Paper 102: Programming & Problem solving through C

Lecture-11:Unit-II Functions

A.V. Hujon, Dept. of Computer Science,SAC

Different ways of calling a function with arguments

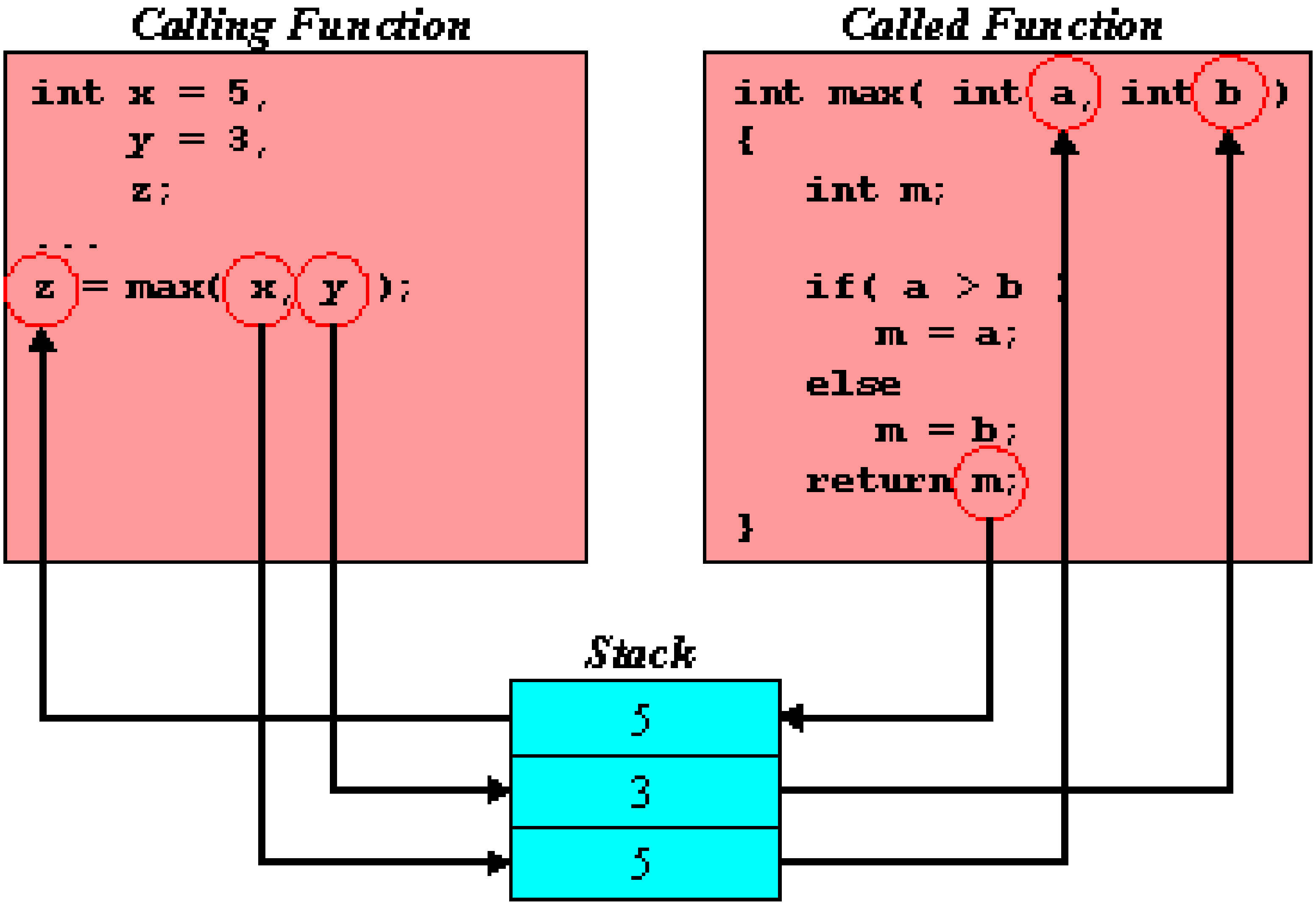
When calling a function, arguments can be passed to a function in two ways

- Call by value
- Call by reference

Call By Value

Here arguments are being passed by value

- temporary copy of argument (constant, variable, expression) is provided to function (by way of a *stack*)
- function can change the copy, but not the original value in calling function
- Call by value results in greater independence between modules



Calling Function

```
int x = 5,
    y = 3,
    z;
...
z = max( x, y );
```

Called Function

```
int max( int a, int b )
{
    int m;

    if( a > b )
        m = a;
    else
        m = b;
    return m;
}
```

