C Programming Primer

Objectives

- Develop a functional knowledge of C programming concepts

- Understand basic variables, constructs, and control flow statements

Special Notes

- Copying and pasting code from these slides can be problematic. It is best to look at these slides and type directly into your programming environment when developing your own programs.

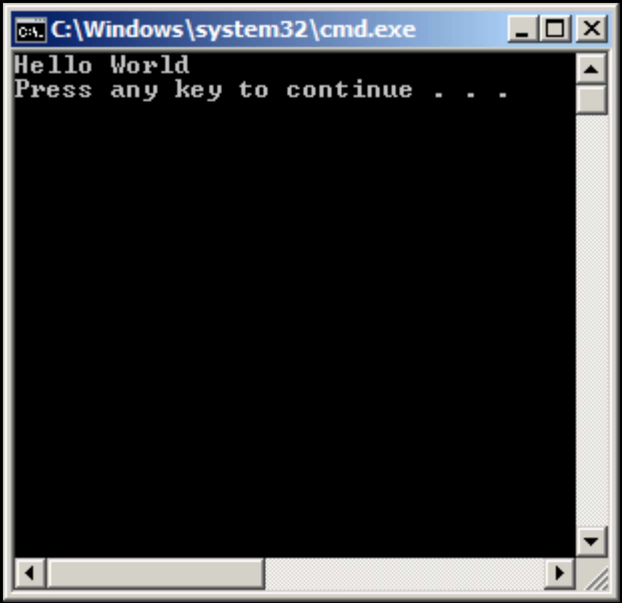
What is C?

- Programming language created between 1969 & 1973 by Dennis Ritchie
- Written to create the UNIX operating system
- Popular programming language for malware authors

Basic Structure of C Program

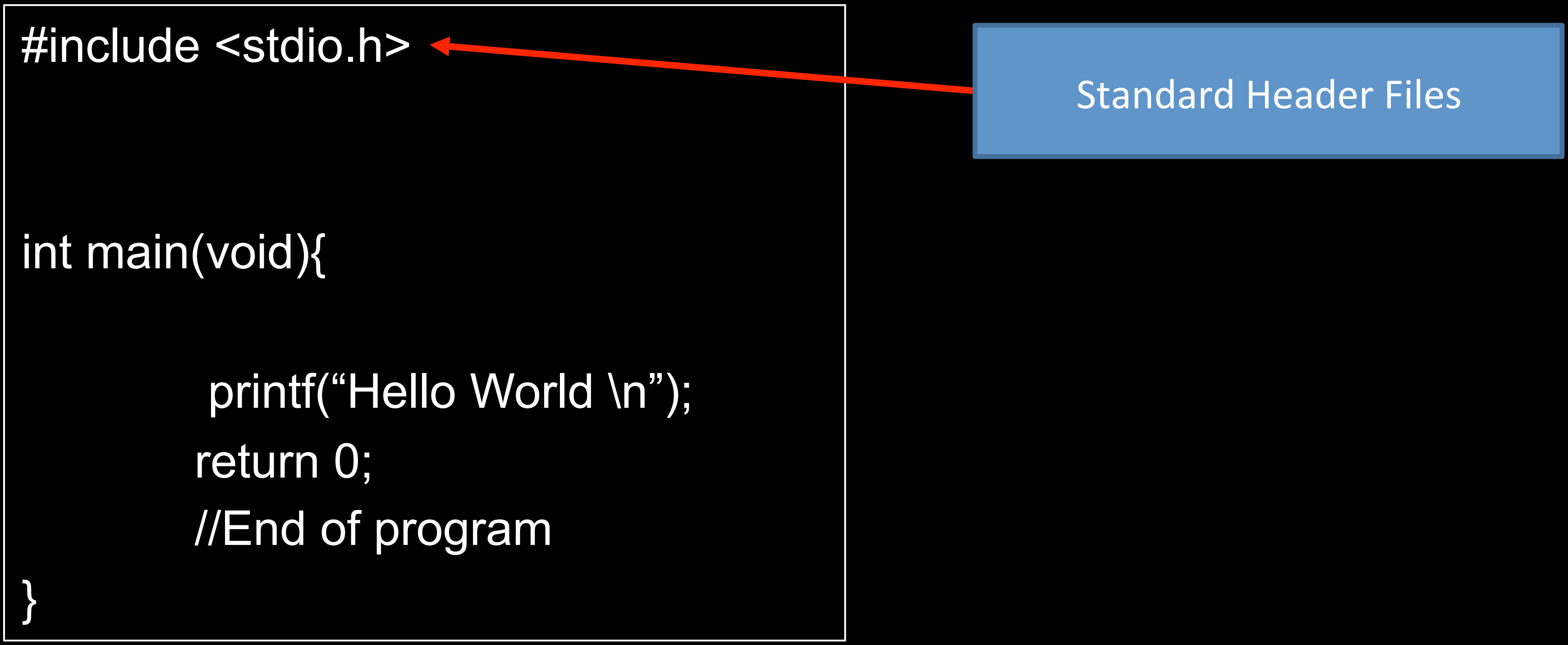
```
#include <stdio.h>

int main(void){

              printf(“Hello World \n”);
             return 0;
}
```

- This program will print the following output to a console screen:



```
Hello World
Press any key to continue . . .
```

Basic Structure of C Program



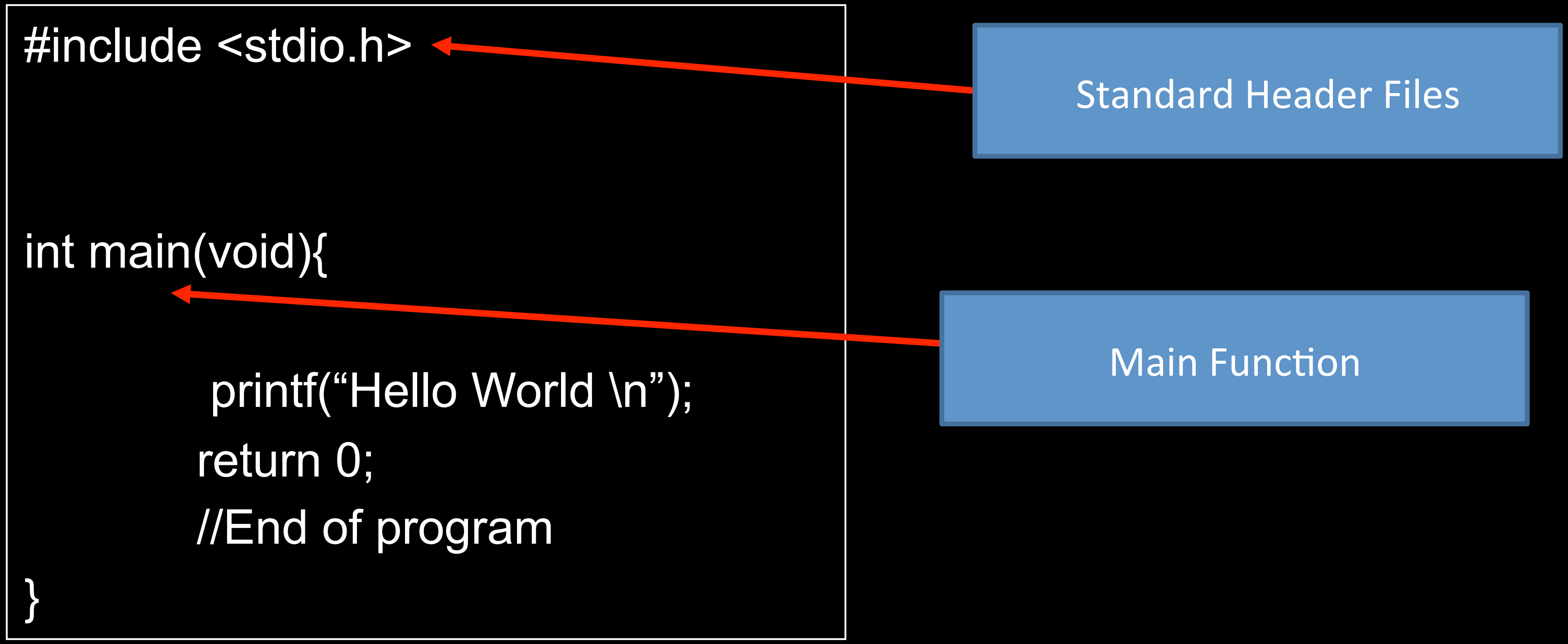
```
#include <stdio.h>


int main(void){

         printf("Hello World \n");
        return 0;
        //End of program
}
```

