Pointers

A *pointer* is a variable which contains the address in memory of another variable. We can have a pointer to any variable type.

The *unary* operator & gives the “address of a variable”.

The *indirection* or dereference operator * gives the “contents of a memory location *pointed to* by a pointer”.

To declare a pointer to a variable do:

```
   int *pointer;
```

Examine the following program and see the pointer usage:

```
int main(void)
{
   int   var1=3,  //integer variables
         var2=5,
         *pVar;   //pointer to integer memory area

   pVar = &var1;
   printf("\nContent of pointer = %p\n", pVar); //... = 003CFD14
   printf("Content of the memory area referred by pointer = %d\n",
          *pVar); //... = 3

   *pVar = *pVar * 2;

   printf("\nContent of pointer = %p\n", pVar); //... = 003CFD14
   printf("Content of the memory area referred by pointer = %d\n",
          *pVar); //... = 6

   pVar = &var2;

   printf("\nContent of pointer = %p\n", pVar); //... = 003CFD08
   printf("Content of the memory area referred by pointer = %d\n",
          *pVar); //... = 5

   return (0);
}
```

Functions with Simple Output Parameters

Argument lists (parameters) provide the communication links between the main program and its functions. So far, we know how to pass inputs into a function and how to use the return statement to send back one result value from a function. This section describes how programmers use output parameters to return multiple results from a function. For this purpose, "by reference" parameter passing technique will be used.

Parameter Passing Techniques

By Value

This parameter passing technique is used if the value of the parameter will not be changed after the function call.

Example

```
     printf("Sum is: %d", total);
```

The value of total is not changed after the printf function is executed.

By Reference

This parameter passing technique is used if the value of the parameter will be changed after the function call.

Example

```
     scanf("%d %d %lf", &num1, &num2, &x);
```

The values of num1, num2 and x are changed after the above function call is executed.

Scope of Variables

If a variable is declared within a function (main function or any other function), than it is only available in the function where it is declared.

Example

```
int square(int value)
{
   int number;
   number = value * value;
   return (number);
}
```

```
int main(void)
{
   int number=4;

   printf("Number = %d, Square = %d.\n", number, square(number));

   printf("\n\n");
   system("PAUSE");
   return 0;
}
```

The variable `number` in the `square` is a completely different variable than the variable `number` in the `main` function. They have different locations in memory.

Call by Value Example

The following calcRoot function calculates the square root and the square of a given double value. Since call by value parameter passing technique is used, changes on the parameters are not reflected to main.

```
void calcRootSquare(double number, double sRoot, double square)
{
   sRoot = sqrt(number);
   square = pow(number, 2);
}

int main(void)
{
   double number=4,
          sRoot=0,
          square=0;

   printf("Before calculation\n");
   printf("\tnumber=%.2f\n\tsquare root=%.2f\n\tsquare=%.2f\n\n",
            number, sRoot, square);

   calcRootSquare(number, sRoot, square);

   printf("After calculation\n");
   printf("\tnumber=%.2f\n\tsquare root=%.2f\n\tsquare=%.2f\n\n",
            number, sRoot, square);

   printf("\n\n");

   return 0;
}
```

Call by Reference Example

When the & of a variable is passed to a function, the indirection operator (*) may be used in the function to modify the value at that location in the caller's memory.

In the following, corrected version of the previous program, the call by reference parameter passing technique is used. That's why, changes on the parameters are reflected to main.

```
void calcRootSquare(double number, double *sRoot, double *square)
{
   *sRoot = sqrt(number);
   *square = pow(number, 2);
}

int main(void)
{
   double number=4,
          sRoot=0,
          square=0;

   printf("Before calculation\n");
   printf("\tnumber=%.2f\n\tsquare root=%.2f\n\tsquare=%.2f\n\n",
            number, sRoot, square);

   calcRootSquare(number, &sRoot, &square);

   printf("After calculation\n");
   printf("\tnumber=%.2f\n\tsquare root=%.2f\n\tsquare=%.2f\n\n",
            number, sRoot, square);
   return 0;
}
```

A declaration of a simple output parameter such as `int *sRoot` and `int *square` tells the compiler that `sRoot` and `square` will contain the memory addresses of `int` type variables, i.e. `sRoot` and `square` are pointers to int variables.

Pointer is a data type which stores the memory address of data types.

address symbol	&
indirection symbol	*

Exercises

1. Given the following program and its partial output below, complete the output and trace how it is produced, clearly.

```
int main(void)
{
   int   a = 3,
         b = 5;

   int   *c,
         *d,
         *temp;


   c = &a;
   d = &b;                                          OUTPUT

   printf("%p\n", c);    ──────────────────────────►0022FF74

   temp = c;
   c = d;
   d = temp;

   printf("%p\n", c);    ──────────────────────────►0022FF70

   printf("%d\n", *c);    ─────────────────────────►?

   printf("%d\n", *d);    ─────────────────────────►?

   *temp = *c + *d;

   printf("%p\n", temp);    ───────────────────────►?

   printf("%d\n", *temp);    ──────────────────────►?

   return 0;
}
```

2. Write a function, which returns the given 3 parameters in the ascending order.