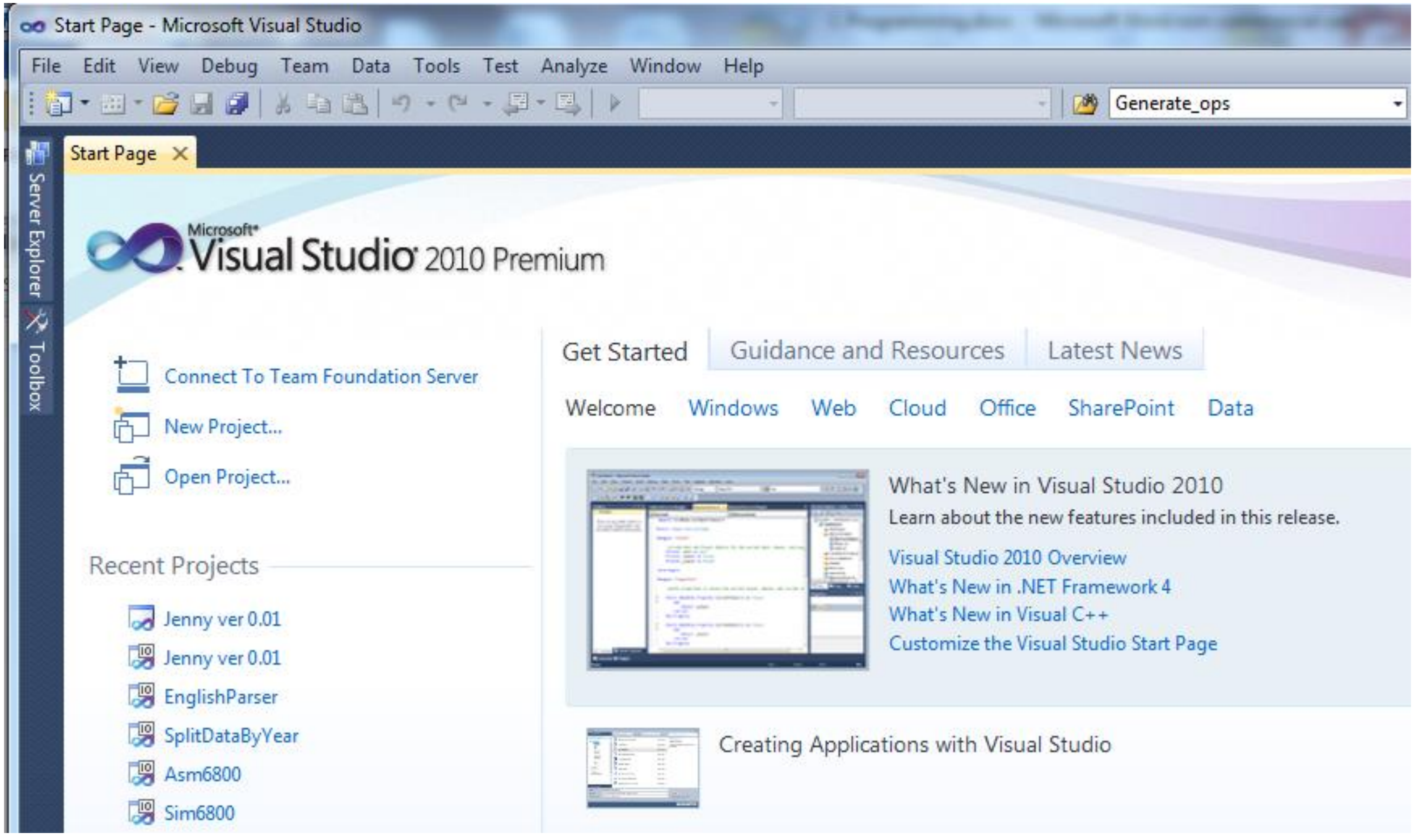
UFCF93-30-1 Computer and Network Systems

