Introduction

- **Understand “Scope” of an Identifier**
- **Know the Storage Classes of variables and functions**

Example - Call-by-Value

1. Problem Definition

   Write a function “swap” that has two integer input arguments. This function should swap the values of the variables.

2. Refine, Generalize, Decompose the problem definition

   (i.e., identify sub-problems, I/O, etc.)

   Input = two integers

   Output= no value is returned to the calling function, but the values of the called variables should be swapped.

3. Develop Algorithm

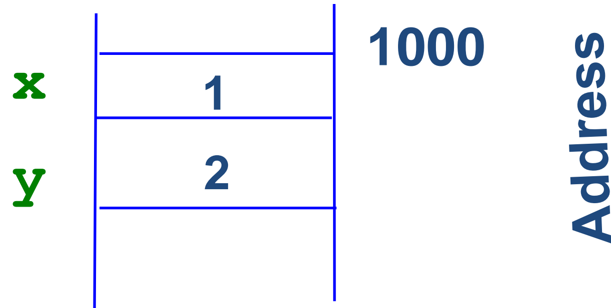
```
temp = a;
a = b;
b = temp;
```

Example - Bad Swap !!!

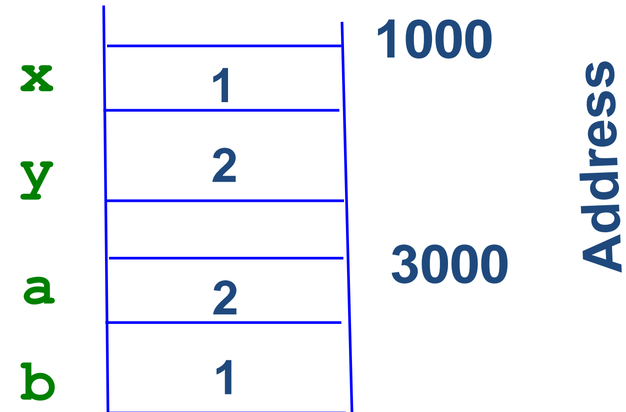
```
/*  C Function to swap values. */
#include <stdio.h>

void swap(int, int); /* prototype for swap */

void main(void) /* function header */
{
  int x = 1;
  int y = 2;
  swap(x,y);
  printf("x = %i , y = %i \n",x,y); /* prints x = 1 , y = 2 */
}

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

Why swap doesn't work.

Main Memory before call



Main Memory while executing swap, but before the return statement



The swap function will successfully swap the values of the variables a and b.

But, since non-arrays are passed by value the swap of the values for a and b will not have any affect on the variables x and y.

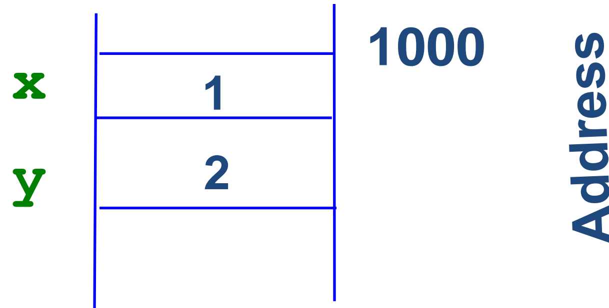
It is important to note that even if we had used the variable names x and y rather than a and b the swap function would still not work. See the next slide...

Example - Bad Swap !!!

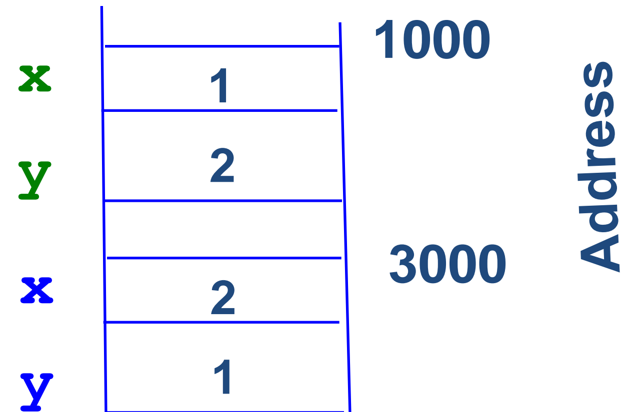
```
/*  Modified function to swap values. */
#include <stdio.h>

