ESC101: Introduction to Computing

f(unction)

Major quiz on Wednesday

- L20
- L17
- L7
- KD 101
- KD 102

- Seating Odd Row Odd Seat (OROS)
- Please check your room number from list on canvas

```
int max (int a, int b) {
    if (a > b)
        return a;
    else
        return b;
}
```

Return Type

Function Name

2 arguments
a and b,
both of type int.
(formal args)

```
int main () {
    int x;
    x = max(6, 4);
    printf("%d",x);
    return 0;
}
```

Body of the function, enclosed inside { and } (mandatory) returns an int.

Call to the function.
Actual args are 6 and 4.

Nested Function Calls

- Functions can call each other
- A declaration or definition (or both) must be visible before the call
  - Help compiler detect any inconsistencies in function use
  - Compiler warning, if both (decl & def) are missing

```
#include<stdio.h>
int min(int, int); //declaration
int max(int, int); //of max, min

int max(int a, int b) {
   return (a > b) ? a : b;
}

// a "cryptic" min, uses max
int min(int a, int b) {
   return a + b – max (a, b);
}

int main() {
  printf("%d", min(6, 4));
}
```

Practice Problem

- Write a function that simulates a bank account. Allow operations Deposit ‘d’ *amount* Withdraw ‘w’ *amount* and add interest at a fixed rate of 10%
- Sample Input format : *InitialAmount* d *amount1* w *amount2*
- Input: 1000 d 100 w 200 i d 300 w 100
- Output: amount = 1190

Solution for Practice problem

```
#include <stdio.h>
int interest(int amnt);
int deposit( int amnt, int sum);
int withdraw( int amnt, int sum);
int main()
{
   int amnt, c, n;
   scanf("%d",&amnt);
   getchar(); // skip a space after reading amount
   while( (c = getchar() )!=EOF)
   {
   }
   printf("amount = %d\n”,amnt);
   return 0;
}
```

Solution for Practice problem

```
#include <stdio.h>
int interest(int amnt);
int deposit( int amnt, int sum);
int withdraw( int amnt, int sum);
int main()
{
   int amnt, c, n;
   scanf("%d",&amnt);
   getchar(); // skip a space after reading amount
   while( (c = getchar() )!=EOF)
   {
      switch(c)
      {
         case 'd': scanf("%d",&n);
                   amnt=deposit(amnt, n);
                   getchar(); break; //skip a space after reading amount
     }
   }
   printf("amount = %d\n",amnt);
   return 0;
}
```

Solution for Practice problem

```
#include <stdio.h>
int interest(int amnt);
int deposit( int amnt, int sum);
int withdraw( int amnt, int sum);
int main()
{
   int amnt, c, n;
   scanf("%d",&amnt);
   getchar(); // skip a space after reading amount
   while( (c = getchar() )!=EOF)
   {
      switch(c)
      {
         case 'd': scanf("%d",&n);
                   amnt=deposit(amnt, n);
                   getchar(); break; //skip a space after reading amount
         case 'w': scanf("%d",&n);
                   amnt=withdraw(amnt, n);
                   getchar();  break; //skip a space after reading amount
         case 'i': amnt=interest(amnt);
                   getchar();  break; //skip a space after reading amount
         default: printf("invalid input\n"); break;
      }
   }
   printf("amount = %d\n",amnt);
   return 0;
}
```

Solution for Practice problem

```
int interest(int amnt)
{
   return (amnt+amnt*10/100.0);
}


int deposit( int amnt, int sum)
{
   return (amnt+sum);
}

int withdraw( int amnt, int sum)
{
   //can have additional checks for seeing amnt is not negative
   return (amnt -sum);
}
```

Predefined Functions

- C has many predefined functions. We have seen scanf, printf.
- To use a predefined function, the corresponding header file must be included.
  - Mathematical functions defined in the library math.h.
  - Input/Output functions in stdio.h
  - String functions in string.h

Some predefined math functions

	Description
double fabs(double x)	absolute value
double cos(double x)	cosine
double sin(double x)	sine
double tan(double x)	tan
double exp(double x)	e^x
double log(double x)	Natural log, x > 0
double log10(double x)	Log base 10, x > 0
double pow(double x, double y)	x^y
double sqrt(double x)	$\sqrt{x}$, x >= 0
double floor(double x)	largest integral value <= x
double ceil(doublex)	smallest integral value >= x

Avoiding Common Errors

- Declare functions before use.
- Argument list of a function:
  - Provide the required number of arguments,
  - Check that each function argument has the correct type (or that conversion to the correct type will lose no information).

