Introduction to C

Wang CHEN

CSC2100 Data Structures Tutorial 1

Information

- Course Information:
- Web Page:
  - http://www.cse.cuhk.edu.hk/irwin.king/teaching/csci2100/2016
- Tutorial Page:
  - http://www.cse.cuhk.edu.hk/irwin.king/teaching/csci2100/2016/tutorial
- Anti-plagiarism Policy:
  - http://www.cuhk.edu.hk/policy/academichonesty/

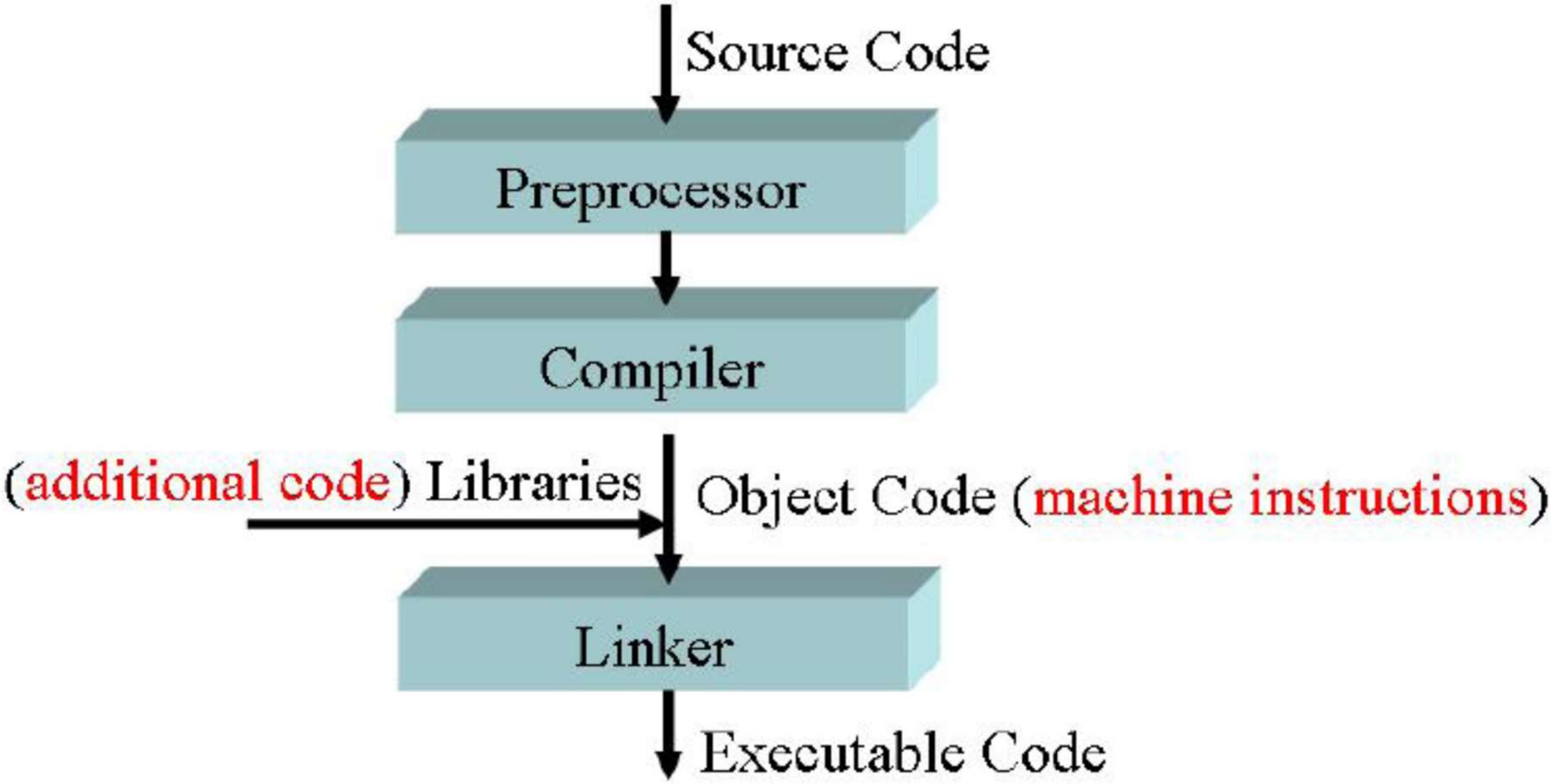
Information

- Assignment
  - There will be both written and programming parts in assignments.
    - Written part: submit to the assignment box in 10/F SHB.
    - Programming part: via Online Judge systems. (Will be introduced next week)
  - For non-CSE student, you will receive your login Id for CSC2100 online judge via your CUHK Link email account (after add/drop week).
    - Keep it safe and do not disclose it.

Introduction to C

- Basics
- If Statement
- Loops
- Functions
- Switch case
- Pointers
- Structures
- File I/O
- Debugging

The C Compilation Model


Source Code
Preprocessor
Compiler
(additional code) Libraries
Object Code (machine instructions)
Linker
Executable Code

Introduction to C: Basics

```
/*a simple program
that has variables*/
#include <stdio.h>
int main()
{
    int x; //(32 bits)
    char y; //(8 bits)
    float z; //(32 bits)
    double t; //(64 bits)
    printf(“hello world…\n”);
    test = 1; //wrong, The variable declaration must appear first
    return 0;
}
```

Introduction to C: Basics

```
//reading input from console
#include <stdio.h>
int main()
{
    int num1;
    int num2;
    printf( "Please enter two numbers: " );
    scanf( "%d %d", &num1,&num2 );
    printf( "You entered %d %d", num1, num2 );
    return 0;
}
```

Introduction to C: if statement

```
#include <stdio.h>
int main()
{
    int age;                    /* Need a variable... */
    printf( "Please enter your age" );  /* Asks for age */
    scanf( "%d", &age );          /* The input is put in age */
    if ( age < 100 )
    {           /* If the age is less than 100 */
        printf ("You are pretty young!\n" ); /* Just to show you it works... */
    }
    else if ( age == 100 )
    {       /* I use else just to show an example */
        printf( "You are old\n" );
    }
    else
    {
        printf( "You are really old\n" );    /* Executed if no other statement is*/
    }
    return 0;
}
```