Computer Practical 4 Learning C programming

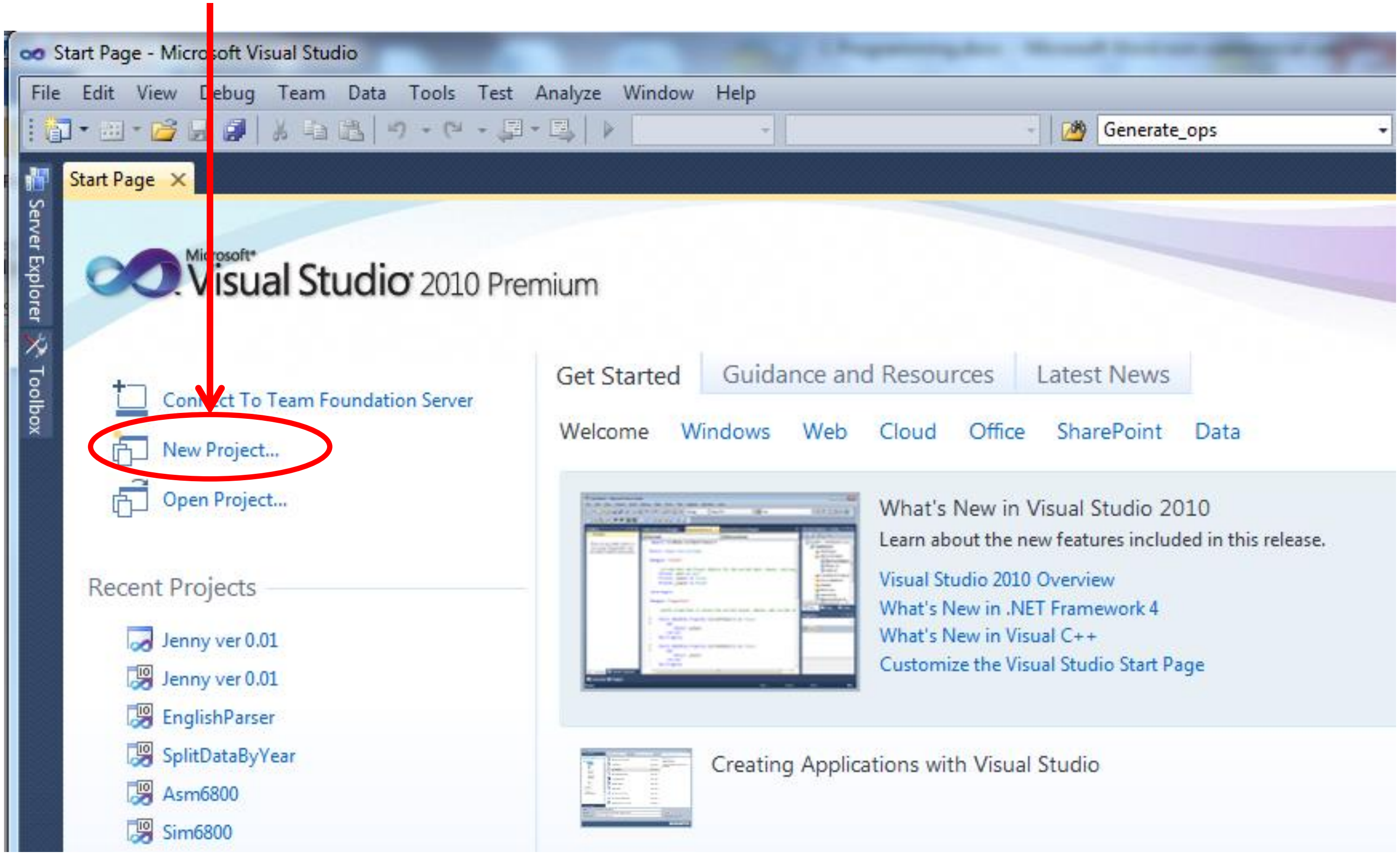
C Programming



Learn about the new features included in this release.

C Programming

1) Select New Project...



What's New in Visual Studio 2010

Learn about the new features included in this release.