Stack

```
#include<stdio.h>
void  callbyval(int,int);
void main()
{
      int a,b;
      a=b=10;
      printf("\n The values of a and b before calling the function is %d
      and %d",a,b);
      callbyval(a,b);
      printf("\n After the function is executed the values of a is %d
      and b is %d",a,b);
}
void callbyval(int a,int b)
{
      a=a * a;
      b=b * b;
     printf("\n The values of a and b inside the callbyval function is
     %d and %d",a,b);
}
```

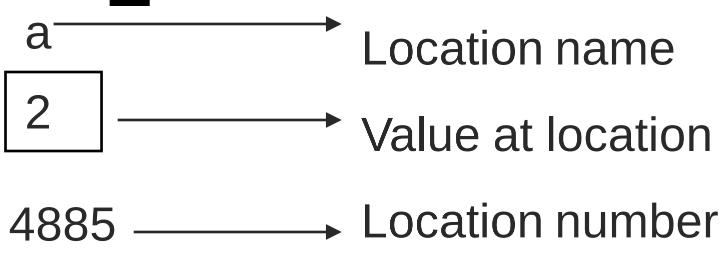
Scope Rules

- Scope of a variable is that part of a program in which the variable can be referenced
- visible only in the block in which they are declared

```
void display(int);
void main()
{   int i=10;
    display(i);
}
void display(int j)
{   int k=27;
    printf(“\n %d   %d”,j,k);
}
```

1. Variable accessible in main() is i
2. Variables accessible in display is j and k
3. Scope of i is within the main() and scope of j and k is in the display()

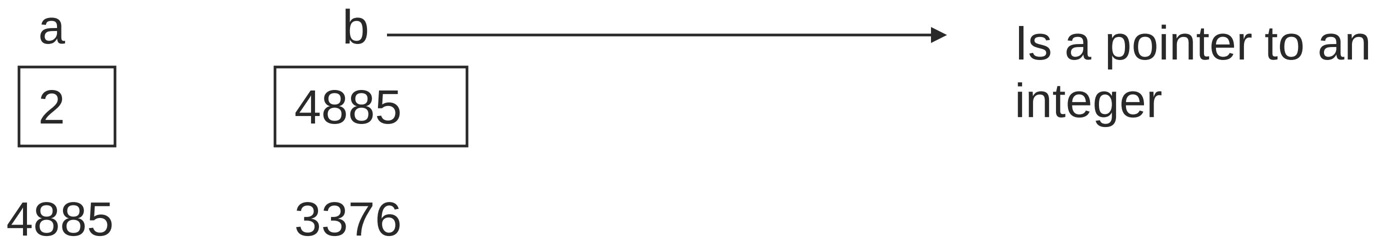
An Introduction to pointers



This declaration tell the C compiler to:

1. Reserve space in memory to hold an integer value
2. Associate the name a with this memory location
3. Store the value 2 at this location

- A *pointer* is a variable that contains the address of a variable
- Pointers are declared to "point" to a variable of a particular type (**int**, **double**, etc.)
- When we use a pointer to access the value stored in the variable to which it points, we are using *indirect addressing*



Is a pointer to an integer

```
int a=2;
int *b;
b=&a;
printf("\n Address of a=%u",&a);
printf("\n Address of a=%u",b);
printf("\n Address of b=%u",&b);
printf("\n Value of b=%u",b);
printf("\n Value of a=%d",a);
printf("\n Value of a=%d",*(&a));
printf("\n Value of a=%d",*b);
```

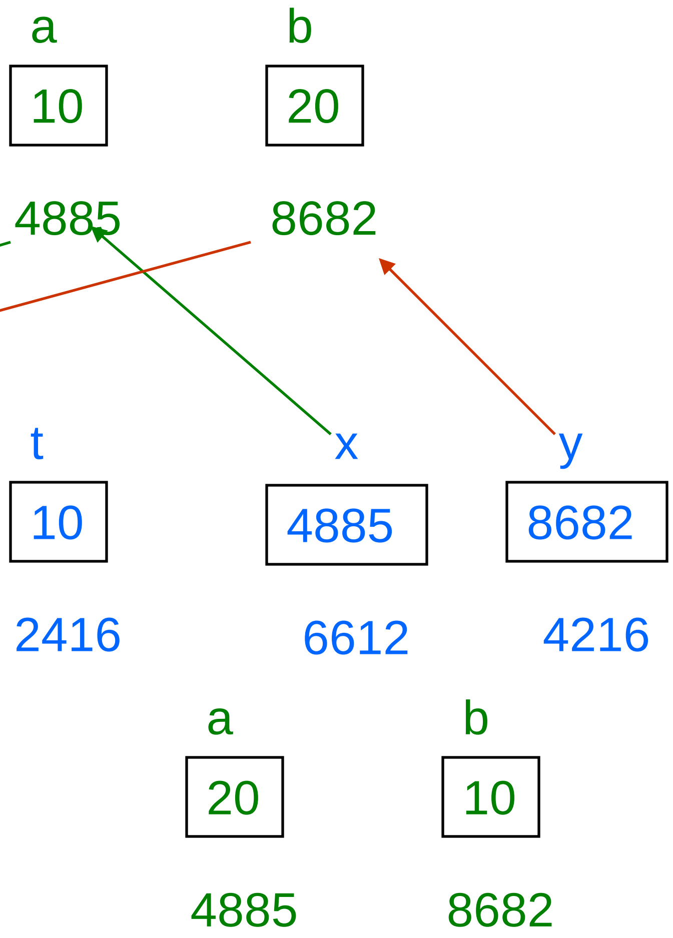
```
Address of a=4885
Address of a=4885
Address of b=3376
Value of b=4885
Value of a=2
Value of a=2
Value of a=2
```