Introduction to C: Loops(for)

```
#include <stdio.h>
int main()
{
    int x;
    /* The loop goes while x < 10, and x increases by one every loop*/
    for ( x = 0; x < 10; x++ )
    {
        /* Keep in mind that the loop condition checks
           the conditional statement before it loops again.
           consequently, when x equals 10 the loop breaks.
           x is updated before the condition is checked. */
           printf( "%d\n", x );
    }
    return 0;
}
```

Introduction to C: Loops(while)

```
#include <stdio.h>
int main()
{
  int x = 0;  /* Don't forget to declare variables */
  while ( x < 10 )
  {   /* While x is less than 10 */
      printf( "%d\n", x );
      x++;    /* Update x so the condition can be met eventually */
  }
  return 0;
}
```

Introduction to C: Loops(do while)

```
#include <stdio.h>
int main()
{
  int x;
  x = 0;
  do
  {
    /* "Hello, world!" is printed at least one time
      even though the condition is false*/
     printf( "%d\n", x );
      x++;
  } while ( x != 10 );
  return 0;
}
```

Introduction to C: Loops(break and continue)

```
#include <stdio.h>
int main()
{
  int x;
  for(x=0;x<10;x++)
  {
        if(x==5)
        {
                break;
        }
        printf("%d\n",x);
  }
  return 0;
}
```

```
0
1
2
3
4
```

```
#include <stdio.h>
int main()
{
  int x;
  for(x=0;x<10;x++)
  {
        if(x==5)
        {
                continue;
        }
        printf("%d\n",x);
  }
  return 0;
}
```

```
0
1
2
3
4
6
7
8
9
```

```c
#include <stdio.h>
//function declaration, need to define the function body in other places
void playgame();
void loadgame();
void playmultiplayer();
int main()
{
    int input;
    printf( "1. Play game\n" );
    printf( "2. Load game\n" );
    printf( "3. Play multiplayer\n" );
    printf( "4. Exit\n" );
    printf( "Selection: " );
    scanf( "%d", &input );
    switch ( input ) {
        case 1:             /* Note the colon, not a semicolon */
            playgame();
            break;          //don't forget the break in each case
        case 2:
            loadgame();
            break;
        case 3:
            playmultiplayer();
            break;
        case 4:
            printf( "Thanks for playing!\n" );
            break;
        default:
            printf( "Bad input, quitting!\n" );
            break;
    }
    return 0;
}
```

switch

case

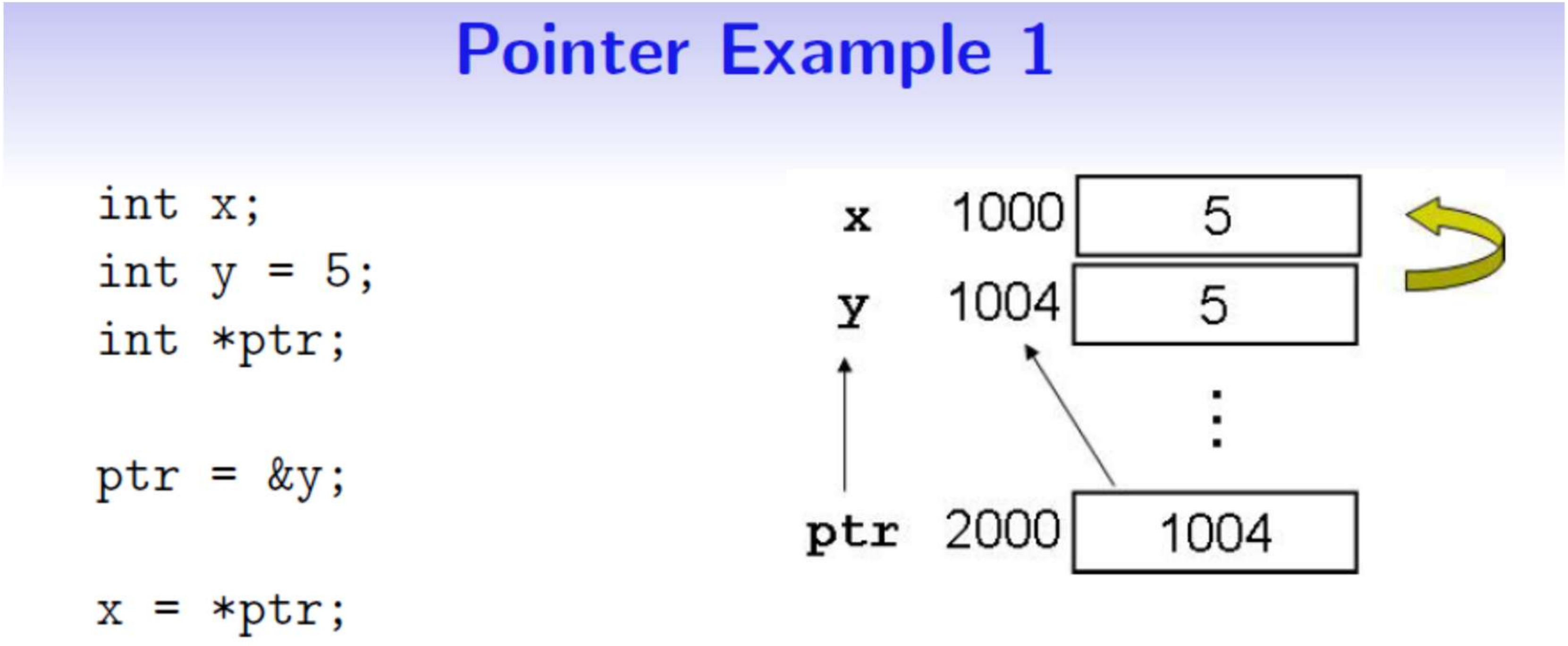
Introduction to C: function

```
#include <stdio.h>
//function declaration
int mult ( int x, int y );
int main()
{
  int x, y;
  printf( "Please input two numbers to be multiplied: " );
  scanf( "%d", &x );
  scanf( "%d", &y );
  printf( "The product of your two numbers is %d\n", mult( x, y ) );
  return 0;
}
//define the function body
//return value: int
//utility: return the multiplication of two integer values
//parameters: take two int parameters
int mult (int x, int y)
{
  return x * y;
}
```