Standard Header Files

Standard Header Files

- Header file contains one or more function declarations
- Gives access to previously created functions
- Any number of standard headers can be included
- stdio.h – provides access to various functions that allow input and output operations
  - printf function

Basic Structure of C Program



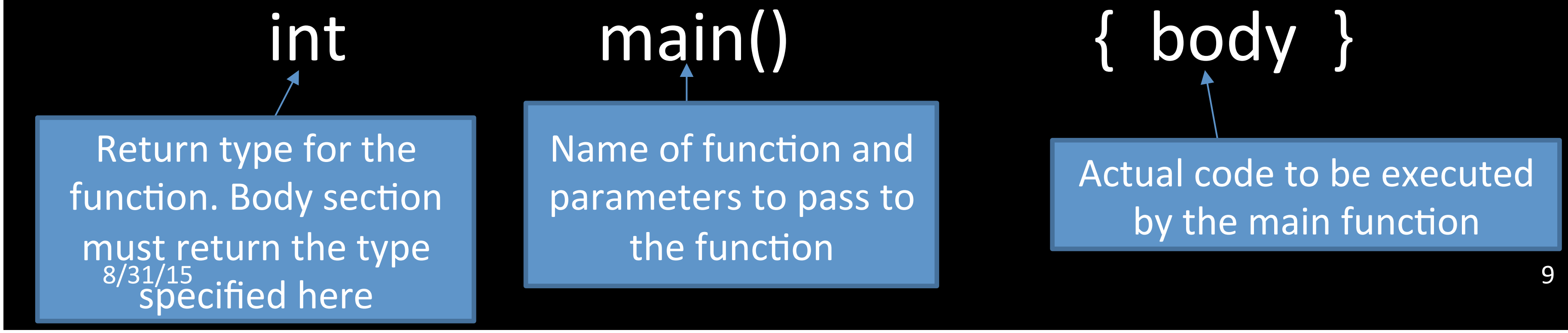
```
#include <stdio.h>


int main(void){

             printf(“Hello World \n”);
            return 0;
            //End of program
}
```

Standard Header Files

Main Function

Main Function

- Global function that designates the start of a program
- Every C program must have a main function
- Function may contain slight variations but will always have following structure



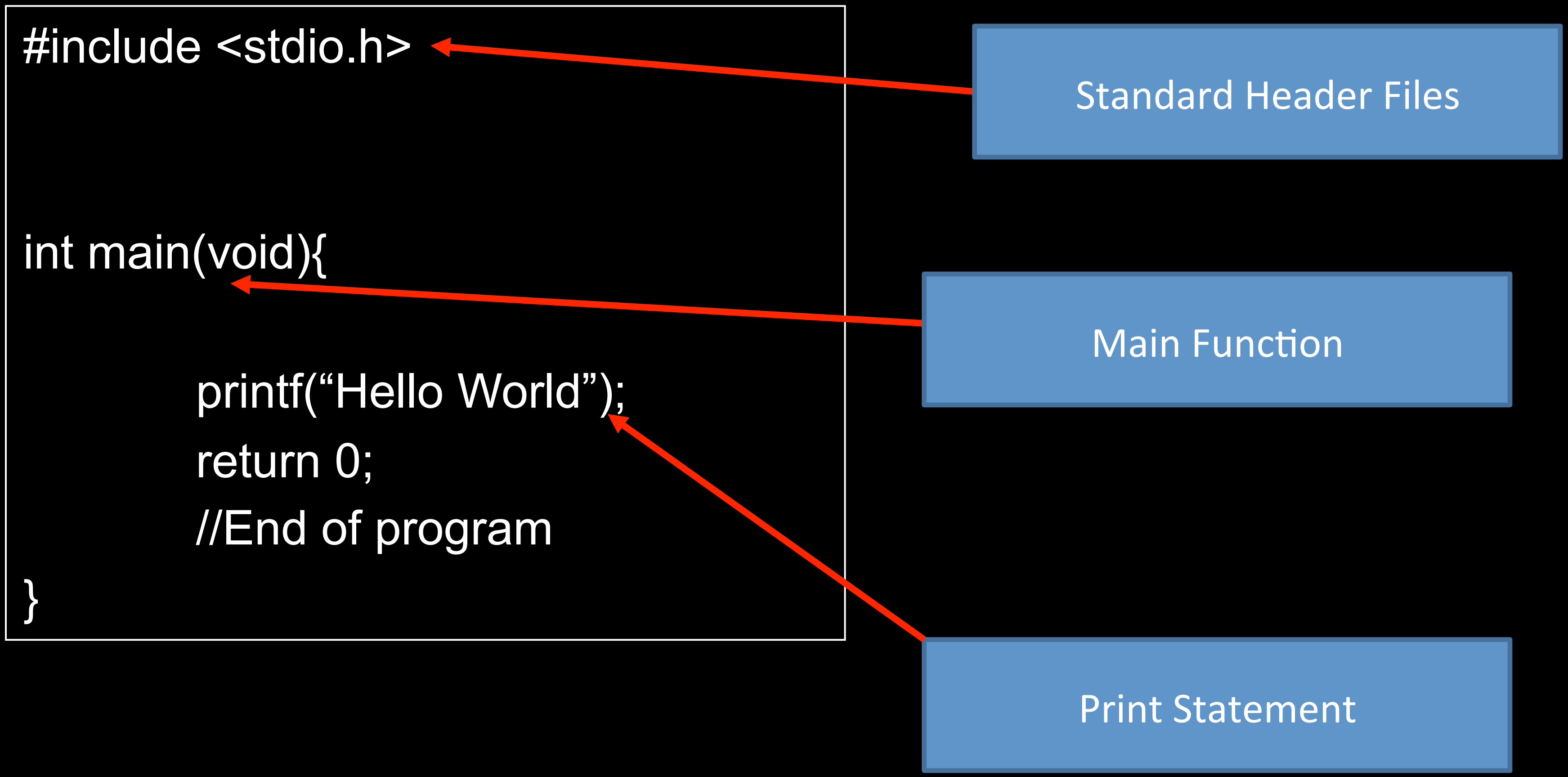
int main() { body }

Return type for the function. Body section must return the type specified here

Name of function and parameters to pass to the function

Actual code to be executed by the main function

Basic Structure of C Program



```
#include <stdio.h>


int main(void){

          printf(“Hello World”);
          return 0;
          //End of program
}
```