Visual Studio 2010 Overview

What's New in .NET Framework 4

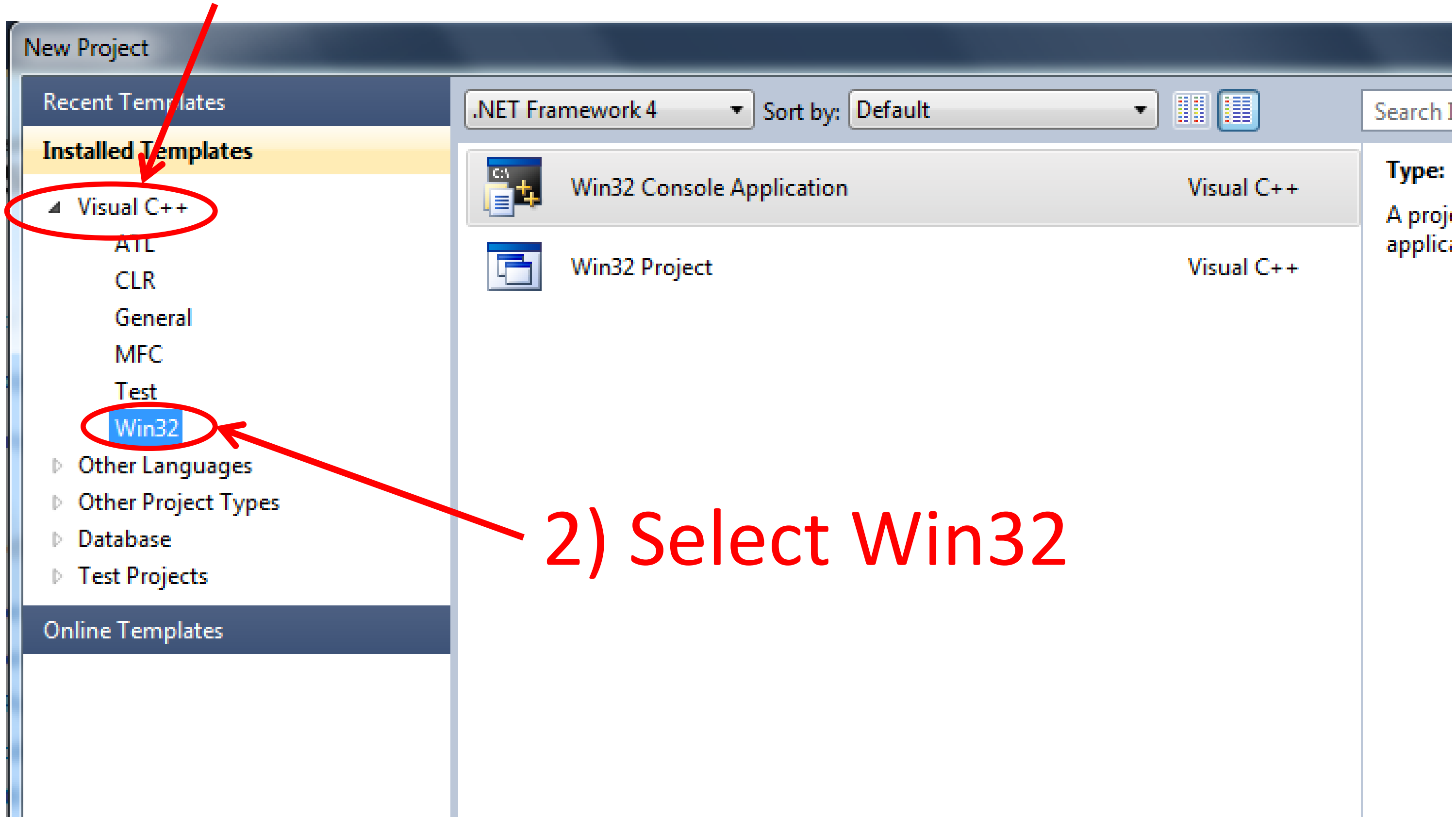
What's New in Visual C++

Customize the Visual Studio Start Page

Creating Applications with Visual Studio

C Programming

1) Select Visual C++

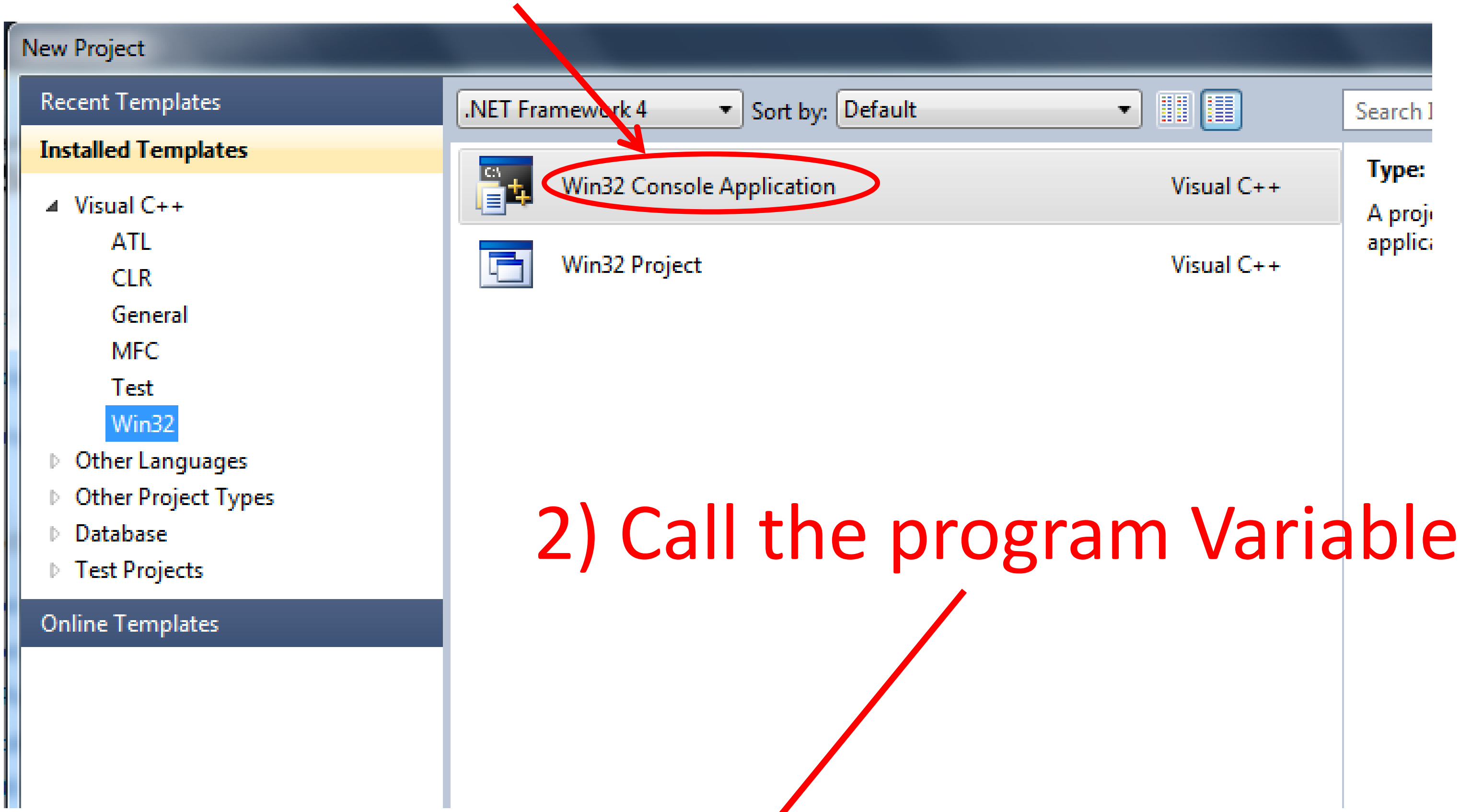


.NET Framework 4 Sort by: Default

2) Select Win32

C Programming

1) Select Win32 Console Application



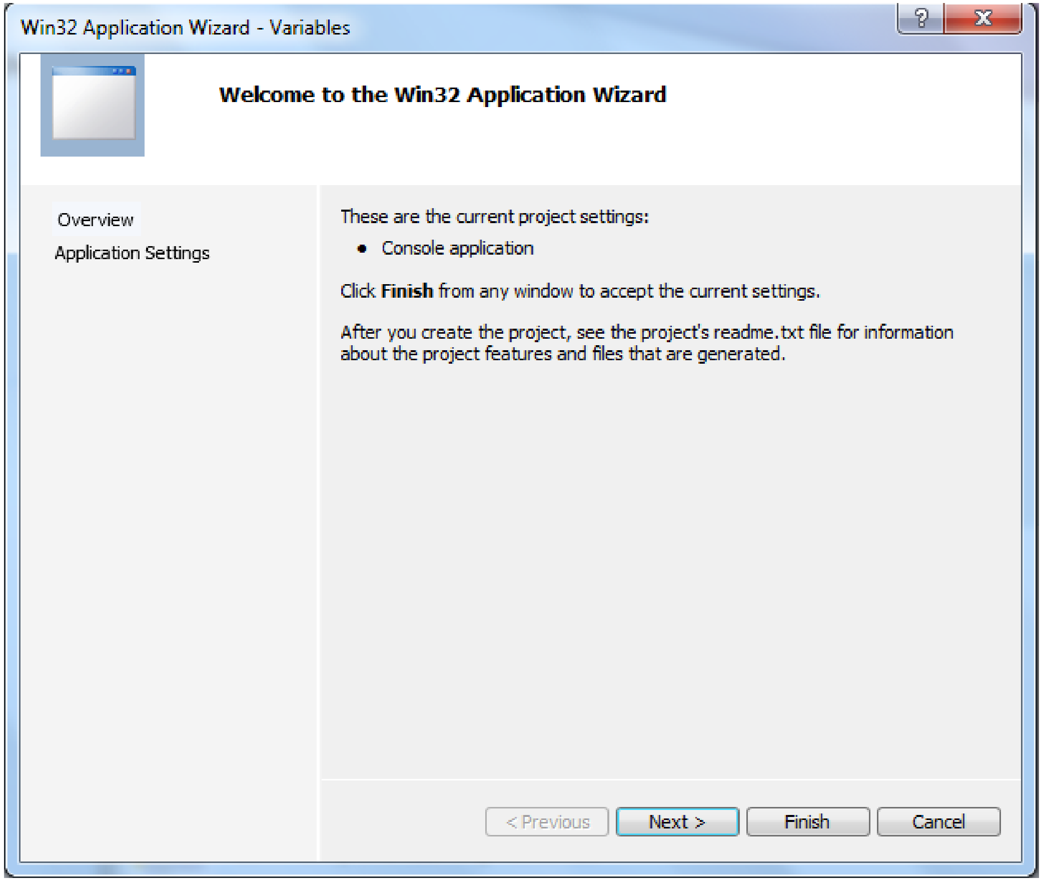
New Project

.NET Framework 4

Sort by: Default

2) Call the program Variables

C Programming



Welcome to the Win32 Application Wizard

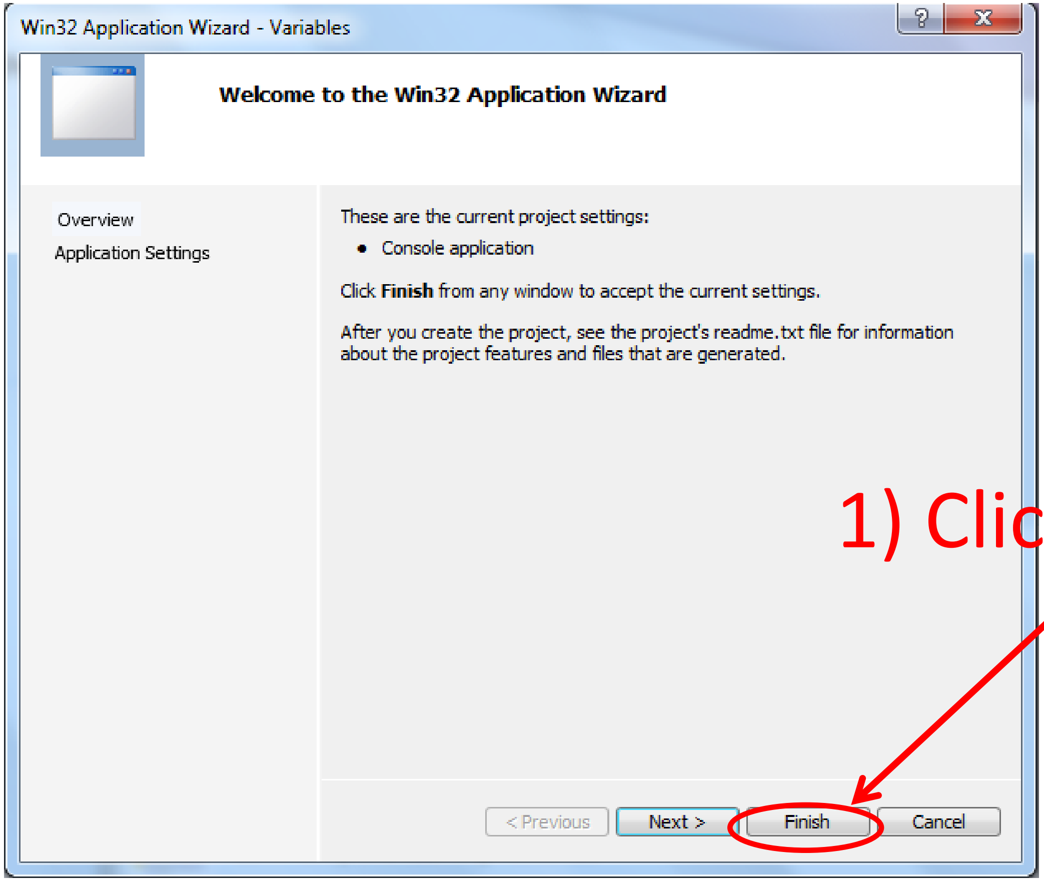