Introduction to C: pointer variables

- Pointer variables are variables that store memory addresses.
- Pointer Declaration:
  - int x, y = 5;
  - int *ptr;
  - /*ptr is a POINTER to an integer variable*/
- Address operator &:
  - ptr = &y;
  - /*assign ptr to the MEMORY ADDRESS of y.*/
- Dereference operator *:
  - x = *ptr;
  - /*assign x to the int that is pointed to by ptr */

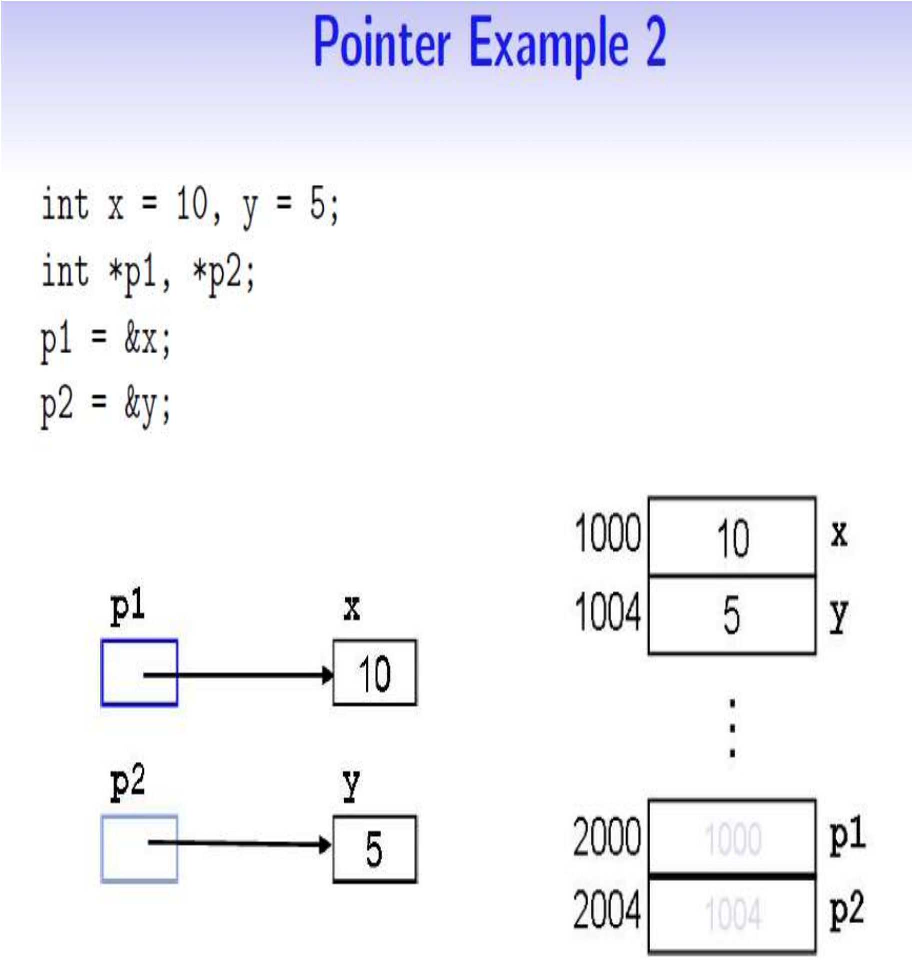
Introduction to C: pointer variables



Pointer Example 1

```
int x;
int y = 5;
int *ptr;

ptr = &y;

x = *ptr;
```

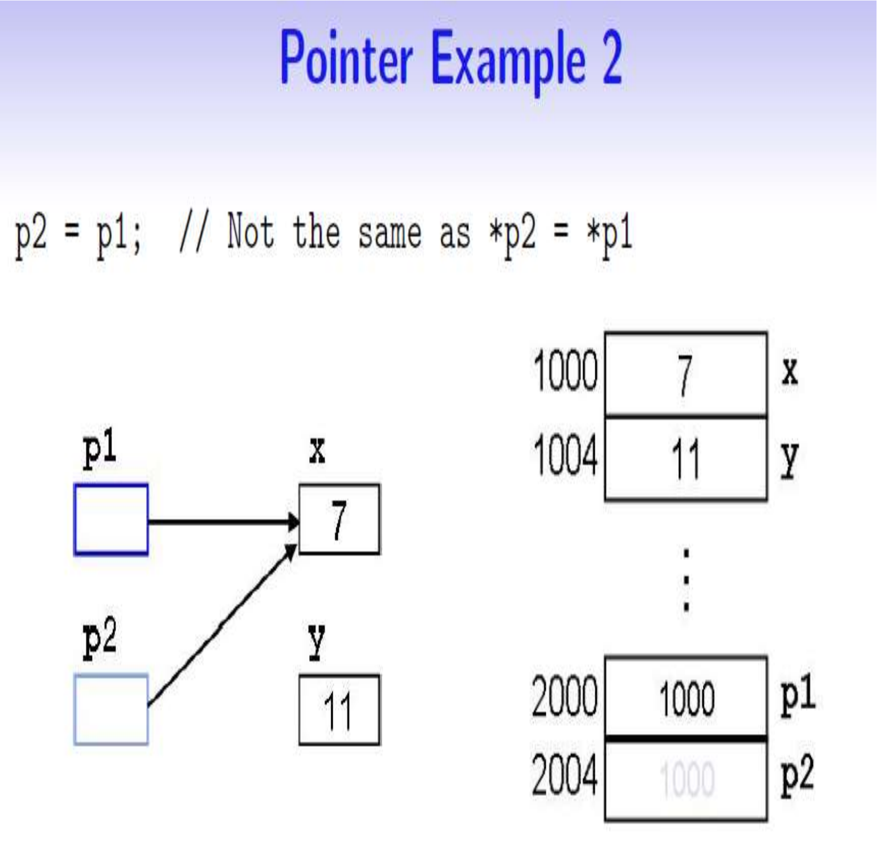
Introduction to C: pointer variables



Pointer Example 2

```
int x = 10, y = 5;
int *p1, *p2;
p1 = &x;
p2 = &y;
```

1000	10	x
1004	5	y
	⋮	
2000	1000	p1
2004	1004	p2



Pointer Example 2

```
p2 = p1;  // Not the same as *p2 = *p1
```

1000	7	x
1004	11	y
	⋮	
2000	1000	p1
2004	1000	p2

```
#include <stdio.h>
//swap two values
void swap(int* iPtrX,int* iPtrY);
void fakeswap(int x, int y);
int main()
{
    int x = 10;
    int y = 20;
    int *p1 = &x;
    int *p2 = &y;
    printf("before swap: x=%d y=%d\n",x,y);
    swap(p1,p2);
    printf("after swap: x=%d y=%d\n",x,y);

    printf("------------------------------\n");
    printf("before fakeswap: x=%d y=%d\n",x,y);
    fakeswap(x,y);
    printf("after fakeswap: x=%d y=%d",x,y);
  return 0;
}
```

```
void swap(int* iPtrX, int* iPtrY)
{
  int temp;
    temp = *iPtrX;
    *iPtrX = *iPtrY;
    *iPtrY = temp;
}
void fakeswap(int x,int y)
{
  int temp;
    temp = x;
    x = y;
    y = temp;
}
```

