ICS103 Programming in C

Lecture 10: Functions II

Outline

- Introducing Functions that return multiple results
- What is a Pointer variable?
- Functions returning multiple results
- Triple use for Asterisk (*)
- More Examples

Introducing Functions that return multiple results

- So far, we know how to pass inputs into a function and how to use the return statement to send back at most one result from a function.
- However, there are many situations where we would like a function to return more than one result. Some Example are:
  - Function to convert time in seconds into hours, minutes and seconds
  - Function to find the quotient and remainder of a division
  - Function to return maximum, minimum and average from a set of values
- In this lecture, we discuss how a function can return more than one result, which is achieved through **output parameters,** which are **pointer variables**.
- Thus, to be able to write functions that return multiple results, we first need to learn about pointer variables.

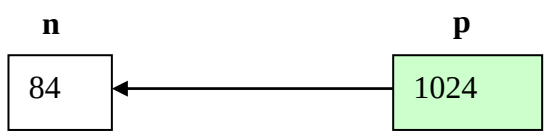
What is a Pointer variable?

- A pointer variable is a special variable, that stores the address of other normal variable.
- If a pointer variable stores the address of a char variable, we call it a character pointer and so on.
- A normal variable directly contains a specific value. A pointer variable on the other hand, contains an address of a variable that contains a specific value.
- Pointers like any other variables must be declared before they can be used. A pointer variable is declared by preceding its name with an asterisk.

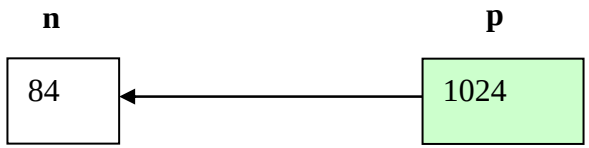
  Example: int *p;
- How can we initialize p? First we must have an integer variable, then we use the & operator to get the address of the variable and assign it to p.

  int n = 84;

  p = &n;
- Suppose that the int variable n is stored in the memory cell # 1024, then the following figure figure shows the relationship between n and p.



What is a Pointer variable? …



- A pointer variable such as p above, has two associated values:
- Its **direct value,** which is referenced by using the name of the variable, p. In this example, this value is 1024. We can print the direct value of a pointer variable using printf by using %p as the place holder.
- Its **indirect value**, which is referenced by using the indirection operator (*). So the value of *p is 84.

Reference	Value
p	Pointer (1024)
*p	84

Example 1:

- The following example demonstrate the relationship between a pointer variable and the character variable it is pointing to.

```
/* Shows the relationship between a pointer variable
*  and the character variable it is pointing to */
#include<stdio.h>
int main(void) {
   char g='z';
   char c='a';
   char  *p;
   p=&c;
   printf("%c\n",*p);
   p=&g;
   printf("%c\n",*p);
   *p='K';
   printf("%c\n",g);
   system("pause");
   return 0;
}
```

```
a
z
K
Press any key to continue . . .
```

Functions returning multiple results

- As we saw in the last example, pointer variables allow us indirect access to variables (e.g *p = ‘K’)
- This ability to indirectly access a variable is the key to writing functions that return multiple results.
- We declare such functions to have pointer variables as their formal parameters.
- From the calling function, instead of passing values of variables as actual arguments, we pass addresses of these variables.
- This will allow the function to indirectly manipulate the variables of the calling function – thus achieving the desired effect.

Example 2:

```
/* shows how function can
return multiple results */
#include <stdio.h>
void test1(int m, int n);
void test2(int *m, int *n);
void test3(int a, int *b);
int main(void) {
   int a=10, b=16;
   printf("a=%d,
b=%d\n",a,b);
   test1(a,b);
   printf("a=%d,
b=%d\n",a,b);
   test2(&a,&b);
   printf("a=%d,
b=%d\n",a,b);
   test3(a,&b);
   printf("a=%d,
b=%d\n",a,b);
   system("pause");
   return 0;
}
```

```
void test1(int m, int n) {
  m=5;
  n=24;
}

void test2(int *m, int *n)
{
  *m=5;
  *n=24;
}

void test3(int a, int *b) {
  a=38;
  *b=57;
}
```

```
a=10,  b=16
a=10, b=16
a=5, b=24
a=5, b=57
Press any key to continue . . .
```