These are the current project settings:

- Console application

Click **Finish** from any window to accept the current settings.

After you create the project, see the project's readme.txt file for information about the project features and files that are generated.

C Programming



Welcome to the Win32 Application Wizard

These are the current project settings:

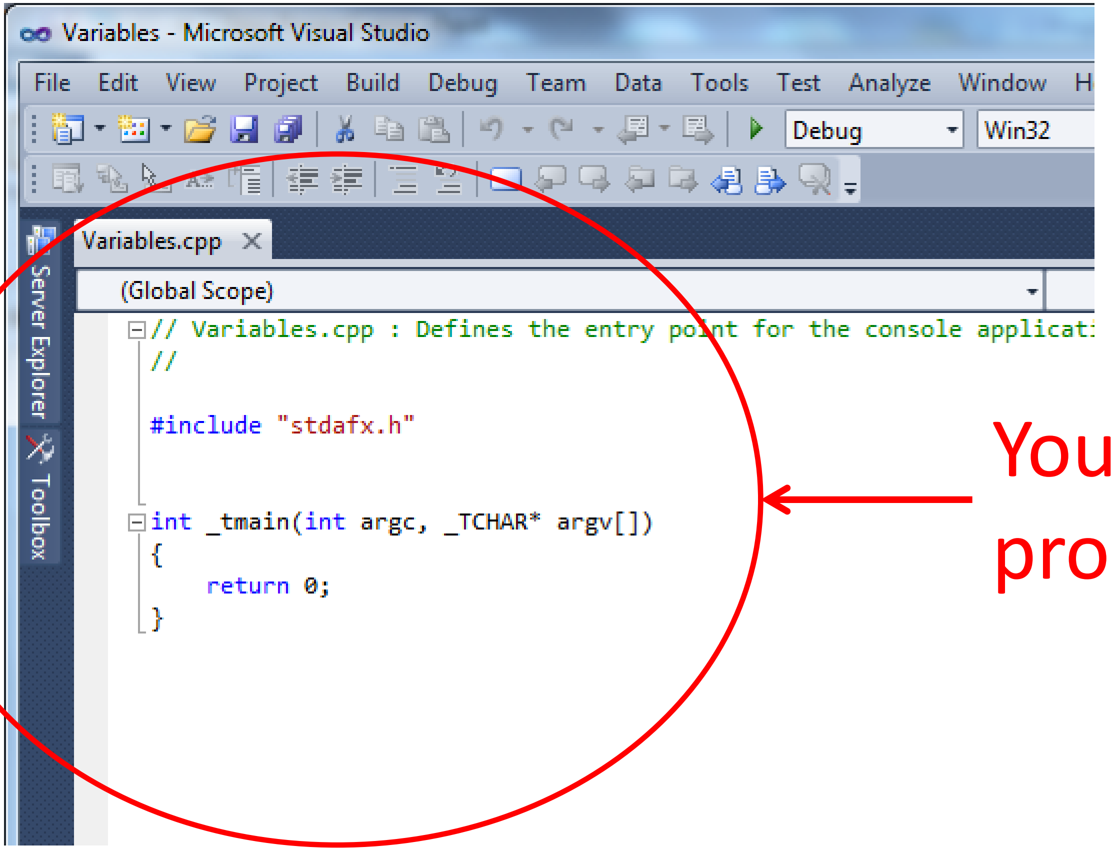
- Console application

Click **Finish** from any window to accept the current settings.

After you create the project, see the project's readme.txt file for information about the project features and files that are generated.