Standard Header Files

Main Function

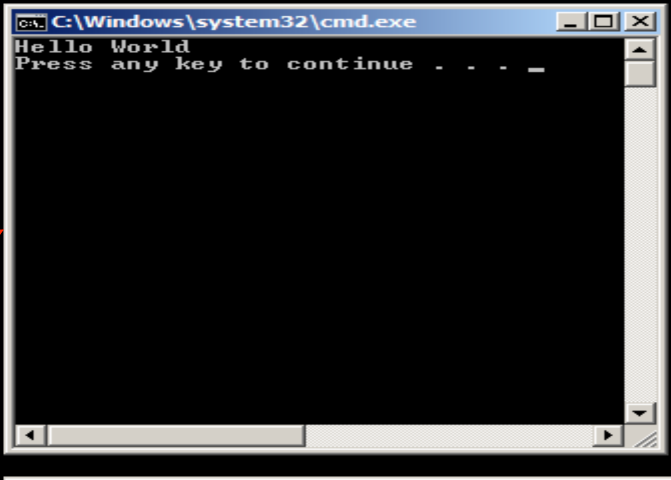
Print Statement

Print Statement

- printf – Writes C strings to standard output
  - printf is a function that is included under the stdio.h header file
- \n – indicates a newline character
  - Not including this on a print statement will print all statements on the same line

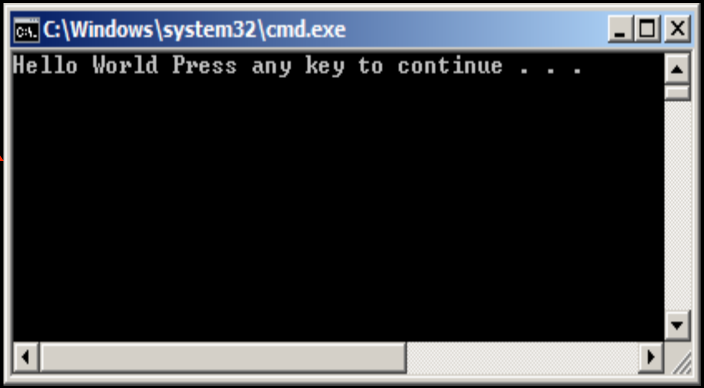
Print Statement Example

```
#include <stdio.h>
int main(){
        printf(“Hello World \n”);
        return 0;
}
```



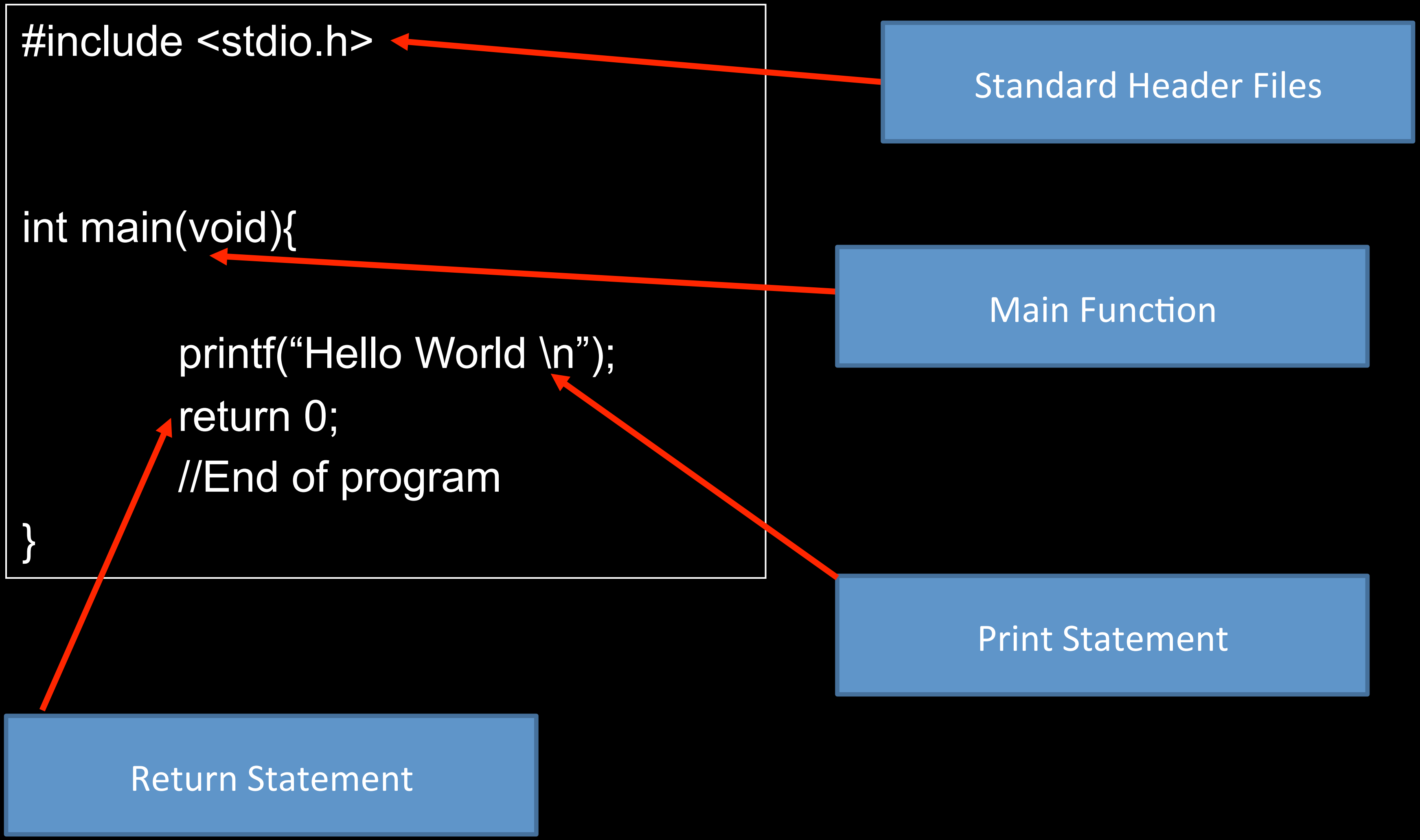
```
Hello World
Press any key to continue . . . _
```

```
#include <stdio.h>
int main(){
        printf(“Hello World ”);
        return 0;
}
```



```
Hello World Press any key to continue . . .
```

Basic Structure of C Program



```
#include <stdio.h>


int main(void){

          printf(“Hello World \n”);
          return 0;
          //End of program
}
```