void swap(int, int);

void main(void)
{
  int x = 1;
  int y = 2;
  swap(x,y);
  printf("x = %i , y = %i \n",x,y); /* prints x = 1 , y = 2 */
}

void swap(int x , int y) /* we now use x and y */
{
  int temp = x;
  x = y;
  y = temp;
  return;
}
```

Why the modified swap doesn't work.

Main Memory before call



Main Memory while executing swap, but before the return statement



The C compiler will keep track of the variables x and y declared in main and the variables x and y declared in swap. The x in main is a different variable than the x in swap. Similarly for the y variable.

The reason that the x variable in main is different than the x variable in swap is that two declarations of x occur in different blocks. That is the scope of the x in main is different than the scope of x in swap.

One solution to the problem is to use pointer variables. We will discuss pointer variables in a future lecture.

Scope of Identifiers

An identifier is a name that is composed of a sequence of letters, digits, and the '_' underscore character.

Variable names are identifiers and so are function names and symbolic constant names.

The scope of an identifier is the portion of the program in which the identifier is visible or accessible.

Both variables and functions have two attributes: type and storage class. The scope of a variable or function is related to its storage class.

Scope of Variable Names

•<u>Local Variables</u> - are declared within a block and cannot be referenced by outside the block; i.e., each function is assigned its own "local" storage areas.

•<u>Global Variables</u> - declared outside any block and are known to all blocks in the same file . By default, global variables are "static" storage class.

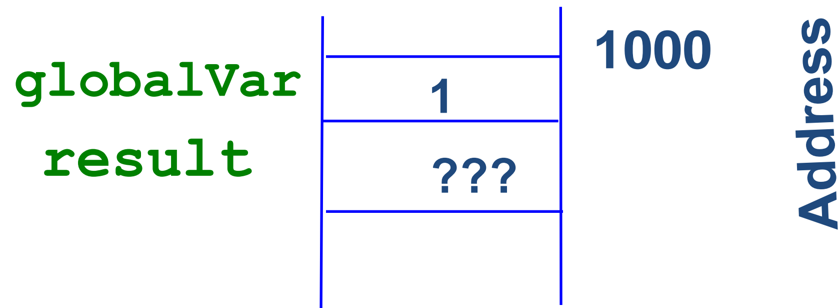
In the examples on the following slides, drawing a picture of memory is helpful in understanding how scoping works.

Example1: Scope of Variables

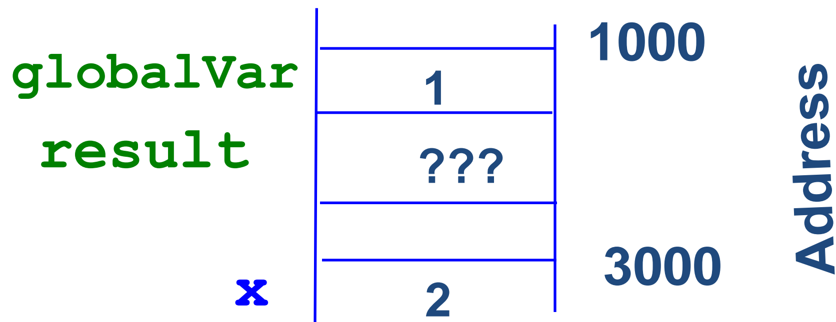
```
int globalVar = 1;                        /* global variable */
int myFunction(int);          /* function prototypes */
void main(void) /* local variable: result, in main */
{
  int result;
  result = myFunction(globalVar); /* call myFunction */
  printf("%i",result); /* prints "2" */
  printf("%i",globalVar); /* prints "2" */
  printf("%i",x);  /* compile error! x not known in main */
}
int myFunction(int x) /* Local variable x */
{
   ++x;
   printf("%i",x);   /* prints value "2" */
   printf("%i",globalVar); /*prints value"1" */
   ++globalVar;
   return x;
}
```

Example1: Scope of Variables

Main Memory before call to `Myfunction`



Main Memory while executing `MyFunction`, but before `++globalVar;`



Example1: Scope of Variables

Main Memory while executing `MyFunction`, but before `return(x);`

globalVar	2	1000
result	???	
x	2	3000

Address

Main Memory after call to `Myfunction`

globalVar	2	1000
result	2	

Address

Example2: Scope of Variables