1) Click Finish

C Programming


Variables - Microsoft Visual Studio
File Edit View Project Build Debug Team Data Tools Test Analyze Window H
Debug
Win32
Variables.cpp
(Global Scope)
Server Explorer
Toolbox

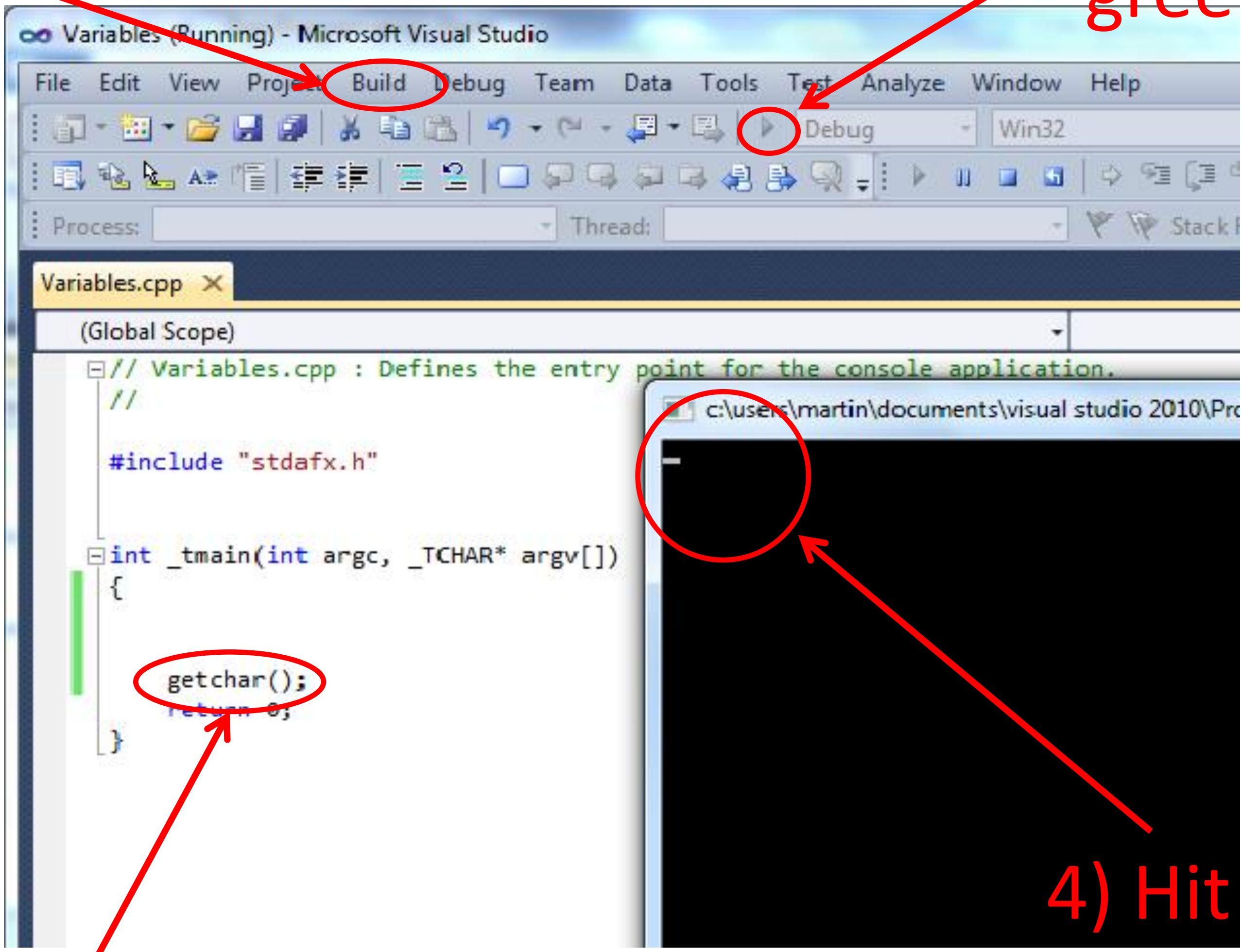
```
// Variables.cpp : Defines the entry point for the console applicat
//

#include "stdafx.h"


int _tmain(int argc, _TCHAR* argv[])
{
	return 0;
}
```

Your new program

C Programming

2) Click Build

3) Click the green arrow



```
// Variables.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"


int _tmain(int argc, _TCHAR* argv[])
{


    getchar();
    return 0;
}
```

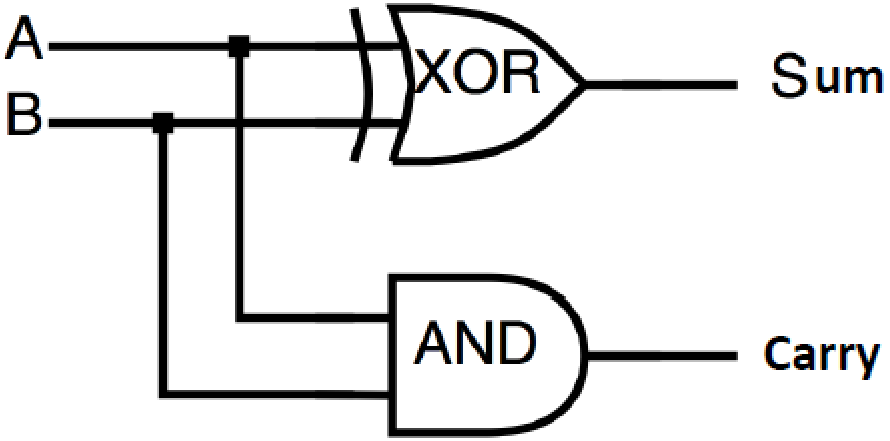
1) Add getchar();

4) Hit any key to close the program

C Programming – Adding Circuits

Half Adder Circuits add two single bit inputs together giving a sum and a carry

Looks like this...