Question

```
// swapping a and b
void swap(int a, int b){
   int temp;
   temp = a;
   a = b;
   b = temp;
   printf("a=%d b=%d\n", a, b);
}
int main(){
   int  a=10, b=15;
   printf("a=%d b=%d\n", a, b);
   swap(a, b);
   printf("a=%d b=%d\n", a, b);
   return 0;
}
```

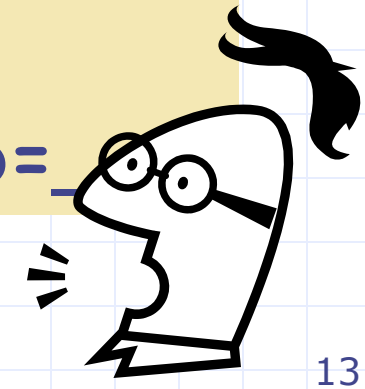
What is the output of the program? (fill the blanks)

OUTPUT

a=10 b=15

a=15 b=10

a=___ b=___



Scope of a Name

- Functions allow us to divide a program into smaller parts
  - each part does a well defined task
- There are other ways to *partition a program*
  - *Statement blocks, Files*
- Scope of a name is the part of the program in which the name can be used

Scope of a Name

- Two variables can have the same name only if they are declared in separate scopes.
- A variable can not be used outside its scope.
- C program has
  - function/**block** scope
  - **file** scope
  - **global**/external scope

Scope Rules: Functions

- The scope of the variables present in the argument list of a function's definition is the body of that function.
- The scope of any variable declared within a function is the body of the function.

scope of m1, a1, b1

```
int max(int a, int b) {
    int m = 0;
    if (a > b) m = a;
    else m = b;
    return m;
}
```

scope of m2, a2, b2

```
int min(int a, int b) {
    int m = 0;
    if (a < b) m = a;
    else m = b;
    return m;
}

int main() { … }
```

Scope Rules : Blocks

- For an identifier declared at the head of a block
  - Scope begins at the end of declaration
  - Scope ends at the end of the block

```
int main() {
   int m = 5;
   {
       float m = 6.5; // shadows
       printf("%f", m);  // prints 6.5
   }
   printf("%d" , m); // prints 5
}
```

Scope of a Name

- Scopes can be nested
  - A name in inner scope *"shadows"* the same name, if present in outer scope

```
int main() {
   int m = 5;
   {
        float m = 6.5; // shadows
        printf("%f", m);  // prints 6.5
   }
   printf("%d" , m); // prints 5
}
```

Global Variable

- Variable declared outside every function definition
- Can be accessed by all functions in the program that follow the declaration
- Also called *External* variable
- What if a variable is declared inside a function that has the same name as a global variable?
  - The global variable is "shadowed" inside that particular function only.

Global Variables

```
#include<stdio.h>
int g=10, h=20;

int add(){
  return g+h;
}

void fun1(){
  int g=200;
  printf("%d\n",g);
}

int main(){
  fun1();
  printf("%d %d %d\n",
          g, h, add());
  return 0;
}
```

```
200
10 20 30
```

1. The variable g and h have been defined as global variables.
2. The use of global variables is normally discouraged. Use local variables of functions as much as possible.
3. Global variables are useful for defining constants that are used by different functions in the program.

Global Variables: example

```
const double PI = 3.14159;
double circum_of_circle(double r) {
   return 2 * PI * r; }
double area_of_circle (double r) {
   return PI * r * r;
}
```

defines PI to be of type double with value 3.14159. Qualified by const, which means that PI is a constant. The value inside the box associated with PI cannot be changed anywhere.