Triple use for Asterisk (*)

- We have now seen three distinct meanings of the symbol *.
- As Multiplication operator: x * y => x times y
- In declaration `int * p`
  - * tells the compiler that a new variable is to be a pointer (read as “pointer to”)
  - Thus, in this case, it is a part of the name of the type of the variable.
- As unary indirection operator : `x = * p`
  - It provides the content of the memory location specified by a pointer. It mean “follow the pointer”.
  - It can also stand on the left side of an assignment. `* p = ‘K’`
  - Here the type depends on the variable being pointed – char in the above case.
  - It is a common mistake by students to interpret the above as a pointer type.

Example 3:

```
/* computes the area and circumference of a circle, given its
radius */
#include <stdio.h>

void area_circum (double radius, double *area, double *circum);

int main (void) {
    double radius, area, circum ;

    printf ("Enter the radius of the circle > ") ;
    scanf ("%lf", &radius) ;

    area_circum (radius, &area, &circum) ;
    printf ("The area is %f and circumference is %f\n", area,
circum) ;
    system("pause");
    return 0;
}

void area_circum (double radius, double *area, double *circum)
{
    *area = 3.14 * radius * radius ;
    *circum = 2 * 3.14 * radius ;
}
```

Example 4:

```
/* Takes three integers and returns their sum, product and average
*/
#include<stdio.h>

void myfunction(int a,int b,int c,int *sum,int *prod, double *average);

int main (void) {
        int n1, n2, n3, sum, product;
        double av_g;
        printf("Enter three integer numbers > ");
        scanf("%d %d %d",&n1, &n2,&n3);
        myfunction(n1, n2, n3, &sum, &product, &av_g);
        printf("\nThe sum = %d\nThe product = %d\nthe avg =
%f\n",sum,product,av_g);
        system("pause");
        return 0;
}

void myfunction(int a,int b,int c,int *sum,int *prod, double *average)
{
   *sum=a+b+c;
   *prod=a*b*c;
   *average=(a+b+c)/3.0;
}
```

Example 5:

```
/* takes the coefficients of
quadratic equation  a, b and c and
returns its roots */
#include<stdio.h>
#include<math.h>

void quadratic(double a,double b,
double c, double *root1, double
*root2);

int main(void) {
   double a,b,c,r1,r2;
   printf("Please enter coefficients
of the equation: [a b c] > ");
   scanf("%lf%lf%lf",&a,&b,&c);

   quadratic(a,b,c,&r1,&r2);

   printf("\nThe first root is :
%f\n",r1);
   printf("The second root is :
%f\n", r2);
   system("pause");
   return 0;
}
```

```
void quadratic(double a,double b,
double c, double *root1, double
*root2) {
   double desc;

   desc =b*b-4*a*c;
   if(desc < 0) {
     printf("No real roots\n");
     system("pause");
     exit(0);
   }
   else {
      *root1=(-b+sqrt(desc))/(2*a);
      *root2=(-b-sqrt(desc))/(2*a);
   }
}
```

Example 6:

```
/* swaps the values between 2 integer
variables */
#include <stdio.h>

void readint(int *a, int* b);
void swap (int *a, int *b);

int main (void ) {
   int num1,num2;
   readint(&num1,&num2);
   printf("before swapping num1= %d,
num2=%d\n",num1,num2);
   swap(&num1,&num2);
   printf("after swapping num1= %d,
num2=%d\n",num1,num2);
   system("pause");
   return 0;
}

void readint (int *a, int *b) {
   printf("enter first integer number > ");
   scanf("%d",a);
   printf("enter second integer number > ");
   scanf("%d",b);
}
```

```
void swap (int *a, int*b)
{
   int temp;
   temp=*a;
   *a=*b;
   *b=temp;
}
```

```
enter first integer number > 3
enter second integer number > 4
before swapping num1= 3, num2=4
after swapping num1= 4, num2=3
```

Because *a* and *b* are pointer variables, we do not use the & operator for scanf.