# Lecture 5 Details – Functions (pass by value & ref.)

1. Prototypes, Function calls, Function Definitions
2. Passing Arguments:
   - Pass by value
   - Pass by reference
   - how arrays work when we pass
3. Programming Example

Announcements

# Review Functions

1. Function Prototypes:

```
return_type function_name (argument_list);
```

* write a prototype for a function that returns nothing and takes an integer as an argument:

```
void calculate (int arg );
```

* returns an int and has no arguments:

```
int calculate ();
```

Prototypes are not required if functions are defined before they are called!

# Function Call

1. Do [NOT] put the data types in the function calls !

```
int value = 100;
calculate (value);
```

← the contents is copied with "pass by value"

```
value = calculat();
```

← 2nd prototype (previous page)

2. what is this?

```
int calculate();
```

why doesn't this do anything?

# Function Definition - Implementation

* functions that have "non-void" return types then the functions need to return a value
* All paths through the function must return a value.
* Once a function is declared or defined, it may be called.
* the arguments & return types must match the prototypes

```
int calculate()
{
    int num1, num2;
    cout << "please enter 2 numbers: ";
    cin >> num1 >> num2;
    return num1 + num2;
}
```

# Arguments

1. Pass by Value

— a copy of the argument is made

— any changes made in the function to the argument will not be detected outside of the function

```
int main()
{
    int num = 100;
    func(num);
```

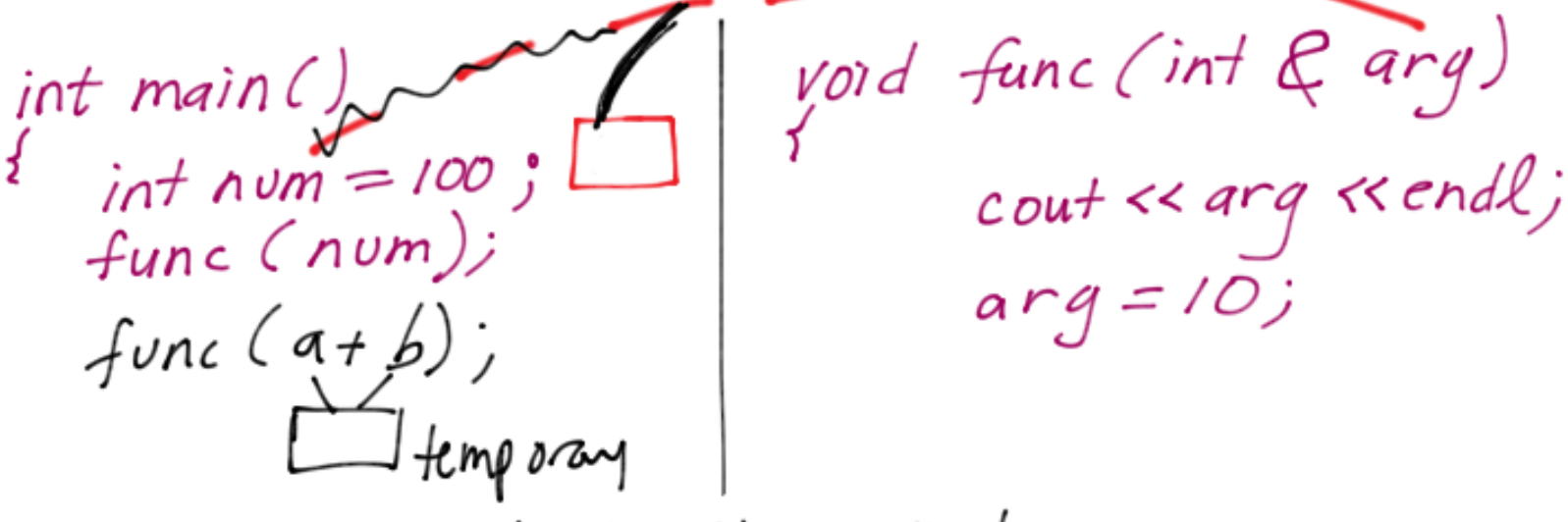
```
void func(int value)
{
    int junk;
    cout << value << endl;
    value = 10;
```


↑ NO AFFECT ON main's number

* Think of an argument passed by value as one that is a "local" variable inside the called function with an initial value from the call.

# Pass by Reference

- creates an alias
- the "address of" the calling routine's value is implicitly sent to the function
- Allows us to "Get Information" from a function without the overhead of returning it

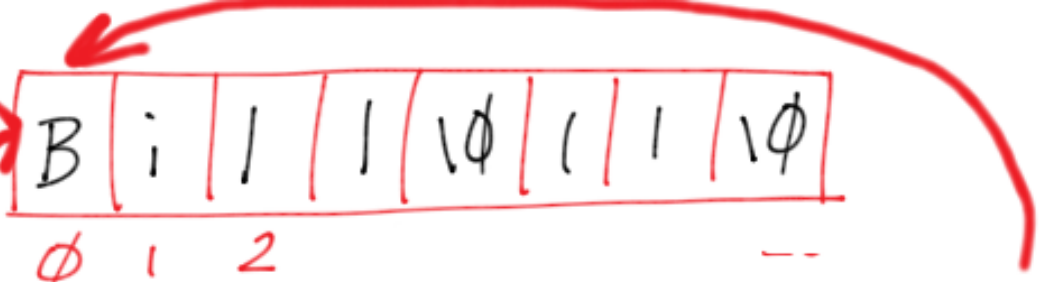



Any changes made in the called function immediately affect the calling routine's value