Standard Header Files

Main Function

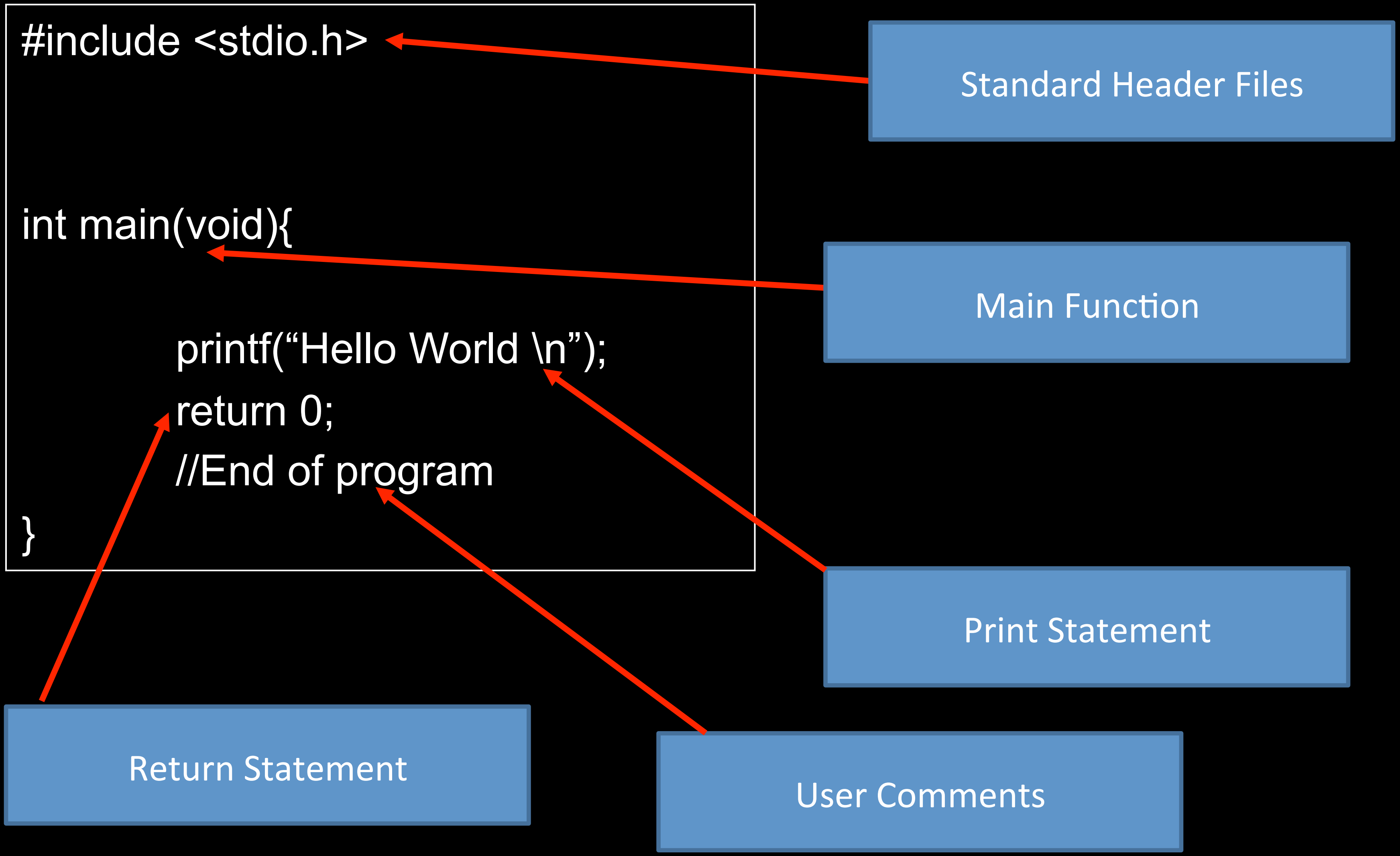
Print Statement

Return Statement

Return Statement

- Returns value from a function after it has been called
- Value return must be the same as the function type
- A function can return any type specified in C
  - If a function is of type void, it does not return a value

Basic Structure of C Program



```
#include <stdio.h>


int main(void){

          printf(“Hello World \n”);
          return 0;
          //End of program
}
```

Standard Header Files

Main Function

Print Statement

Return Statement

User Comments

User Comments

- Comments are not executed as part of the program
- Provides clarity as to what is occurring in a program
- Comments can be noted by // or /* */

```
//A C program would not execute
//lines such as these.
```

```
/*
Anything between these two
symbols would not be executed.
*/
```

Semicolons in C

- Notify C compiler of the end of a statement
- They are used after statements such as:
  - printf("Hello, World! \n");
  - return 0;

Variables

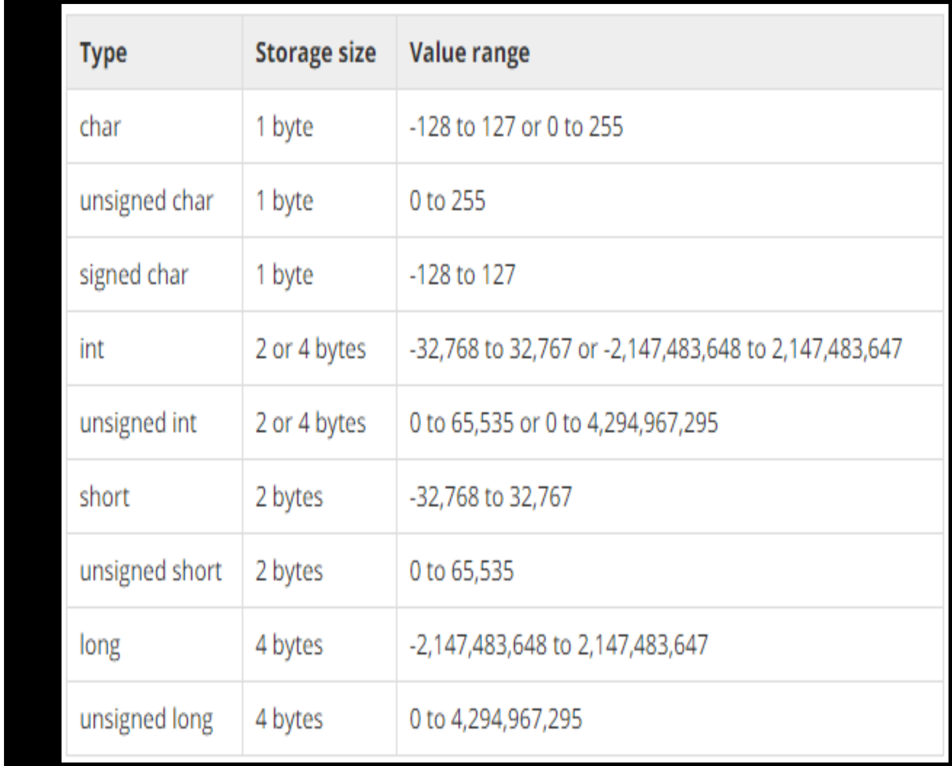
- Name given to a storage area that computer programs can manipulate
- Different variable types will be able to represent different types of values
- Variable names can be composed of letters, digits, and the underscore character
  - It must begin with an underscore or letter
    - Variable1 – Accepted
    - _Var2 – Accepted
    - 13Var3 – Not Accepted

