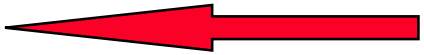
Lecture 12

Functions

Examining printMessage

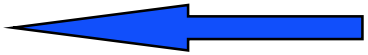
```
#include <stdio.h>

void printMessage ( void ) ;

int main ( void )
{
    printMessage ( ) ;
    return 0 ;
}
```

function **prototype**

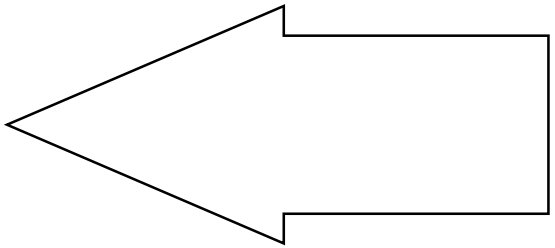


function call

```
void printMessage ( void )
{
    printf ("A message for you:\n\n") ;
    printf ("Have a nice day!\n") ;
}
```



function **header**

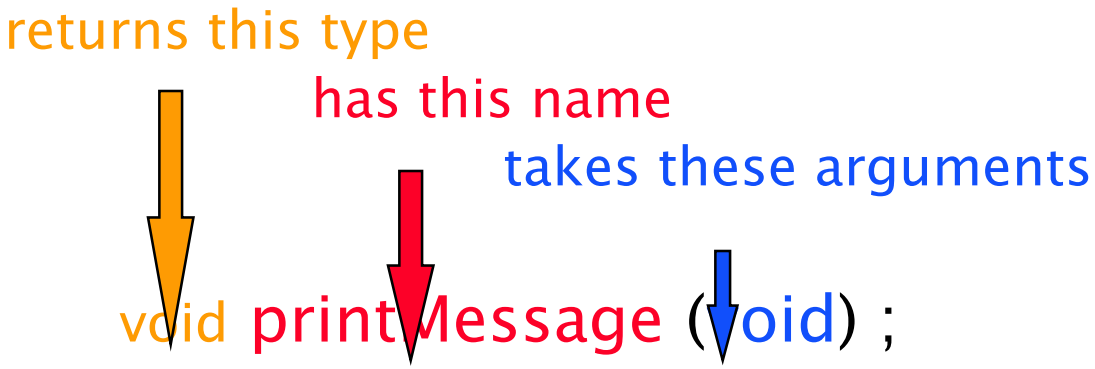


function **body**

function **definition**

The Function Prototype

- Informs the compiler that there will be a function defined later that:



- Needed because the function call is made before the definition -- the compiler uses it to see if the call is made properly

The Function Call

- Passes program control to the function
- Must match the prototype in name, number of arguments, and types of arguments

```
void printMessage (void) ;

int main ( void ) same name        no arguments
{
        printMessage ( ) ;
            return 0 ;
}
```

The Function Definition

- Control is passed to the function by the function call. The statements within the function body will then be executed.

```
void printMessage ( void )
{
        printf (“A message for you:\n\n”) ;
        printf (“Have a nice day!\n”) ;
}
```

- After the statements in the function have completed, control is passed back to the **calling function**, in this case main( ) . Note that the calling function does not have to be main( ) .

General Function Definition Syntax

type functionName (*parameter*$_1$, . . . , *parameter*$_n$)
{
  variable declaration(s)
  statement(s)
}

- If there are no parameters, either
  functionName( ) OR functionName(void)
  is acceptable.
- There may be no variable declarations.
- If the **function type** (**return type**) is void, a return statement is not required, but the following are permitted:
  return ; OR return( ) ;

Possibilities

- A function may:
  - Have no arguments and return nothing.
  - Have arguments and return nothing.
  - Have arguments and return one value.
  - Have no arguments and return one value.
- A function can never return more than one value!

Parameter List

- A function can have more than one parameter:

```
int functionA( int a, int b, float c )
```

- When the function is invoked, the parameters can be a variable, constant, or expression:

```
result = functionA( 7, 3 + a, dollars);
```

- The value 7 is put into to local variable a, 3 + a is put into local variable b, and a copy of the value in dollars is put into c in this example.

Using Parameters

```
void printMessage (int counter) ;
int main ( void )
{
    int num;
    printf (“Enter an integer: “) ;
    scanf (“%d”, &num) ;
    printMessage (num) ;
    parameter
      return 0 ;
}

void printMessage (int counter)
{
    int i ;
    for ( i = 0; i < counter; i++ )
    {
        printf (“Have a nice day!\n”) ;
    }
}
```

one argument matches the one **formal**

of type int of type int

Using Parameters (cont'd)

- In this example, we do not know in advance what the user will enter for the value of num, however, it is copied into the variable **counter** in the function printMessage( ).
- Both the variables **counter** and **i** are considered to be local variable, because they are only visible inside (or local to) the function printMessage( ).