Constants via #define

```
#define PI  3.14159
double circum_of_circle(double r) {
   return 2 * PI * r; }
double area_of_circle (double r) {
   return PI * r * r;
}
```

replaces PI by the constant 3.14159 everywhere.

#define *name replacement-text*

Static Variables

- We have seen two kinds of variables: local variables and global variables.
- There are static variables too.

```
int f () {
  static int ncalls = 0;
  ncalls  = ncalls + 1;
/* track the number of
times f() is called  */
  … body of f() …
}
```

- Use a local variable?
  - gets destroyed every time f returns
- Use a global variable?
  - other functions can change it! (dangerous)

GOAL: count number of calls to f()

SOLUTION: define ncalls as a static variable inside f().

It is created as an integer box the first time f() is called.

Once created, it never gets destroyed, and retains its value across invocations of f().

It is like a global variable, but visible only within f().

Static variables are not allocated on stack. So they are not destroyed when f() returns.

SumDigits problem: What is the output for 99

```
#include <stdio.h>
int sumDigits(int n);
int main()
{

   int sumD;
   int n;
   scanf("%d",&n);
   int flag=0;
   while( n >= 9)
   {
      sumD = sumDigits(n);
      if(sumD == 9) {
         flag = 1;
         break;
      }
      n = sumD;
   }
   if(flag == 1)
      printf("Yes\n");
   else
      printf("No\n");
   return 0;
}
```

```
int sumDigits(int n)
{
   int sum=0;
    while(n>0)
   {
      sum = sum + n%10;
      n = n/10;
   }
   return sum;
}
```

SumDigits problem: What is the output for 99

```
#include <stdio.h>
int sumDigits(int n);
int main()
{

   int sumD;
   int n;
   scanf("%d",&n);
   int flag=0;
   while( n >= 9)
   {
      sumD = sumDigits(n);
      if(sumD == 9) {
         flag = 1;
         break;
      }
      n = sumD;
   }
   if(flag == 1)
      printf("Yes\n");
   else
      printf("No\n");
   return 0;
}
```

```
int sumDigits(int n)
{
   int sum=0;
   static int cnt=0;
   cnt=cnt+1;
   printf("sumDigits called %d
  times\n",cnt);
   while(n>0)
   {
      sum = sum + n%10;
      n = n/10;
   }
   return sum;
}
```

SumDigits problem: What is the output for 99

```
#include <stdio.h>
int sumDigits(int n);
int main()
{

   int sumD;
   int n;
   scanf("%d",&n);
   int flag=0;
   while( n >= 9)
   {
      sumD = sumDigits(n);
      if(sumD == 9) {
         flag = 1;
         break;
      }
      n = sumD;
   }
   if(flag == 1)
      printf("Yes\n");
   else
      printf("No\n");
   return 0;
}
```

```
int sumDigits(int n)
{
   int sum=0;
   static int cnt=0;
   cnt=cnt+1;
   printf("sumDigits called %d
  times\n",cnt);
   while(n>0)
   {
      sum = sum + n%10;
      n = n/10;
   }
   return sum;
}
```

Output

SumDigits called 1 times
SumDigits called 2 times
Yes

ClassQuiz: Global variables

```
#include <stdio.h>
static int shared = 3; //file scope
int shared2; //external scope
void changeShared() {   shared = 5;}
void localShadow() { int shared = 1000;}

void paramShadow(int shared) {
        shared = -shared;}

int main()      {
        printf("%d\n", shared);
        changeShared( );
        printf("%d\n", shared);
        localShadow();
        printf("%d\n", shared);
        paramShadow(1);
        printf("%d\n", shared);
        return 0;           }
```

What is the output of the

OUTPUT

3
5
5
5

Courtesy: Wikipedia

Summary

- Global Variable
  - Visible everywhere
  - Lives everywhere (never destroyed)
- Local Variable
  - Visible in scope
  - Lives in scope (destroyed at the point where we leave the scope)
- Static Variable
  - Visible in Scope
  - Lives everywhere! (but can not be accessed outside scope)

Major quiz

- L20
- L17
- L7
- KD 101
- KD 102

Best of Luck for the Major Quiz