Variables cont…

- Variables are also case sensitive
  - Using upper and lower case letters creates different variables
- Example
  - Number
  - number
  - NumbeR
- The strings above would create three distinct variables

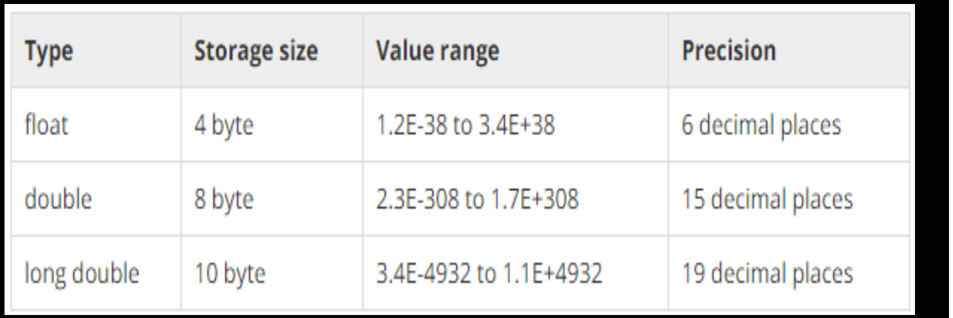
Variables cont..

Integer Types



Type	Storage size	Value range
char	1 byte	-128 to 127 or 0 to 255
unsigned char	1 byte	0 to 255
signed char	1 byte	-128 to 127
int	2 or 4 bytes	-32,768 to 32,767 or -2,147,483,648 to 2,147,483,647
unsigned int	2 or 4 bytes	0 to 65,535 or 0 to 4,294,967,295
short	2 bytes	-32,768 to 32,767
unsigned short	2 bytes	0 to 65,535
long	4 bytes	-2,147,483,648 to 2,147,483,647
unsigned long	4 bytes	0 to 4,294,967,295

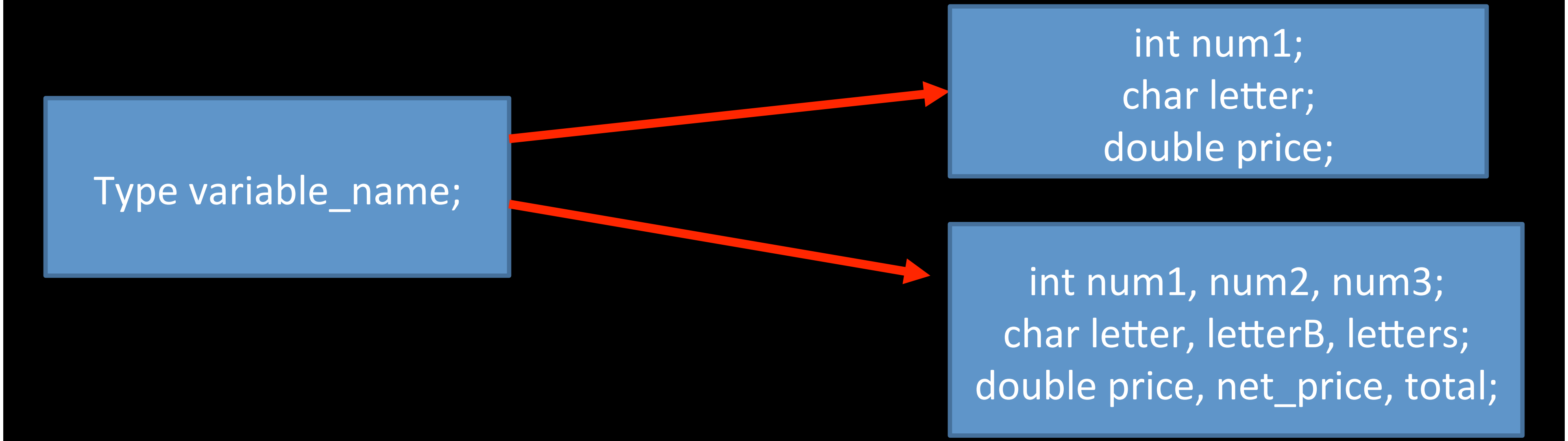
Floating-Point Types



Type	Storage size	Value range	Precision
float	4 byte	1.2E-38 to 3.4E+38	6 decimal places
double	8 byte	2.3E-308 to 1.7E+308	15 decimal places
long double	10 byte	3.4E-4932 to 1.1E+4932	19 decimal places

Variable Declaration

- Variables can only be declared using valid C data types
- The general structure is defined below



Type variable_name;

int num1;
char letter;
double price;

int num1, num2, num3;
char letter, letterB, letters;
double price, net_price, total;

Variable Initialization

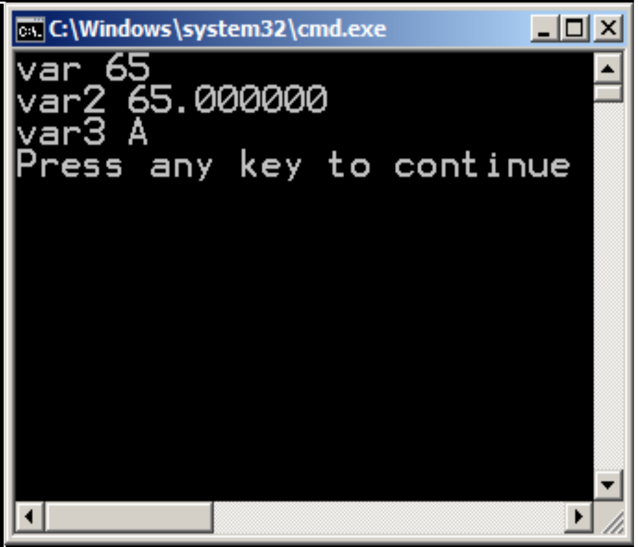
- Variables can also be given a value during declaration

```
int num1=10;
char letter=‘a’;
double price=3.14;
```

```
int num1=3, num2=17, num3=47;
char letter=‘V’, letterB=‘b’;
double price=3.33, net_price=19.54, total=37.80;
```

