Functions

Used to break problem down to small, bite sized pieces

Have an optional type of return value, a name and optional arguments

Functions return at most, ONE value

Functions must be prototyped or declared before usage

Call by Address

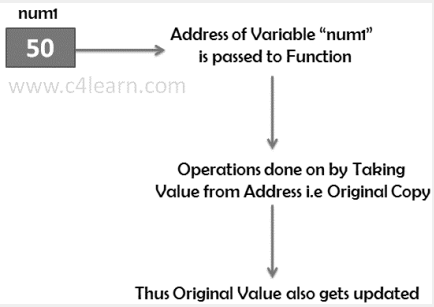
```
#include<stdio.h>
void interchange(int num1,int num2)
{
    int temp;
    temp = *num1;
    num1 = num2;
    *num2 = temp;
}
int main() {
    int num1=50,num2=70;
    interchange(&num1,&num2);
    printf("\nNumber 1 :
%d",num1);
    printf("\nNumber 2 :
%d",num2);
    return(0);
}
```

OUTPUT

Number 1 : 70

Number 2 : 50

Call Value



Extra Types

int	–2,147,483,648 to +2,147,483,647
unsigned int	0 to 4,294,967,295
int64	-9,223,372,036,854,775,808 to +9,223,372,036,854,775,807

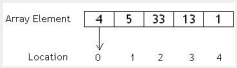
Switch

```
switch ( <condition> )
{
case <value> :
      <statements>
      [break; ]
case <value> :
      <statements>
      [break; ]
      :
[default:
      <statements>
      [break; ]]
}
```

CBR vs CBV

Point	Call by Value	Call by Reference
Copy	Duplicate Copy of Original Parameter is Passed	Actual Copy of Original Parameter is Passed
Modification	No effect on Original Parameter after modifying parameter in function	Original Parameter gets affected if value of parameter changed inside function

Array Sample



a[0] = 4;
a[1] = 5;
a[2] = 33;
a[3] = 13;
a[4] = 1;

Call-By-Value Steps

Copy of original parameter is created & passed to the called function

Updates inside method will NOT affect the original value of the variable in the calling function

scope is limited, therefore it cannot alter values inside main function

Call by Value

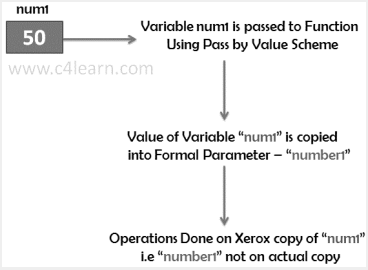
```
#include<stdio.h>
void interchange(int number1,int
number2)
{
    int temp;
    temp = number1;
    number1 = number2;
    number2 = temp;
}
int main() {
    int num1=50,num2=70;
    interchange(num1,num2);
    printf("\nNumber 1 :
%d",num1);
    printf("\nNumber 2 :
%d",num2);
    return(0);
}
```

OUTPUT

Number 1 : 50

Number 2 : 70

CallRef



Call by ref = call by address