Function (Call by Reference)

Call by Reference (Address)

- Passes an address as an argument to a function
  - address = pointer to memory address
  - The argument must NOT be an "expression to be evaluated"
- Avoids copying data

Call by Reference

- The called function can change the arguments in the calling function
  - side effects, similar to global variables
- Multiple address arguments can result in changes in multiple input values.
  - way to return multiple values
  - (call by value can return only one data.)

Call by value

```
void main ()
{
   void newVal (float );  /* function prototype */
   float  testval;

   printf ("₩nEnter a number: ");
   scanf ("%f", &testval);

   newVal (testval);      /* function call        */

}

void newVal (float num) /* function definition */
{
   num = num + 20.2;
}
```

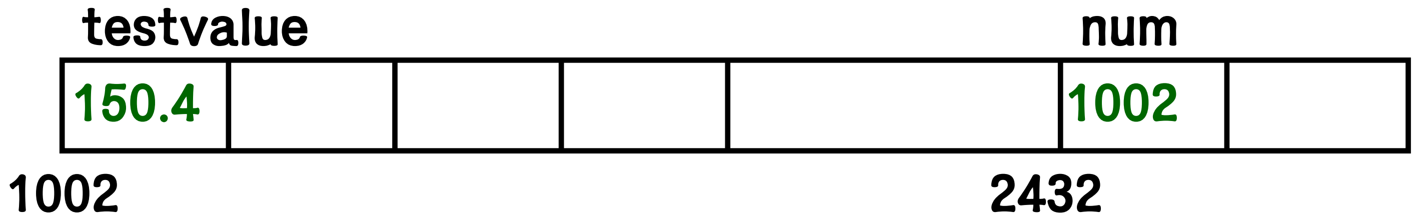
Using return type

```
void main ()
{
 float newVal (float );  /* function prototype */
   float  testval;

   printf (“₩nEnter a number: ”);
   scanf (“%f”, &testval);

   result =newVal (testval);      /* function call
*/

}

float newVal (float num) /* function definition */
{
   num = num + 20.2;
   return num;
}
```

Call by reference

```
void main ()
{
   void newVal (float *);  /* function prototype */
   float  testval;

   printf ("₩nEnter a number: ");
   scanf ("%f", &testval);

   newVal (&testval);       /* function call        */

}

void newVal (float *num) /* function definition */
{
   *num = *num + 20.2;
}
```

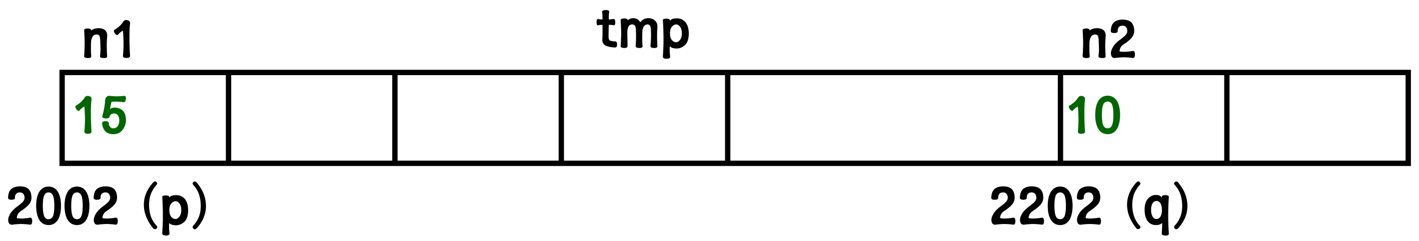
Memory Diagram



Exercise

```
void main ()
{
   void swap (int *, int *);  /* function prototype */
   int n1, n2;

   /* read n1 and n2   */

   swap (&n1, &n2);      /* function call        */

}

void swap (int *p, int *q) /* function definition */
{
   int  tmp;
   tmp = *p;
   *p = *q;
   *q = tmp;
}
```

Memory Diagram



Example: Function “Returning” Multiple Values

```
void main()
{
   int n1, n2;
   float f1, f2;
   void newVal (int, int, float *, float *);
   /*  read n1 and n2  */
   newVal (n1, n2, &f1, &f2);
}

void newVal (int val1, int val2, float *sum, float *remain)
{
   *sum = val1 + val2;
   *remain = val1 % val2;
}

   /* “returns” two values */
/* side effects – same as global variables */
```

Exercise

Write a function void minmax with 4 parameters.

minmax has 2 parameters whose data types are int and 2 parameters whose data types are int *.

minmax receives 2 integers, and determines the smallest and largest of the 2 integers, and returns them through the two pointer parameters.