Introduction to C: Array

Array is a fixed size, sequenced collection of elements of the same data type, with index starts with zero

- Array declaration:

```
int a[4];
```

- Array initialization:

```
int a[4] = {3,4,5,6};
```

- Assignment to and from array element

```
a[0] = 3;
value = a[1];
```

Introduction to C: Array

Use pointer to access Array

```
    int *ptr = a;   // int a[4] in the last slides.
```

Let's set a[1] to 1

```
    ptr[1] = 1;
    *(ptr+1) = 1;   // we could use ptr+i  to get the address of (i-1)th
                                    element in one array
```

```
int i = 3;
ptr2 = a;
ptr = ptr + i; // Ok if i is smaller than the array size
i = ptr - ptr2;  // Ok,  i = 3
ptr = ptr + ptr2;   // Wrong! It's forbidden
```

Introduction to C: struct

```
#include <stdio.h>
//group things together
struct database {
  int id_number;
  int age;
  float salary;
};

int main()
{
  struct database employee;
  employee.age = 22;
  employee.id_number = 1;
  employee.salary = 12000.21;
  printf("Employeee No.%d is %d and his salary is %f\n", employee.id_number,
      employee.age, employee.salary);    // Output:  Employee No.1 is 22 and his salary
      is 12000.21
 return 0;
}
```

File I/O

```
#include <stdio.h>

int main()
{
 FILE *ifp, *ofp;
 char *mode = "r";
 char outputFilename[] = "out.list";
 char username[9];
 int score;
 ifp = fopen("in.list", mode);
 if (ifp == NULL) {
  fprintf(stderr, "Can't open input file in.list!\n");
  exit(1);
 }
 ofp = fopen(outputFilename, "w");
 if (ofp == NULL) {
  fprintf(stderr, "Can't open output file %s!\n", outputFilename);
  exit(1);
 }
 while (fscanf(ifp, "%s %d", username, &score) == 2) {
  fprintf(ofp, "%s %d\n", username, score+10);
 }
 fclose(ifp);
 fclose(ofp);
 return 0;
}
```

mode:
r - open for reading
w - open for writing (file need not exist)
a - open for appending (file need not exist)
r+ - open for reading and writing, start at beginning
w+ - open for reading and writing (overwrite file)
a+ - open for reading and writing (append if file exists)

Debugging

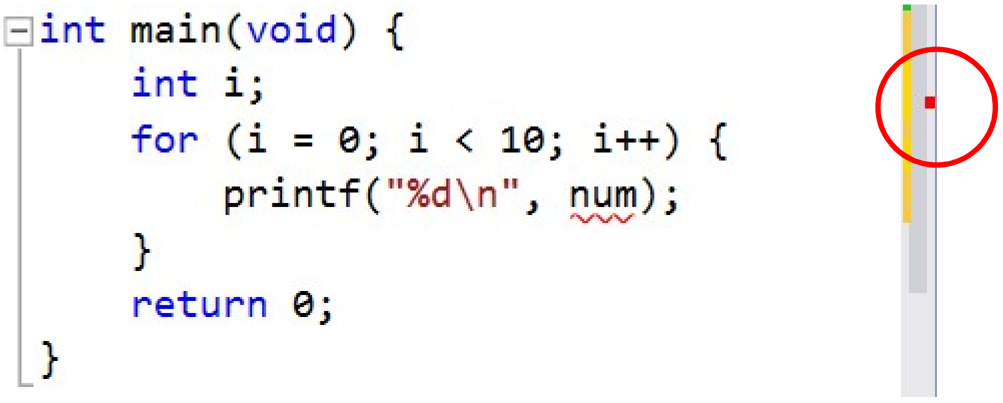
From engg1110 lecture slices.

Outline

- Types of Errors
  - Compilation/Syntax errors
  - Run-Time errors
  - Logical errors
- Debugging techniques to locate logical errors
- Using Debugger in Visual Studio

Types of Errors – syntax errors

- Type 1. **Compilation/Syntax Errors**.
  - The errors that are detected during the compilation of the program.
  - Visual Studio can highlight these errors for you.



```
int main(void) {
    int i;
    for (i = 0; i < 10; i++) {
        printf("%d\n", num);
    }
    return 0;
}
```

Move your mouse cursor over the squiggly red line to get more info about the error.

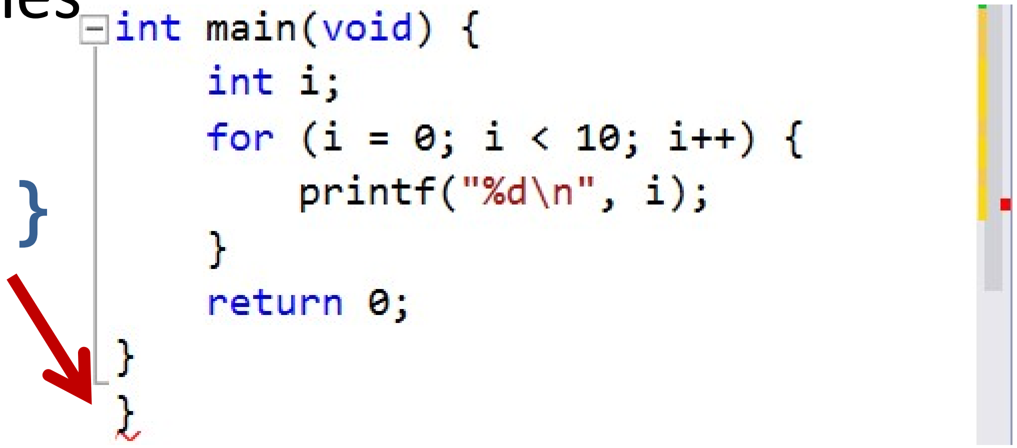
The red bar indicates that there is an error in this file.