Variables Example

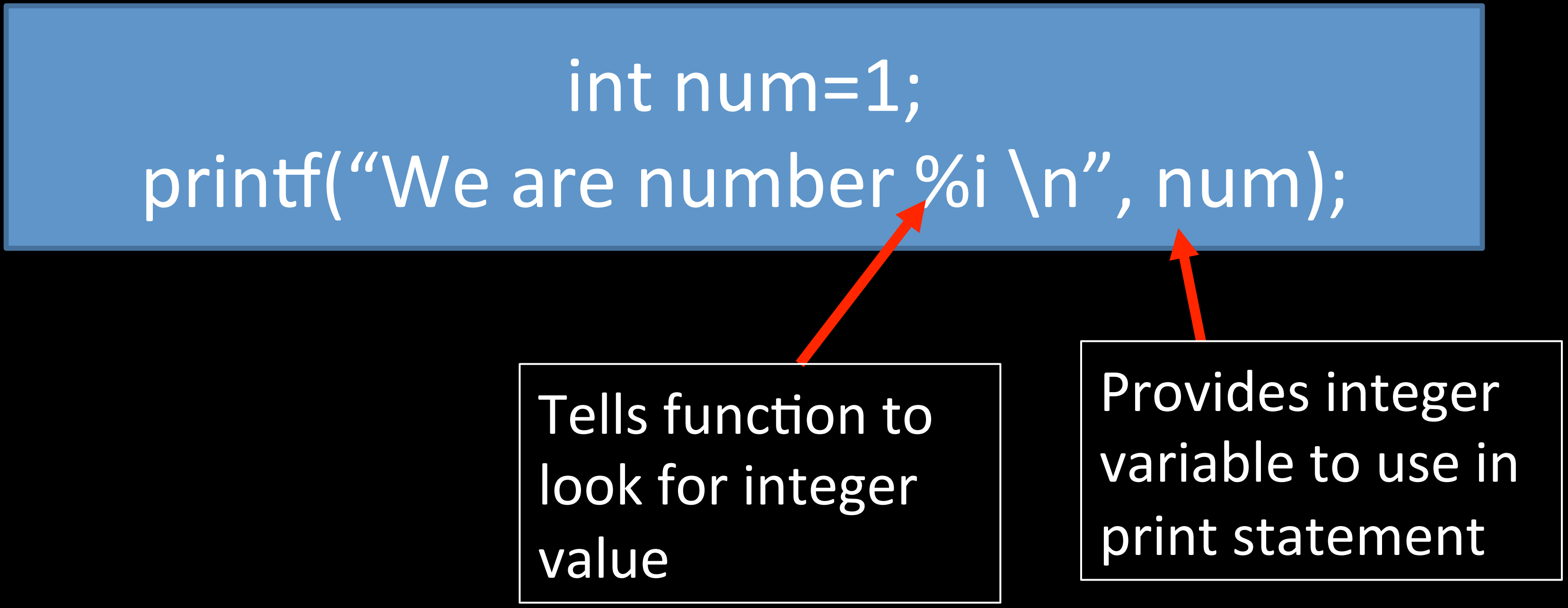
```
#include <stdio.h>

int main(){
		int var = 65;
		float var2 = 65;
		char var3 = ‘A’;

		printf(“var %i \n”,var);
		printf(“var2 %f \n”,var2);
		printf(“var3 %c \n”,var3);

		return 0;
}
```



```
var 65
var2 65.000000
var3 A
Press any key to continue
```

More printf Info

- C uses formatted output
  - The % sign with a character following it designates a certain format for a variable



```
int num=1;
printf(“We are number %i \n”, num);
```

Tells function to look for integer value

Provides integer variable to use in print statement

Local vs. Global Variables

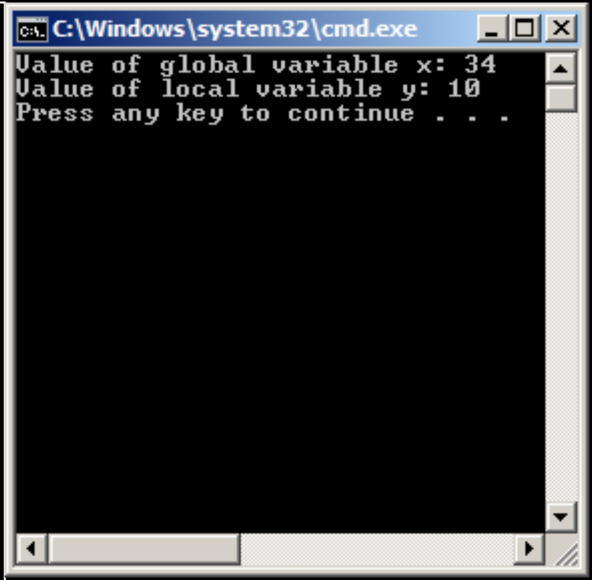
- Local variable- declared inside a function

- Global variable declared outside of all functions

- A local variable can only be used in the function where it is declared. A global variable can be used in all functions.

Global/Local Variables Example

```
#include <stdio.h>


int x = 17;
int main(){
        int y =20/2;
        x = x*2;
        printf("Value of global variable x: %d\n", x);
        printf("Value of local variable y: %d\n",y);
        return 0;
}
```



```
Value of global variable x: 34
Value of local variable y: 10
Press any key to continue . . .
```

If Statements

- Statement in C which tells a program what to execute based on a given condition
- Programs will often check if a variable is greater, smaller, or equal to another value

Relational Operators

>	greater than	7 > 4 is TRUE
<	less than	3 < 9 is TRUE
>=	greater than or equal	5 >= 5 is TRUE
<=	less than or equal	2 <=4 is TRUE
==	equal to	6 == 6 is TRUE
!=	not equal to	9 !=4 is TRUE

If Statement Structure

```
if (statement is TRUE){
          Execute line of code
}
```

Example of how statements work.

```
int x =15;


if (x > 10){
          printf(“Greater than ten”);
}
```

The variable *x* is equal to 15. *x* is greater than 10, so the print statement will be executed.

```
int x = 9

if (x != 9){
          printf(“Not equal to 9”);
}
```

The variable *x* is equal to 9. Since x is equal to 9 the print statement will not execute. It only executes when x is not equal to 9.

If/Else Statements

- Else statements add more control to how a program can be executed
- By using else statements additional conditions can be checked

```
if (statement is TRUE){
        Execute these lines of code if condition is TRUE
}

else{
        Execute these lines of code if condition is
        FALSE
}
```

If/Else Examples

```
void main(){
        int x = 12;

        if( x < 19){
                printf(“x is less than 19”);
        }
        else{
                printf(“x is not less than 19”);
        }
}
```

x is equal to 12. The program checks the first if statement. x < 19 is TRUE so it executes the first print statement. The else statement is not considered since the if statement was TRUE.

If/Else Examples cont..

```
void main(){
        int y = 30;

        if( x < 19){
                printf(“x is less than 19”);
        }
        else{
                printf(“x is not less than 19”);
        }
}
```

x is equal to 30. The program checks the first if statement. x < 19 is FALSE. It does not enter into the if statement. It does enter the else statement and prints “x is not less than 19”.