Solution

```
void minmax (int num1, int num2,  int *min, int *max)
{
    if (num1 <= num2) {
      *min = num1;
      *max = num2;
    }
    else {
       *min = num2;
       *max = num1;
    }
}
```

Exercise

- Write a C (main) program that reads two integers, N1 and N2, and calls a function, calculate, by passing the addresses of N1 and N2.
- calculate calculates N1*N2, N1/N2, N1%N2, N1+N2 and returns the four results. The main program prints the results.
- (* hint: calculate has six parameters, (int *n1, int *n2, int *re1, int *re2, int *re3, int *re4), where re1, re2, re3, re4 store the four results.

Exercise

Write a C program that computes and prints the total salaries of people.
The program receives 5 base salaries (float) and 5 overtime payments (float), stores them in arrays base and overtime. The program calls a function void totpay with 3 array arguments, base, overtime and total.

totpay computes the total pay for each person by adding the base salary and overtime payment, and storing the result in total.

The main program prints total.

Solution (Sketch)

```
void totpay (float[5], float[5], float[5]);

void main()
{
   /*  declarations   */
   float base[5], overtime[5], total[5];
   /*  read and store values in array pay */
   totpay (base, overtime, total);
   /*  print total     */

void totpay (float b[], float o[], float t[])
{
  /* compute and store total for each person  */
}
```

Homework (30 points)

- Textbook Chapter 7.3
- Programming Exercises 2, 3a and 3b
- Must follow the problem solving steps (including the drawing of memory diagrams).

Chapter 7.3 Exercises

- 2. Write a C function named change() that accepts a single-precision number and the addresses of the integer variables named *quarters, dimes, nickels,* and *pennies*. The function should determine the number of quarters, dimes, nickels, and pennies in the number passed to it and write these values directly into the respective variables declared in its calling function.
- 3a. Write a function named secs() that accepts the time in hours, minutes, and seconds; and determines the total number of seconds in the passed data. Write this function so that the total number of seconds is returned by the function as an integer number.
- 3b. Repeat Exercise 3a but also pass the address of the variable *totSec* to the function secs(). Using this passed address, have secs() directly alter the value of *totSec*.

Homework (10 points)

- Textbook Chapter 9.4
  - Programming Exercise #2
- Write a C function named liquid() that is to accept an integer number and the addresses of the variables gallons, quarts, pints, and cups. The passed integer represents the total number of cups, and the function is to determine the number of gallons, quarts, pints, and cups in the passed value. Using the passed addresses, the function should directly alter the respective variables in the calling function. Use the relationships of 2 cups to a pint, 4 cups to a quart, and 16 cups to a gallon.

Homework (20 points)

- Textbook Chapter 8.3
- Programming Exercises 2 and 4
- Must follow the problem solving steps (including the drawing of memory diagrams).

Chapter 8.3 Exercises

- 2. Write a program that has a declaration in main() to store the following numbers into an array named *rates*: 6.5, 8.2, 8.5, 8.3, 8.6, 9.4, 9.6, 9.8, 10.0. There should be a function call to show() that accepts the *rates* array as a parameter named *rates* and then displays the numbers in the array.
- 4. Write a program that declares three one-dimensional arrays named *price*, *quantity*, and . Each array should be declared in main() and should be capable of holding 10 double-precision numbers. The numbers that should be stored in *price* are 10.62, 14.89, 13.21, 16.55, 18.62, 9.47, 6.58, 18.32, 12.15, 3.98. The numbers that should be stored in *quantity* are 4, 8.5, 6, 8.35, 9, 15.3, 3, 5.4, 2.9, 4.8. Your program should pass these three arrays to a function called extend(), which should calculate the elements in the *amount* array as the product of the equivalent elements in the *price* and *quantity* arrays (for example, amount[1] = price[1] * quantity[1]). After extend() has put values in the *amount* array, the values in the array should be displayed from within main().

End of Lecture