```
int globalVar = 1;                           /* global variable */

int myFunction(int);          /* function prototypes */

void main(void)
{
   int result;
   result = myFunction(globalVar); /* call myFunction */
   printf("%i",result); /* prints "2" */
}


int myFunction(int x) /* Local variables: x, globalVar */
{
  int globalVar;              /* new "local" variable */
  printf("%i\n",globalVar);/* prints ??? */
  return x + 1;
}
```

Example3: Scope of Variables

```
int myFunction(int);          /* function prototypes */

void main(void) /* local variables: x,result in main */
{
  int result, x = 2;
  result = myFunction(x);   /* call myFunction */
  printf("%i",result); /* prints “3” */
  printf("%i",x); /* prints “2” WHY???  */
}


int myFunction(int x) /* Local variable: x */
{
   x = x + 1;
   printf("%i\n",x);  /* prints “3” */
   return x;
}
```

Example4: Scope of Variables

```
#include <stdio.h>
int x;  /* x is a global variable */
int y;  /* y is a global variable  */

void swap(int, int); /* prototype for swap */

void main(void)
{
  x = 1;   /* x and y are not declared here!!! */
  y = 2;
  swap(x,y);
  printf("x = %i , y = %i \n",x,y); /* prints x = 1,y = 2 */
}

void swap(int x , int y)
{
  int temp = x;
  x = y;
  y = temp;
  return;
}
```

Scope of Function Names

A function can be called by any other function, provided that either the function definition or its prototype is in the same file as the calling function and precedes the function call.

If no prototype is specified for a function, its header serves as the function prototype.

Note: Functions cannot be defined within each other

Storage Classes - auto

- Determines the storage duration and scope of identifiers, and also linkage between files.

auto - creates storage for variables when the block that declares them is entered, and deletes the storage when the block is exited. Local variables have "auto" storage by default.

e.g.,typing auto float a, b, c; is equivalent to typing float a, b, c;

Storage Classes - static

static - creates and initializes storage for variables when the program begins execution. Storage continues to exist until execution terminates. If an initial value is not explicitly stated, a static variable is initialized to 0. We can retain values of local variables by declaring them to be static.

In the following example, the static variable sum is initialized to 1.