Note: Sometimes the errors may appear before the indicated line.

Types of Errors – syntax errors

- Type 1. **Compilation/Syntax Errors**.

- Common syntax errors:
  - Duplicate variable names
  - Missing semi-colons **;**
  - Mismatched braces **{ }**



```
int main(void) {
    int i;
    for (i = 0; i < 10; i++) {
        printf("%d\n", i);
    }
    return 0;
}
}
```

This is called a dangling brace.

Types of Errors – run-time errors

- Type 2. **Run-Time Errors**.
  - The errors occur while the program is running and cause the program to crash.

```
int a, b;
a = 3;
b = 0;
printf("%d\n", a / b);
```

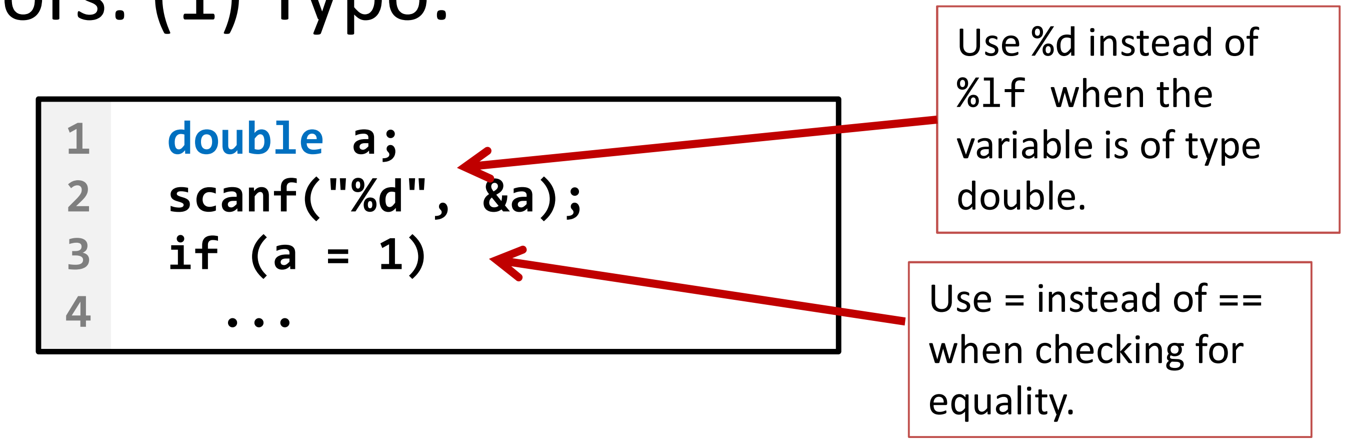
- Common run-time errors:
  - Division by zero
  - Array index out of bound
    - The consequence of the array index out of bound error is unpredictable;
    - The program may crash (run-time errors), or
    - Some variables may get modified unknowingly (the program does not crash).

```
int array[10] = { 0 };
array[10] = 50;
printf("%d\n", array[1000]);
```

Types of Errors – logical errors

- Type 3. **Logical Errors**: the result is unexpected!
  - Not syntax errors or run-time errors.
    - i.e., the program can be compiled and executed successfully.
  - But, the program logic is wrong.

- Source of errors: (1) Typo.



```c
double a;
scanf("%d", &a);
if (a = 1)
  ...
```

Use %d instead of `%lf` when the variable is of type double.

Use = instead of == when checking for equality.

Types of Errors – logical errors

- Source of errors: (2) **Incorrect program logic**.
  - This is the most frustrating moment!
  - Because we usually spend **most of the programming time** in discovering where the error is.

- Don’t give up yet!
  - We have systematic way to locate logical bugs.

Outline

- Types of Errors
  - Compilation/Syntax errors
  - Run-Time errors
  - Logical errors

- **Debugging techniques to locate logical errors**

- Using Debugger in Visual Studio

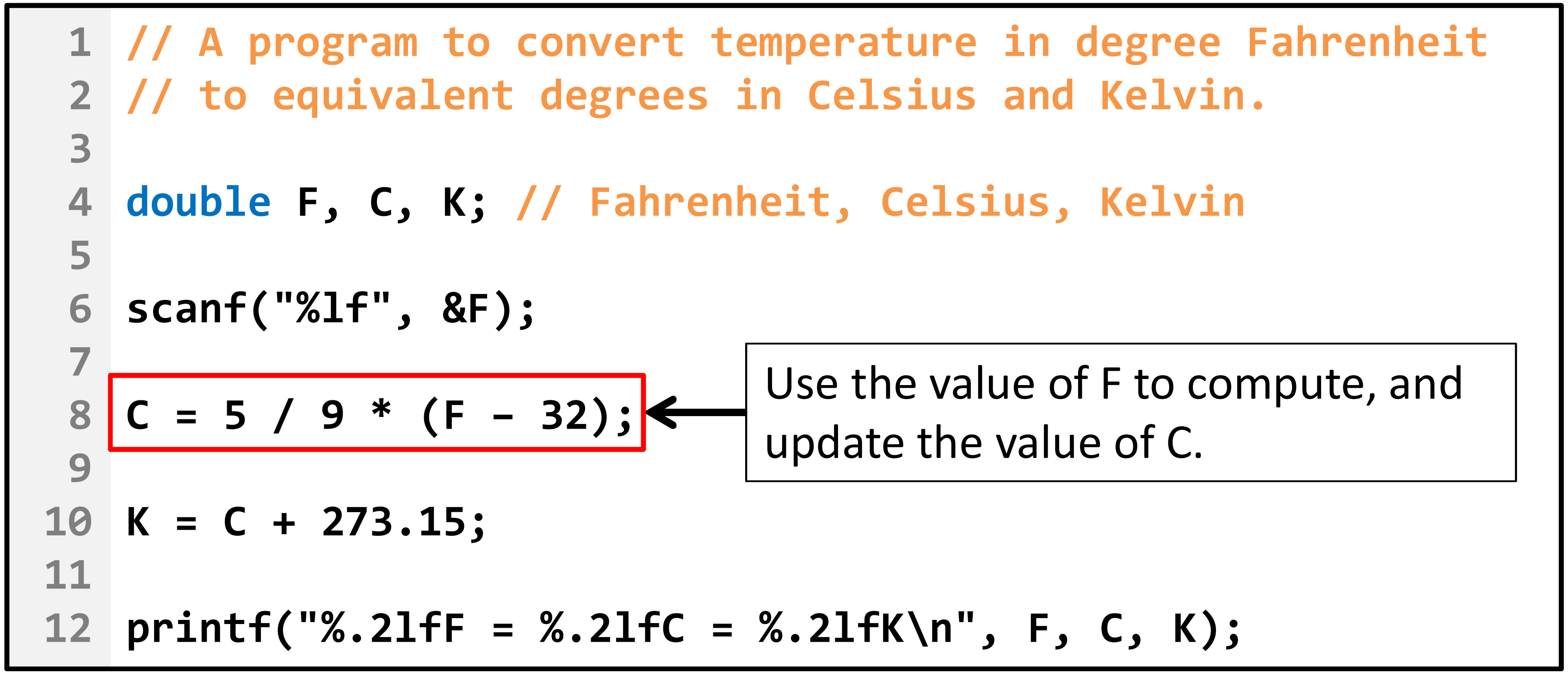
How to locate a logical error?