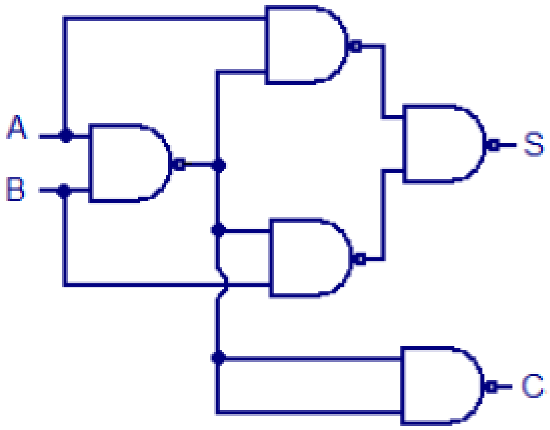
...and is called a half adder

Picture Source: Internet

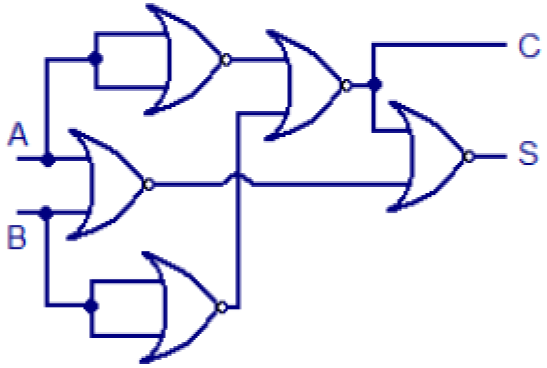
C Programming – Adding Circuits

Half Adder Circuits add two single bit inputs together giving a sum and a carry

Which can be re-written like this to remove the XOR…



Half adder using NAND logic

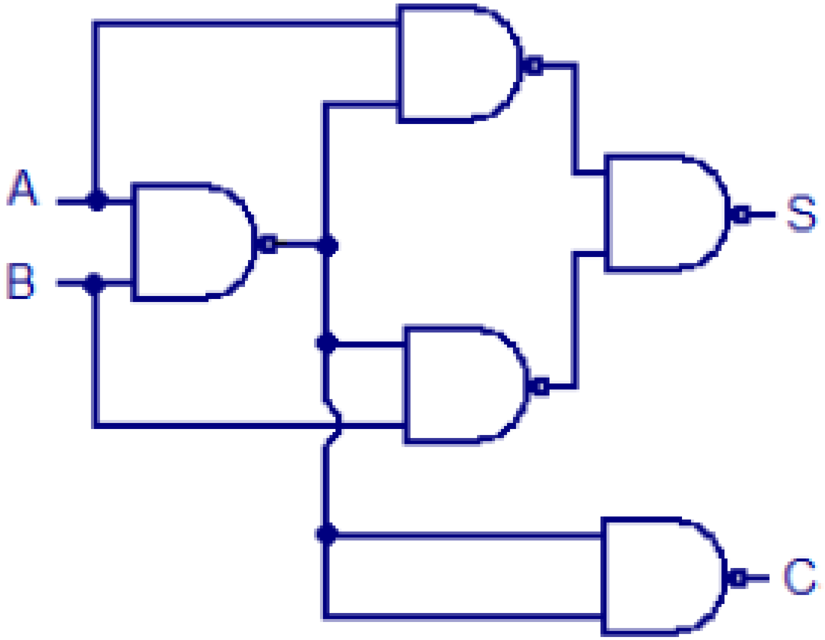


Half adder using NOR logic

Picture Source: Internet

C Programming – Adding Circuits

Lets create a C program to emulate a NAND gate half-adder circuit…

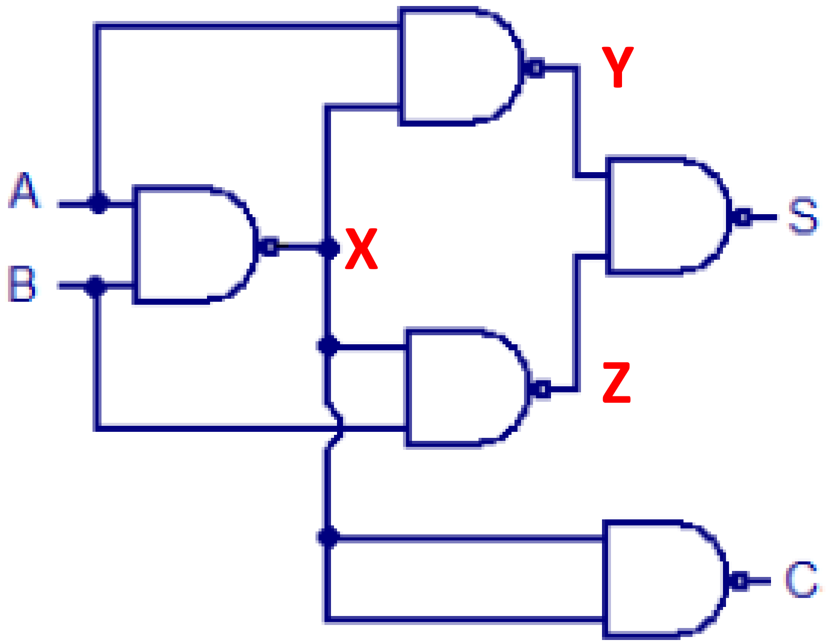


Half adder using NAND logic

Picture Source: Internet

C Programming – Adding Circuits

Lets create a C program to emulate a NAND gate half-adder circuit…



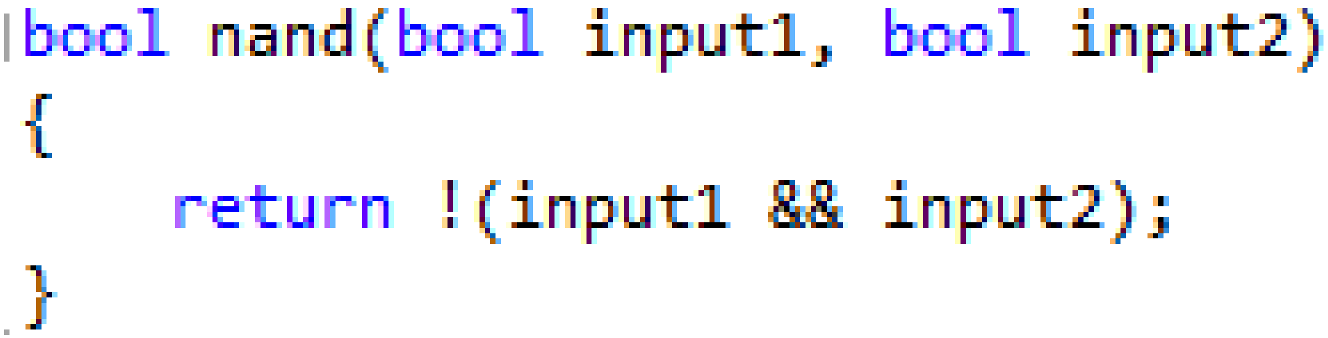
Half adder using NAND logic

Lets create some local variables X, Y and Z to make our life easier

Picture Source: Internet

C Programming – Adding Circuits

Now lets write some C code, add a function to do a NAND above main()…

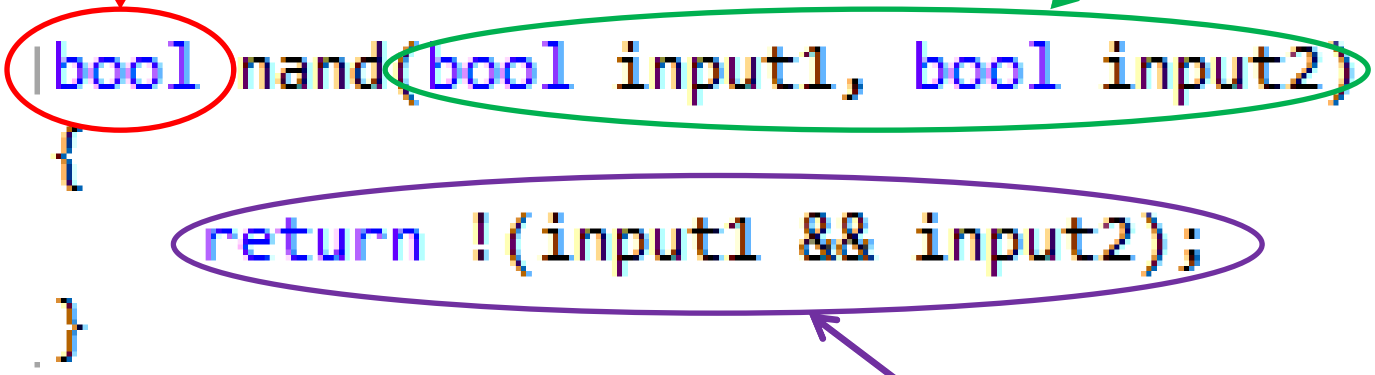


```
bool nand(bool input1, bool input2)
{
    return !(input1 && input2);
}
```