```
static int sum = 1;
```

The initialization takes place only once. If the above declaration appears in a user defined function , the first time the function is called, the variable sum is initialized to 1. The next time the function (containing the above declaration) is executed sum is not reset to 1.

Storage Classes - extern

extern - used to reference identifiers in another file. Function names are extern by default.

e.g., extern int foreignVar;

Example - Static variable

1. Problem Definition

   Write a function “sum”that keeps a running total of the sum of every value passed to “sum”. To test this function, modify the previous program of Lecture 15 -3 that compute the average of values in a file.

2. Refine, Generalize, Decompose the problem definition

   (i.e., identify sub-problems, I/O, etc.)

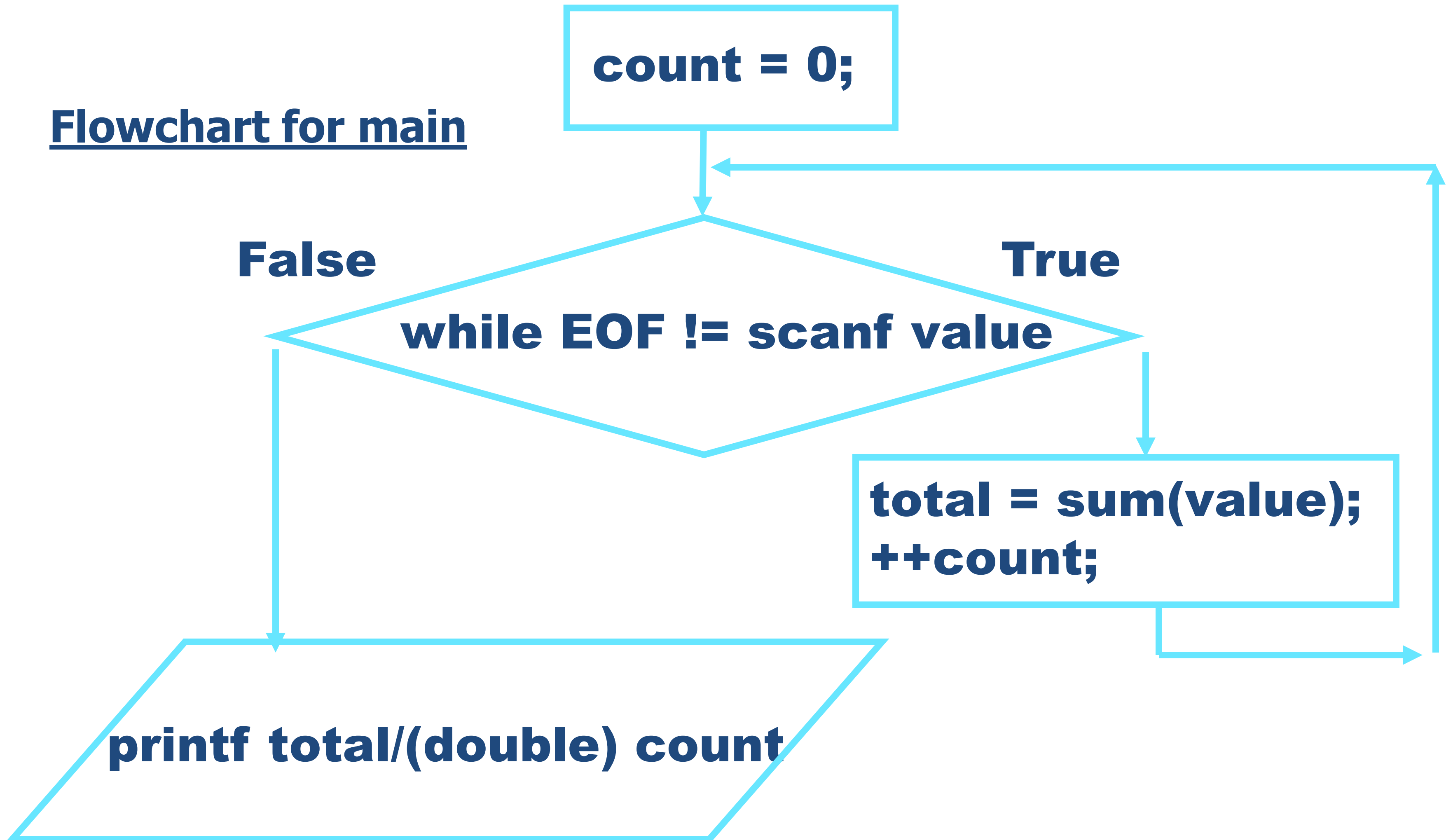
   Input = integers from file “input.dat”

   Output=real number representing the arithmetic average (sum of values)/(count of number of values)

3. Develop Algorithm

   (processing steps to solve problem)

Example - Static variable



Example - Static variable

```
#include <stdio.h>

int sum(int); /* function protoype */

void main(void)

{
  int value,total,count = 0;

  while (EOF != scanf("%i", &value)) /*  read value   */

  {
       total = sum(value);
       ++count;
  } /* end of while loop */
  printf("Average of the %i numbers = %f \n",count,total/(double)count);
}

int sum(int val) /* function header */

{
  static int total_val = 0;   /* static variable, */

  total_val += val;

  return total_val;
}
```