Final “Clean” C Code

```
#include <stdio.h>

void printMessage (int counter) ;

int main ( void )
{
   int num ;      /* number of times to print message */

   printf (“Enter an integer: “) ;
   scanf (“%d”, &num) ;
   printMessage (num) ;

      return 0 ;
}
```

Pass by value

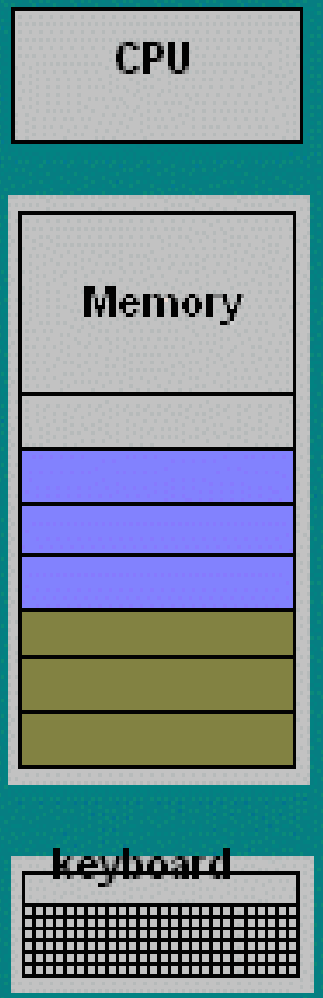
Arguments can be passed to a function by two methods, They are
pass by value
pass by reference

Function in C passes all arguments by value. When a single value is passed to a function via an actual argument, the value of the actual argument is copied into the function. Therefore, the value of the corresponding formal argument can be altered within the function, but the value of the actual argument within the calling routine will not change. This procedure for passing the value of an argument to a function is known as passing by value.

Pass by value

```
#include<stdio.h>
#include<conio.h>
Void interchange(int,int);
Void main()
{
Intx=50,y=70;
Interchange(x,y);
Printf("x=%d y=%d",x,y);
}
Voidinterchange(intx1,inty1)
{
Intz1;
Z1=x1;
X1=y1;
Y1=z1;
Printf("x1=%d y1=%d" ,x1,y1);
}
```

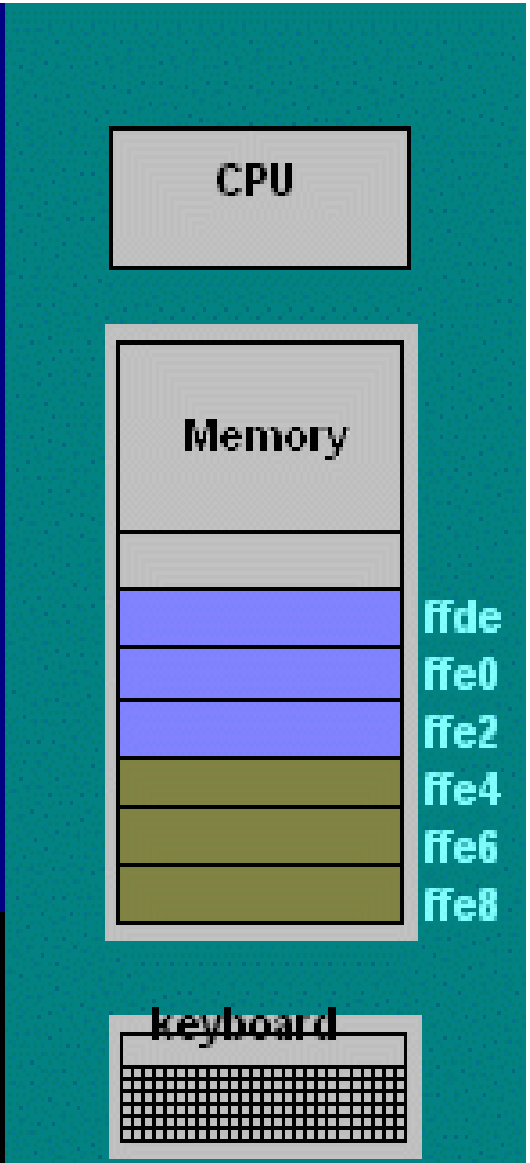
```c
#include  <stdio.h>
#include  <conio.h>
int add(int p,int q);
int main()
{
    int a,b,c;
    clrscr();
    printf("Enter two numbers\n");
    scanf("%d%d",&a,&b);
    c=add(a,b);
    printf("\nSum of %d and %d is %d",a,b,c);
    getch();
    return 0;
}
int add(int p,int q)
{
    int result;
    result=p+q;
    return(result);
}
```



Pass by Reference

When passing by reference technique is used, the address of the data item is passed to the called function. Using & operator we can determine the address of the data item. Note that function once receives a data item by reference, it acts on data item and the changes made to the data item also reflects on the calling function. Here you don't need to return anything to calling function.

```c
#include  <stdio.h>
#include  <conio.h>
void add(int *p,int *q,int *r);
void main()
{
     int a,b,c;
     clrscr();
     printf("Enter two numbers\n");
     scanf("%d%d",&a,&b);
     add(&a,&b,&c);
     printf("Sum of %d and %d is %d",a,b,c);
     getch();
}

void add(int *p,int *q,int *r)
{
     *r = *p + *q;
}
```



Recursion

Recursive functions are those functions, which call itself within that function. A recursive function must have the following type of statements.

- A statement to test and determine whether the function is calling itself again.
- A statement that calls the function itself and must be argument.
- A conditional statement (if-else)
- A return statement.

```c
#include  <stdio.h>
#include  <conio.h>
long int fact(int x)
{
     long int f;
     if(x==1)
          return(x);
     else
          f=x*fact(x-1);
      return(f);
}


int main()
{
    int n;
     clrscr();
     printf("Enter a number\n");
     scanf("%d",&n);
     printf("Factorial of %d is %ld",n,fact(n));
     getch();
     return 0;
}
```