C Programming – Adding Circuits

This function returns a boolean value

There are two boolean inputs

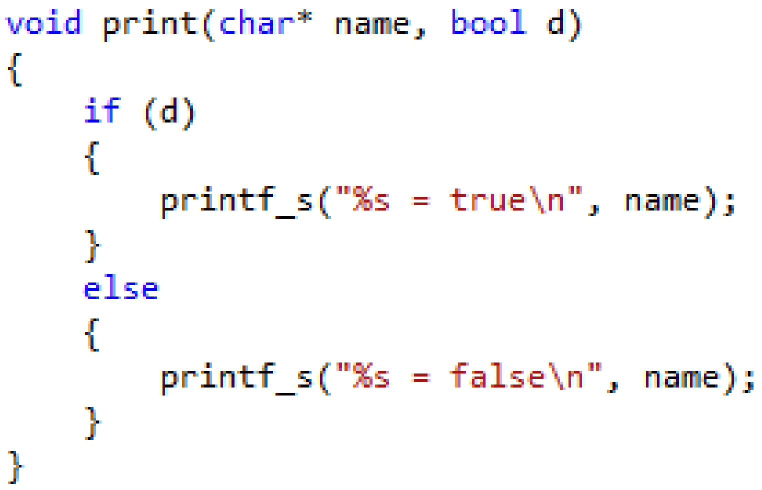


```
bool nand(bool input1, bool input2)
{
    return !(input1 && input2);
}
```

‘return’ returns a value of the same type as its function (boolean in this case)

C Programming – Adding Circuits

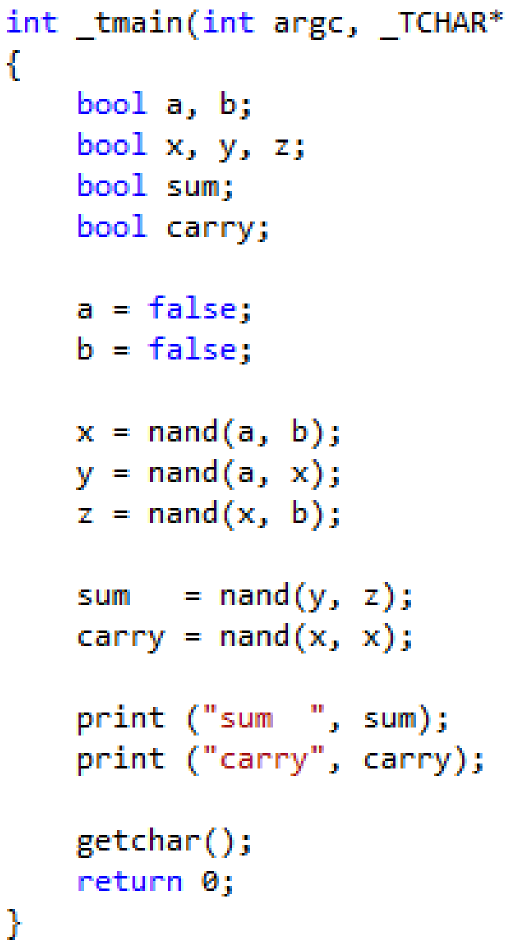
Add the print() function from last week…



```
void print(char* name, bool d)
{
    if (d)
    {
        printf_s("%s = true\n", name);
    }
    else
    {
        printf_s("%s = false\n", name);
    }
}
```

C Programming – Adding Circuits

Update main()...



```
int _tmain(int argc, _TCHAR* argv[])
{
    bool a, b;
    bool x, y, z;
    bool sum;
    bool carry;

    a = false;
    b = false;

    x = nand(a, b);
    y = nand(a, x);
    z = nand(x, b);

    sum   = nand(y, z);
    carry = nand(x, x);

    print ("sum  ", sum);
    print ("carry", carry);

    getchar();
    return 0;
}
```

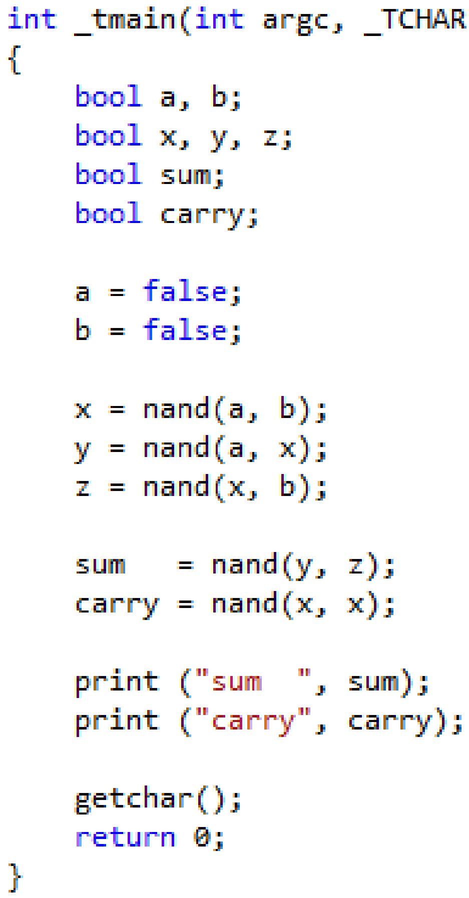
Allocate memory

Initialise inputs

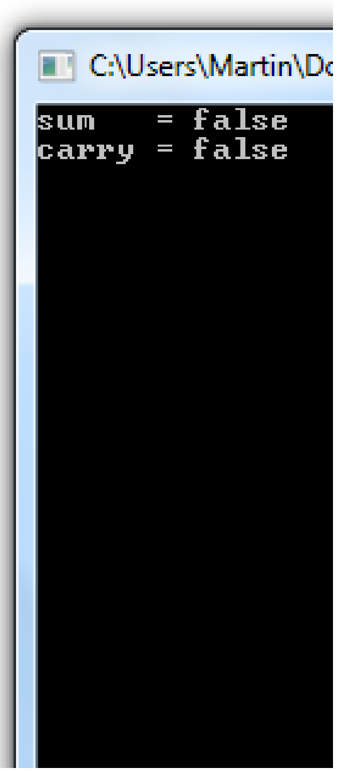
Half adder logic

C Programming – Adding Circuits

Build and run…



```
int _tmain(int argc, _TCHAR* argv[])
{
    bool a, b;
    bool x, y, z;
    bool sum;
    bool carry;

    a = false;
    b = false;

    x = nand(a, b);
    y = nand(a, x);
    z = nand(x, b);

    sum   = nand(y, z);
    carry = nand(x, x);

    print ("sum  ", sum);
    print ("carry", carry);

    getchar();
    return 0;
}
```



```
sum   = false
carry = false
```