- The output of this program is incorrect.
- How should we approach to find the bug?

```c
// A program to convert temperature in degree Fahrenheit
// to equivalent degrees in Celsius and Kelvin.

double F, C, K; // Fahrenheit, Celsius, Kelvin

scanf("%lf", &F);

C = 5 / 9 * (F – 32);

K = C + 273.15;

printf("%.2lfF = %.2lfC = %.2lfK\n", F, C, K);
```

How to locate a logical error?

- Every statement computes in the following manners
  - Base its computation on the value of some variable(s)
  - Update the value of some variable(s)



```c
// A program to convert temperature in degree Fahrenheit
// to equivalent degrees in Celsius and Kelvin.

double F, C, K; // Fahrenheit, Celsius, Kelvin

scanf("%lf", &F);

C = 5 / 9 * (F - 32);

K = C + 273.15;

printf("%.2lfF = %.2lfC = %.2lfK\n", F, C, K);
```

Use the value of F to compute, and update the value of C.

How to locate a logical error?

- If a variable is assigned a wrongly computed value, subsequent computations will likely produce wrong results.

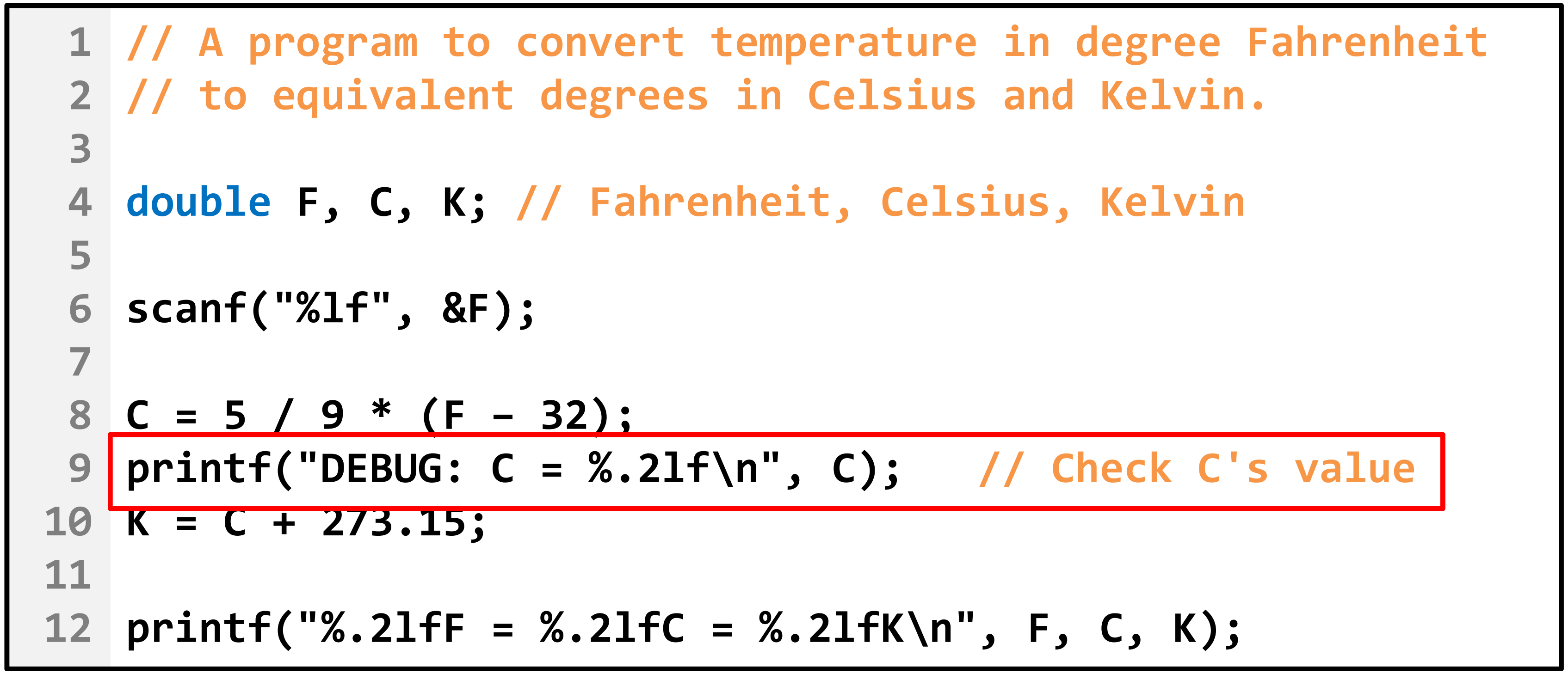
```
// A program to convert temperature in degree Fahrenheit
// to equivalent degrees in Celsius and Kelvin.

double F, C, K; // Fahrenheit, Celsius, Kelvin

scanf("%lf", &F);

C = 5 / 9 * (F - 32);

K = C + 273.15;

printf("%.2lfF = %.2lfC = %.2lfK\n", F, C, K);
```

If C is assigned a wrong value here, then subsequently the value of K and the output will be affected.

How do we find out if C's value is wrong?

