.14;
}
int _tmain(int argc, _TCHAR* argv[])
{
double   x,y,dModulus,dAngle;
  x=50;   y=50;
  Complex(x,50,&dModulus,&dAngle);
  printf("Modulus=%lf,Angle=%lf\n",dModulus,dAngle);
  return 0;
}
```

Home Work

- Page 252-Application Program 5.2 ?
- Page 235-2
- Page 235-4
- Page 260-5.4