Loops

- Used to perform repeated operations until a condition is reached
- Like if statements, loops use relational operators and condition statements to determine how long to execute
- Here while loops and for loops will be studied

While Loop

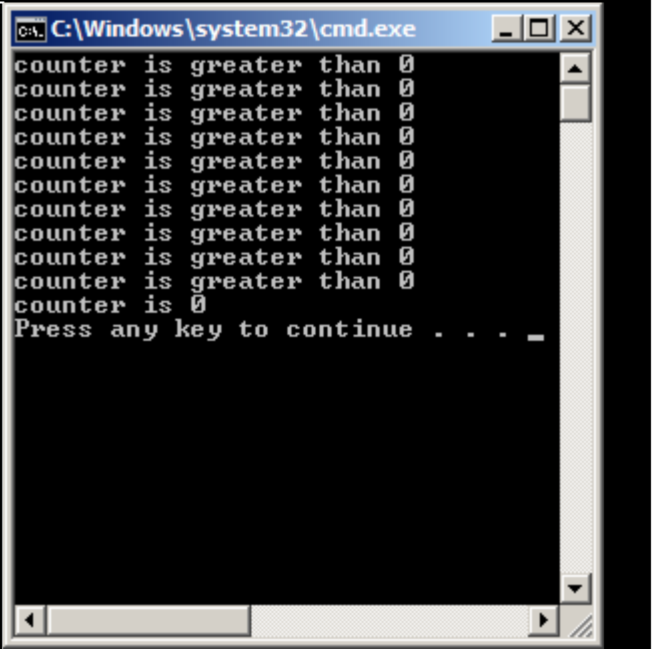
- Two components
  - Test condition
  - Counter modification

```
while (condition is TRUE){
            -Execute lines of code
            -Modify variable that affects test condition
}
```

```
x=0
while (x < 10){
            printf(“Hello”);
            x++; //Increments x by 1
}
```

While Loop Example

```
#include <stdio.h>


int main(){
        int counter=10;
        while (counter > 0){
                printf("counter is greater than 0\n");
                counter--;
        }
        printf("counter is %d\n",counter);
        return 0;
}
```

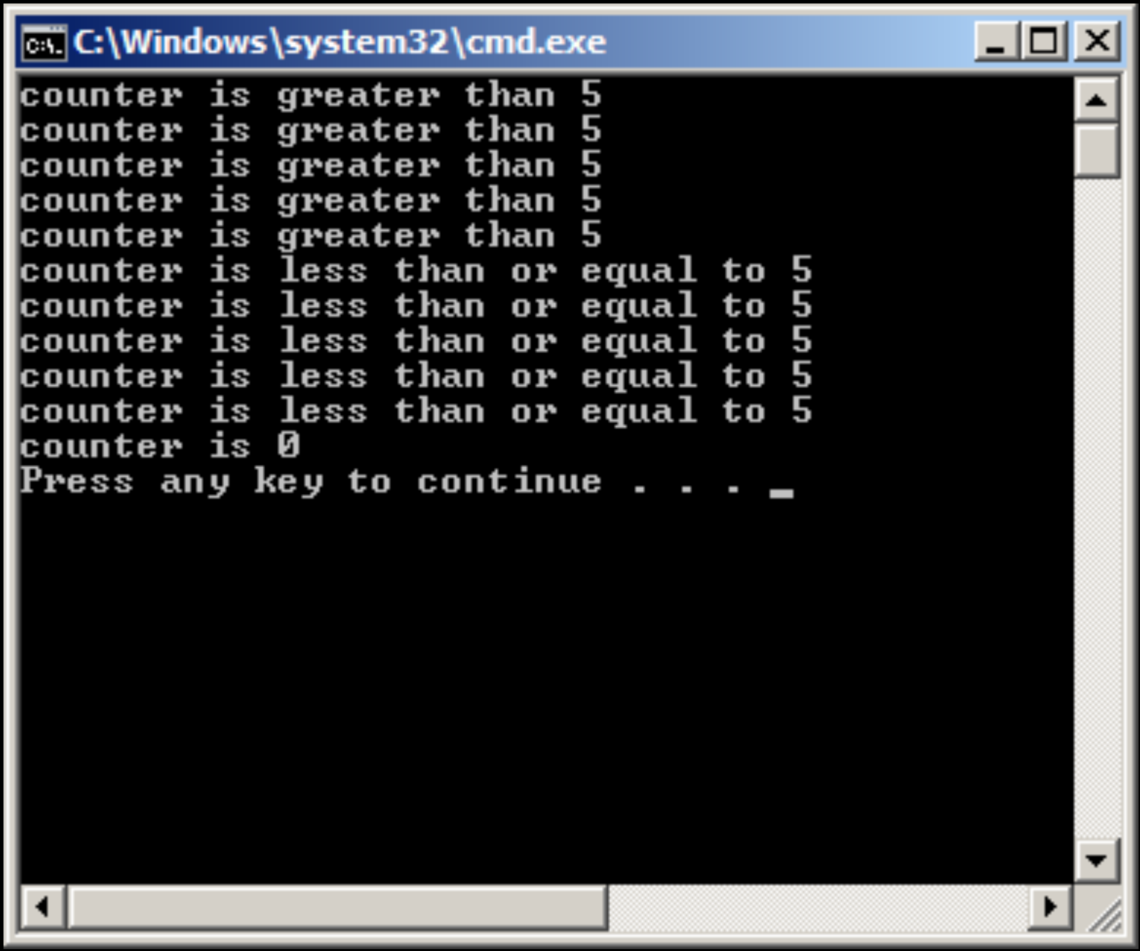


```
counter is greater than 0
counter is greater than 0
counter is greater than 0
counter is greater than 0
counter is greater than 0
counter is greater than 0
counter is greater than 0
counter is greater than 0
counter is greater than 0
counter is greater than 0
counter is 0
Press any key to continue . . . _
```

While Loops & If Statements

```
#include <stdio.h>
int main(){
        int counter=10;
        while (counter > 0){
                if (counter > 5){
                    printf("counter is greater than 5\n");
                }
                else{
                    printf("counter is less than or equal to 5\n");
                }
                counter--;
        }
        printf("counter is %d\n",counter);
        return 0;
}
```

While Loops & If Statements



```
counter is greater than 5
counter is greater than 5
counter is greater than 5
counter is greater than 5
counter is greater than 5
counter is less than or equal to 5
counter is less than or equal to 5
counter is less than or equal to 5
counter is less than or equal to 5
counter is less than or equal to 5
counter is 0
Press any key to continue . . .
```

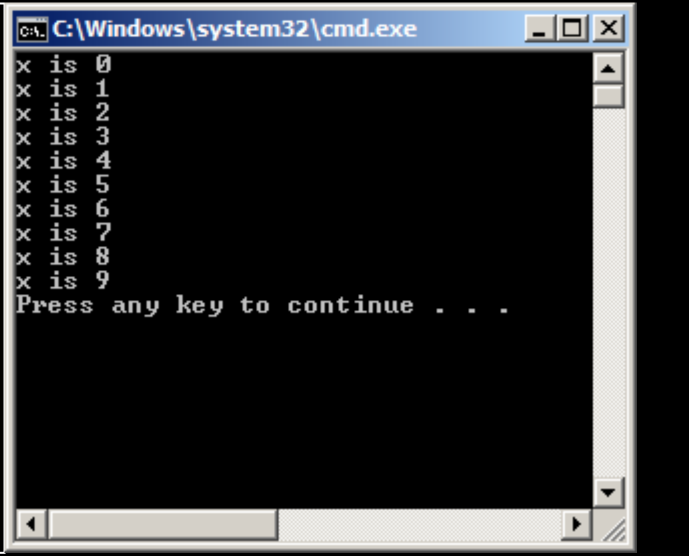
For Loops

- Three components
  - – Variable initialization
  - – Test condition
  - – Variable modification

```
for (variable initialization; test condition; variable modification){

        Code to execute when the test condition is true
}
```