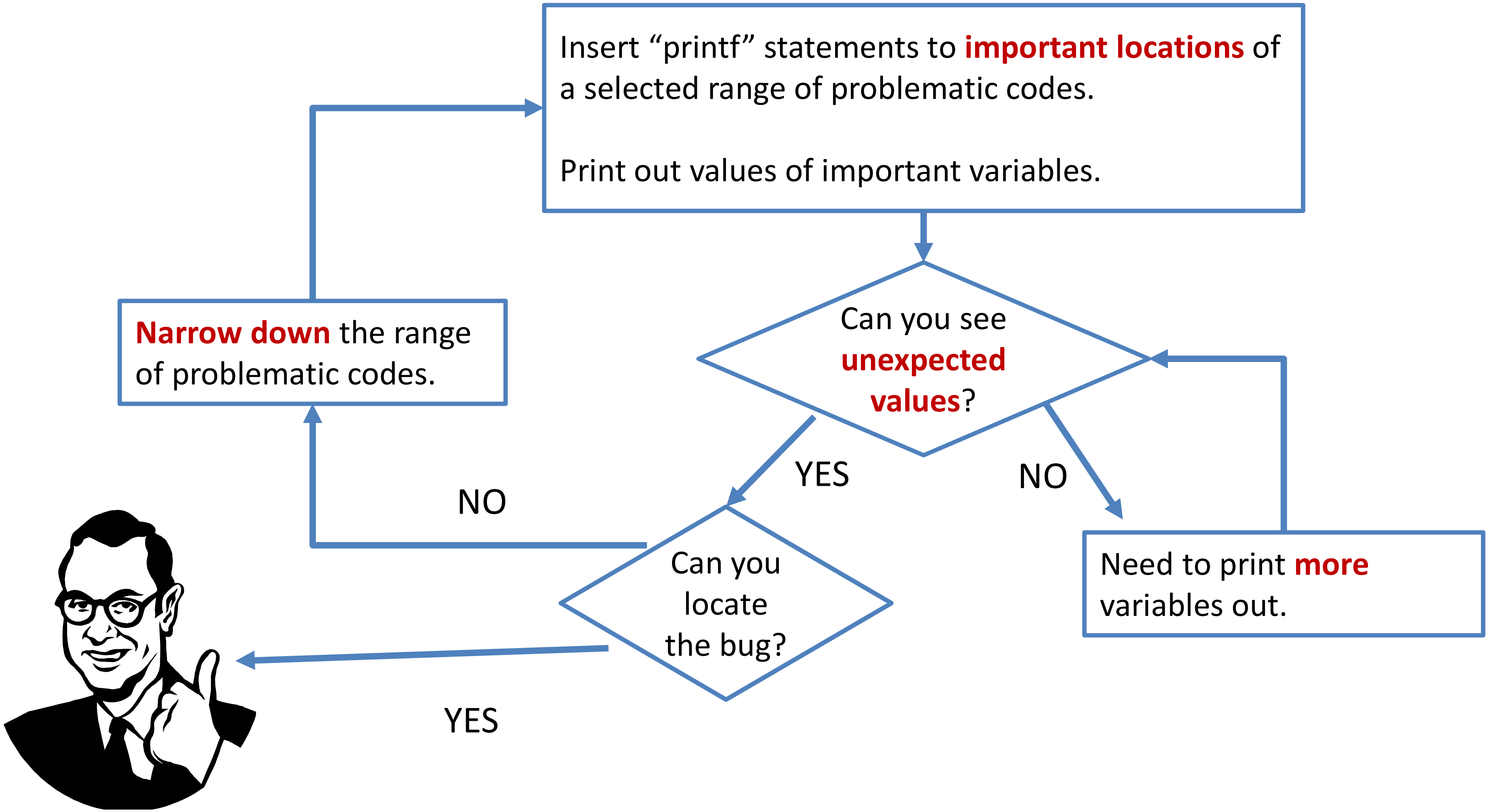
How to locate a logical error?

- Variables usually hold some clues to the bug.
  - One way to inspect variables is to output their values.



```c
// A program to convert temperature in degree Fahrenheit
// to equivalent degrees in Celsius and Kelvin.

double F, C, K; // Fahrenheit, Celsius, Kelvin

scanf("%lf", &F);

C = 5 / 9 * (F - 32);
printf("DEBUG: C = %.2lf\n", C);    // Check C's value
K = C + 273.15;

printf("%.2lfF = %.2lfC = %.2lfK\n", F, C, K);
```

Steps in locating logical errors



Insert “printf” statements to **important locations** of a selected range of problematic codes.

Print out values of important variables.

Can you see **unexpected values**?

YES

NO

Need to print **more** variables out.

Can you locate the bug?

NO

Narrow down the range of problematic codes.

YES

Outline

- Types of Errors
  - Compilation/Syntax errors
  - Run-Time errors
  - Logical errors
- Debugging techniques to locate logical errors
- **Using Debugger in Visual Studio (Self-Study)**

Start with/without Debugging

- In the menu on the top, choose “DEBUG”
  - Start Debugging (F5)
    - Run the program in debug mode.
      - The program can be paused in the middle.
    - After the execution of the program, the console window will be closed immediately.
  - Start Without Debugging (Ctrl+F5)
    - The program runs normally.
    - After the execution of the program, the console window will pause and show “Press any key to continue . . .”.

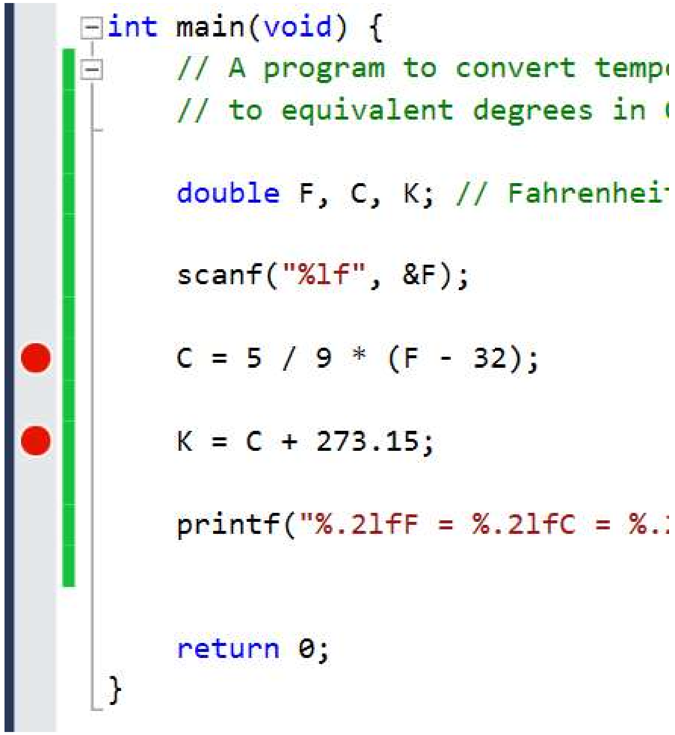
Setting the Breakpoints

- A **breakpoint** typically represents a location (usually a line of code) where an executing program is paused.