Call By Reference

passed by reference

- temporary copy of the *address of argument* is provided to function
- called function can change value of the local variable in the calling function because it knows where in memory it is
- results in less independence between modules
- pass by reference simulated in C by using the address operator (**&**), an array name, or a pointer variable

```
 void swap(int *x,int *y);
 void main()
 {
      int a=10,b=20;
      swap(&a,&b);
      printf(“\n a=%d b=%d”,a,b);
}
 void swap(int *x,int *y)
 {
      int t;
      t=*x;
      *x=*y;
      *y=t;
}
```



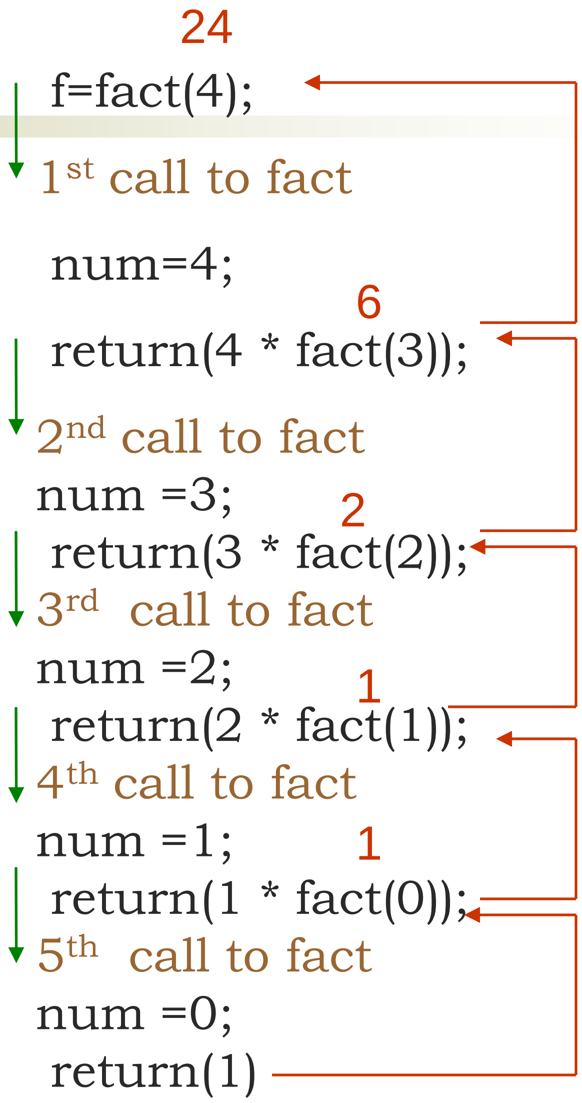
Recursion

- In C, a function that calls itself repeatedly, is called a recursion.
- Using recursion sometimes makes coding more straightforward; consider the calculation of *n*!,

Recursion Example

```
#include<stdio.h>
long int fact( unsigned int num);
void main();
{    long f;
     unsigned int n;
     printf(“Enter an integer number:”);
     scanf(“%u”,&n);
     f=fact(n);
     printf(“The factorial of a number is %ld \n”,f);
}
long int fact(unsigned int num)
{
      if(num==0)
         return(1);
      else
         return ( num * fact(num-1));
}
```

```
long int fact(unsigned int num)
{
     if(num==0)
        return(1);
     else
        return ( num * fact(num-1));
}
```



24

f=fact(4);

1st call to fact

num=4;

6

return(4 * fact(3));

2nd call to fact

num =3;

2

return(3 * fact(2));

3rd call to fact

num =2;

1

return(2 * fact(1));

4th call to fact

num =1;

1

return(1 * fact(0));

5th call to fact

num =0;

return(1)