For Loop Example

```
#include <stdio.h>
void main(){
        int x;
        for(x=0;x<10;x++){
                printf("x is %i\n",x);
        }
}
```



```
x is 0
x is 1
x is 2
x is 3
x is 4
x is 5
x is 6
x is 7
x is 8
x is 9
Press any key to continue . . .
```

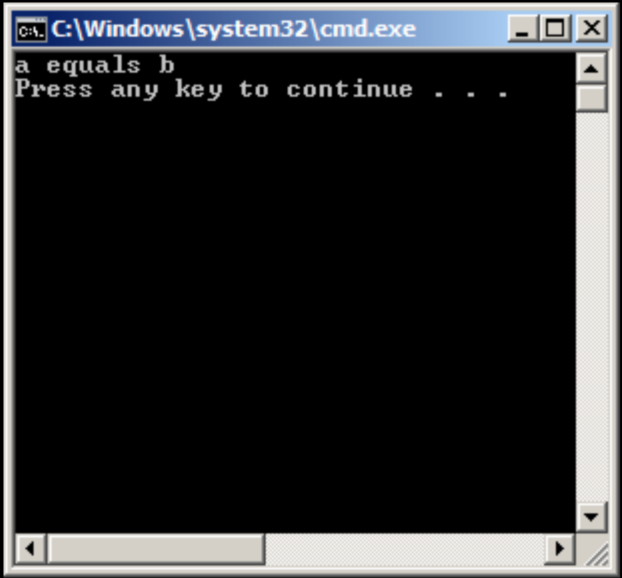
Switch Statements

- Helps control complex conditional and branching operations
- Takes a variable and test it for equality against a set of values

```
switch ( <variable> ) {
case this-value:
     Code to execute if <variable> == this-value
     break;
default:
      Code to execute if <variable> does not equal the value following any of the
cases
      break;
}
```

Switch Statement Example

```
void main(){
  int a = 10;
  const int b = 10;
  const int c = 20;

  switch ( a ) {
    case 10:
          printf(“a equals b”);
          break;
    case 20:
          printf(“a equals c”);
          break;
    default:
          printf(“Execute default case”);
          break;
    }
}
```



```
a equals b
Press any key to continue . . .
```

Array Data Structures

- C data structure that stores a fixed size sequential collection of elements
- Can be thought of a collection of variables of same type
- Parts of the array can be accessed via an index

Declaring Array’s

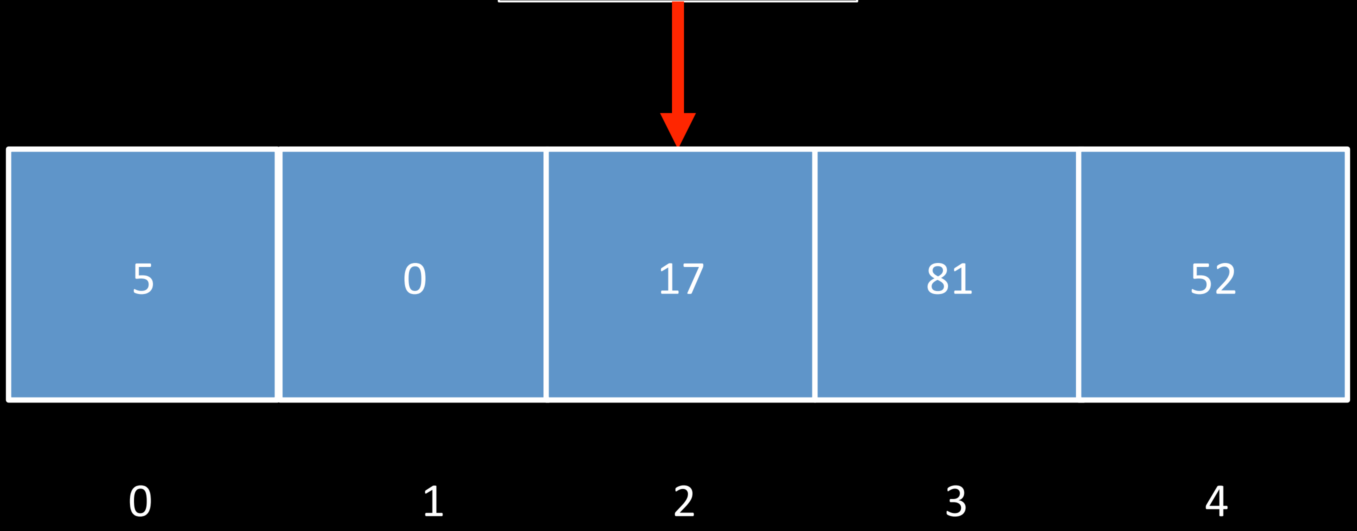
```
type arrayName [arraySize];
```

Actual Example

```
int numbers[10];
```

Array Example

```
int number[5];
number[0]=5;
number[1]=0
number[2]=17;
number[3]=81;
number[4]=52;
```

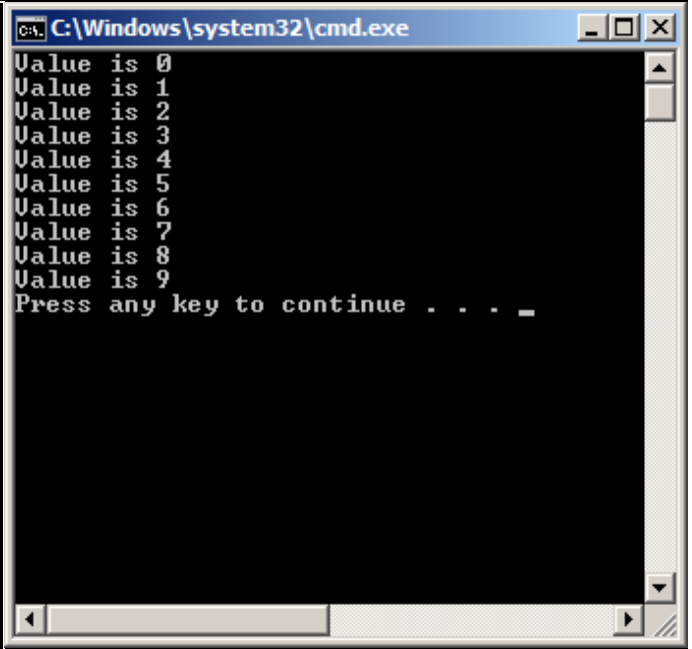


5	0	17	81	52
0	1	2	3	4

Index Positions
Every array starts at index 0

Array Examples

```
#include <stdio.h>
void main(){
        int numbers[10];
        int x;
        for(x=0;x<10;x++){
                numbers[x]=x;
                printf(“Value is %i\n”,numbers[x]);
        }
}
```



```
Value is 0
Value is 1
Value is 2
Value is 3
Value is 4
Value is 5
Value is 6
Value is 7
Value is 8
Value is 9
Press any key to continue . . . _
```

Additional Material

- Links for addition material
  - http://www.tutorialspoint.com/cprogramming/cprogramming_tutorial.pdf
  - http://www.learn-c.org/

Summary

- Presented basic C programming constructs
- Discussed basic variables, constructs, and control flow statements

Questions