To toggle (set or remove) a breakpoint at a line of code, click at the spot in the left grey area next to the line in the editing window.

A small red circle indicates that a break point is set at that line.

Multiple breaks points are allowed.

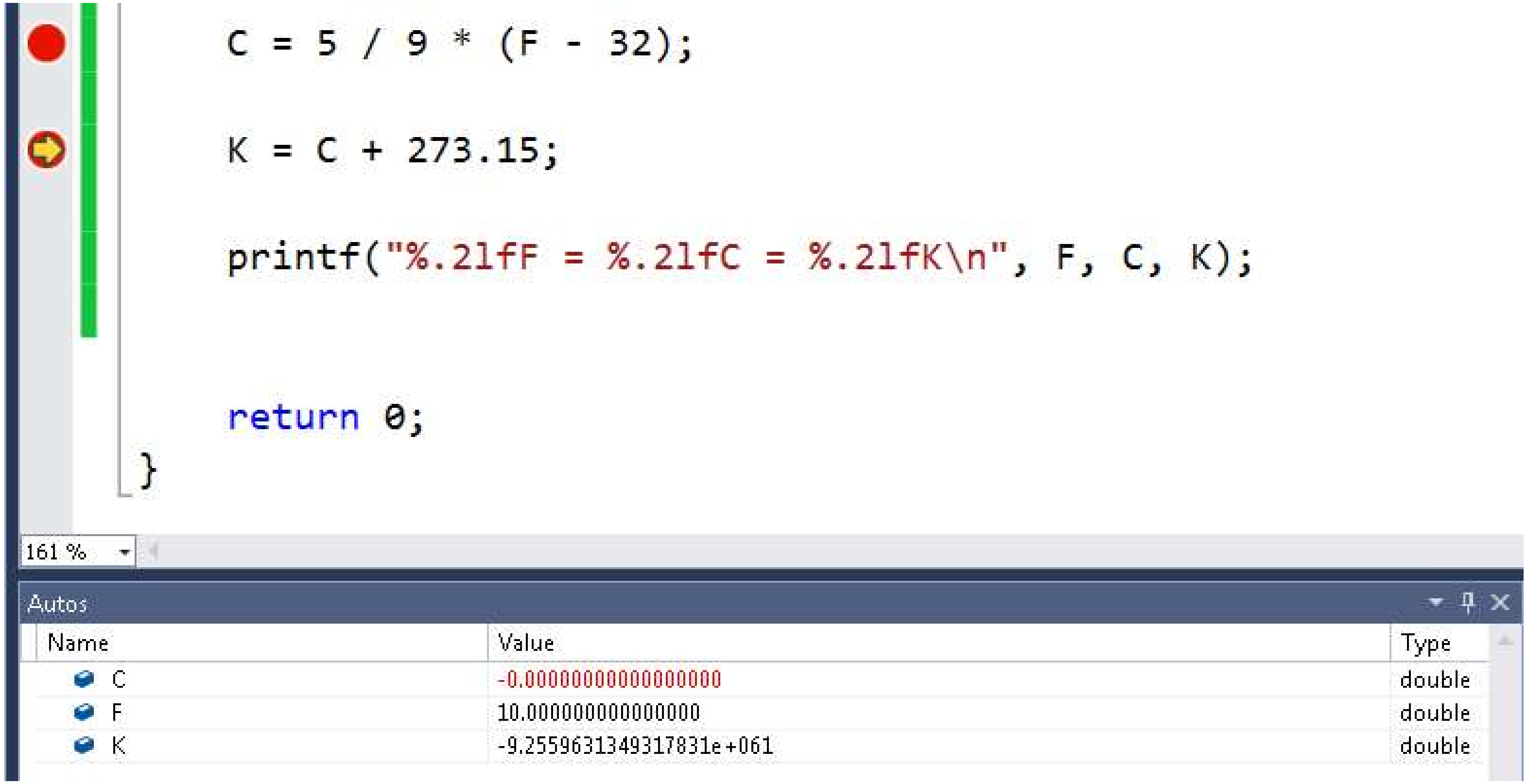


```
int main(void) {
        // A program to convert temp
        // to equivalent degrees in

        double F, C, K; // Fahrenhei

        scanf("%lf", &F);

●       C = 5 / 9 * (F - 32);

●       K = C + 273.15;

        printf("%.2lfF = %.2lfC = %.


        return 0;
}
```

Inspecting Values of Variables

- When a program is paused, we can view the values of the variables through:
  - "Autos" window
    - View variables that are related to the current statement
  - "Locals" window
    - View variables in the current scope
  - "Watch" window
    - View variables that are manually added into this window

Inspecting Values of Variables



```
    C = 5 / 9 * (F - 32);

    K = C + 273.15;

    printf("%.2lfF = %.2lfC = %.2lfK\n", F, C, K);


    return 0;
}
```

Name	Value	Type
C	-0.0000000000000000	double
F	10.000000000000000	double
K	-9.2559631349317831e+061	double

Values of C, F, and K just before the execution of “K = C + 273.15;”

In Between Breakpoints

- When we want to continue running the program, we can use
  - Continue (F5)

    - Continue the execution until the next breakpoint is encountered.
  - Step Over (F10)
    - Execute one line of code and pause.
  - Step Into (F11)
    - Similar to Step Over if the line of code does not involve a function call.
    - But if the line of code involves a function call, go into the function if possible and pause.
  - Step Out (Shift+F11)
    - Similar to Continue but it will also pause just after returning from a function.

Stop and Restart

- Stop Debugging (Shift+F5) ■
  - – Stop debugging the program.

- Restart (Ctrl+Shift+F5) ↻
  - – Restart the program.