(can't be used for passing literals (numbers) or in this case constants)

# Passing Arrays

* Technically, when an array is an argument, the starting address to the first element is actually passed by value
* This seems like pass by reference because the contents of the array can be altered by the called function
* It is not possible to pass an array by reference (with the &) because that would mean that we the starting address or location of the array could be altered!




```
int main()
{
    char name[21];
    cout << "Enter a name";
    cin.get(name, 21);
    cin.ignore(100, '\n');

    func(name);
    cout << name;
```

pass an array:

```
void func(char name[])
{
    cout << array << endl;

    // what about:
    cout << "Re-enter: ";
    cin.get(array, 21);
    cin.ignore(100, '\n');
    cout << array;
```

# Create a Function to Prompt and read in an array.

- Pass in the prompt
- "Get back" thru the argument list the info
- send in the max # characters so this can work with different sized arrays

```
void read(char prompt[], int size, char result[]);
```

never use a & with []

pass by value

call

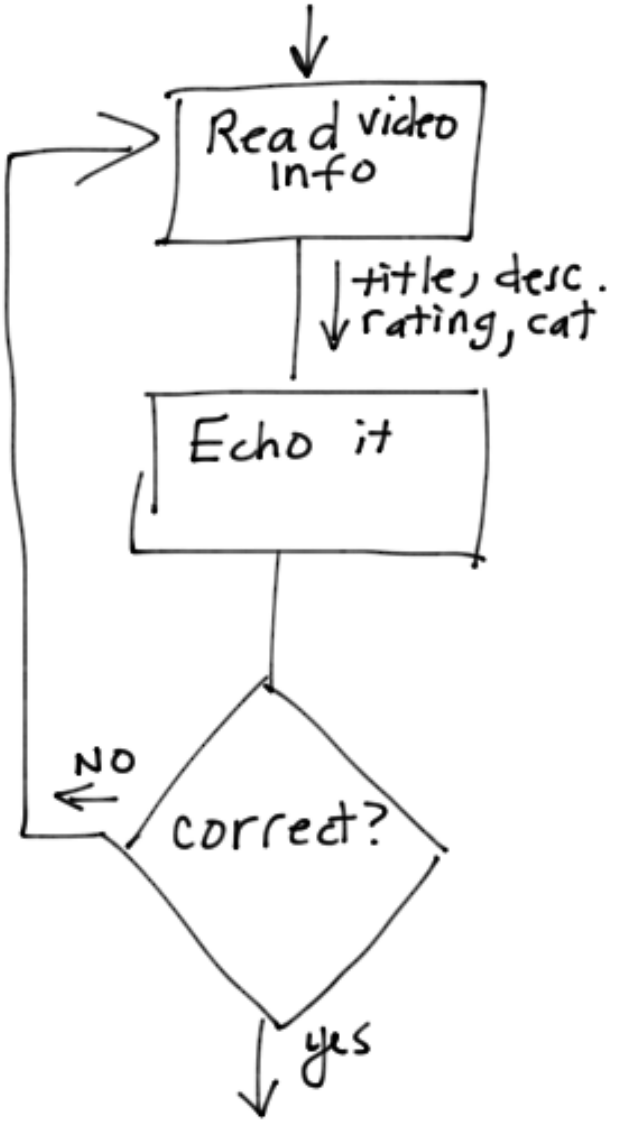
```
char phrase[131];
read("Entra phrase", 131, phrase);
```

function body

```
cout << prompt << ": ";
cin.get(result, size);
cin.ignore(100, '\n');
```

# Create a Program to mange videos online

- A video will have a title, description, rating, category
  - title: 131 chars
  - description: 300 char max
  - rating: 6 chars
  - category: 16




```
char title [131];
char description [300];
char rating [6];
char category [16];

do
{
        readall (title, description,
                 rating, category);

        displayall (title, description,
                    rating, category);
} while (!correct());
```

```
bool correct()
{
  char response = 'n';
  cout << "Is this corred? Y,N";
  cin >> response;
  if (toupper(response) == 'n')
        return false;
  return true;
}
```

Returns true when the user is satisfied

---

```
void readall(char n[], char d[], char r[], char c[])
{
    read("Enter the title, 131, n);
    read("Description", 300, d);
    read("Rating", 6, r);
    read("Category", 16, c);
}
```

pick better names!

Uses other functions to assist!

# Preview for Next Lecture

Group the information as members in a structure:

```
struct video          <- "tag" name
{
    char title [131];      <- use a constant!
    char description [300];
    char rating [6];
    char category [16];
};                    <- important!
```

main

```
video show;

do
{
    readall (show);
    displayall (show);
} while (! correct ());
```

```
void readall (video & s)
{
    read ("Title", 131, s.title);
}
```

object of a structure

member

Direct member access operator