a + b = sum, carry

As false is 0 and true is 1 then 0 + 0 = 0 carry 0

C Programming – Adding Circuits

Modify the program to complete the following table…

A	B	Sum	Carry
False	False	?	?
False	True	?	?
True	False	?	?
True	True	?	?

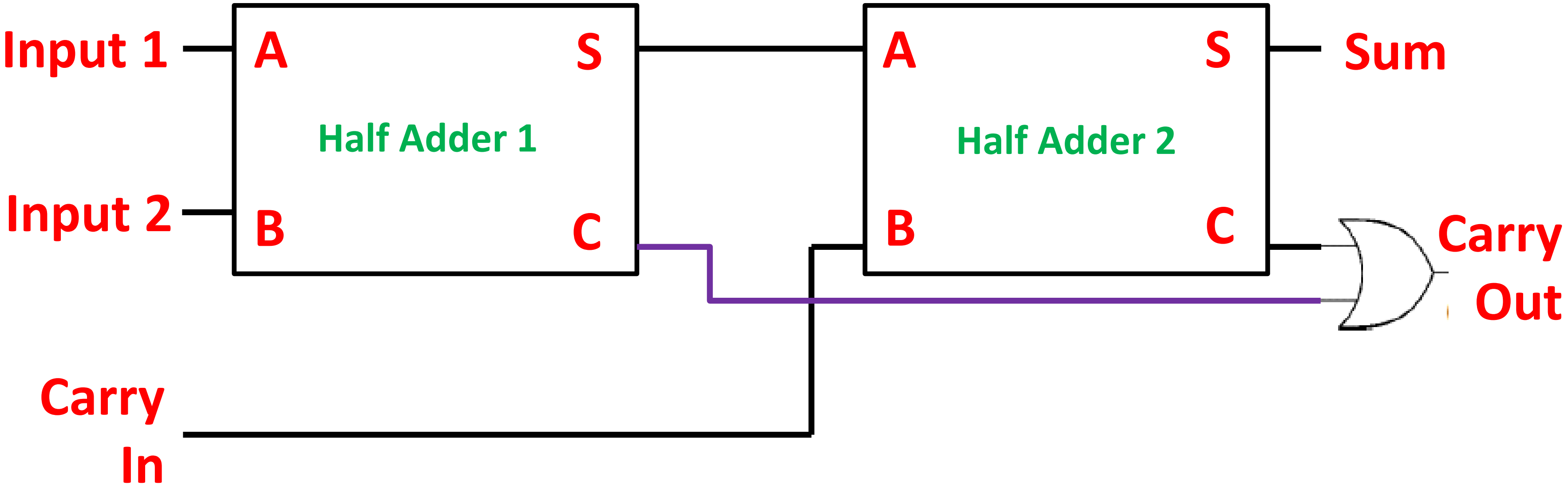
C Programming – Adding Circuits

Converting to zeroes and ones…

A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

C Programming – Adding Circuits

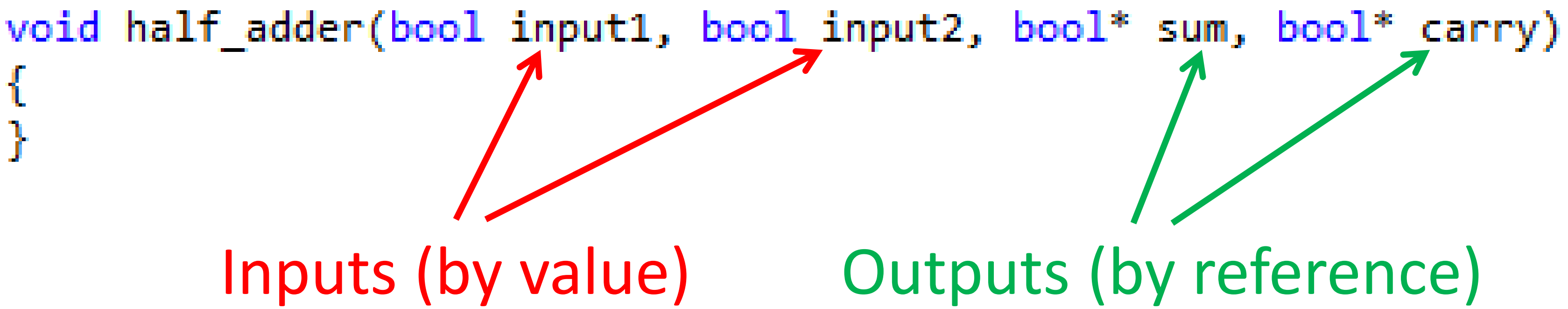
Full Adder Circuits add two single bit inputs and an input carry together giving a sum and an output carry



C Programming – Adding Circuits

Lets emulate a full adder in C code, first lets create a half adder circuit…

It is a little harder as we want two inputs and two outputs…

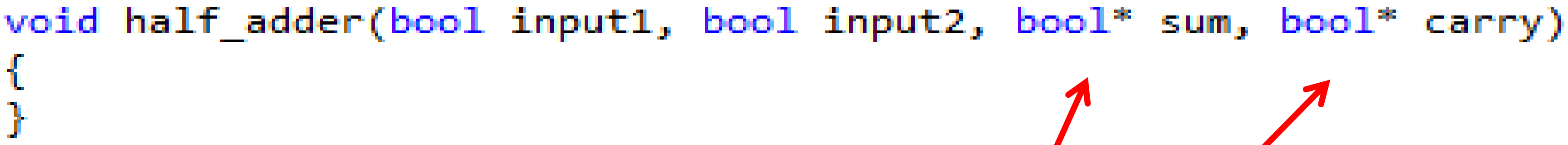


```
void half_adder(bool input1, bool input2, bool* sum, bool* carry)
{
}
```

Inputs (by value)

Outputs (by reference)

C Programming – Adding Circuits

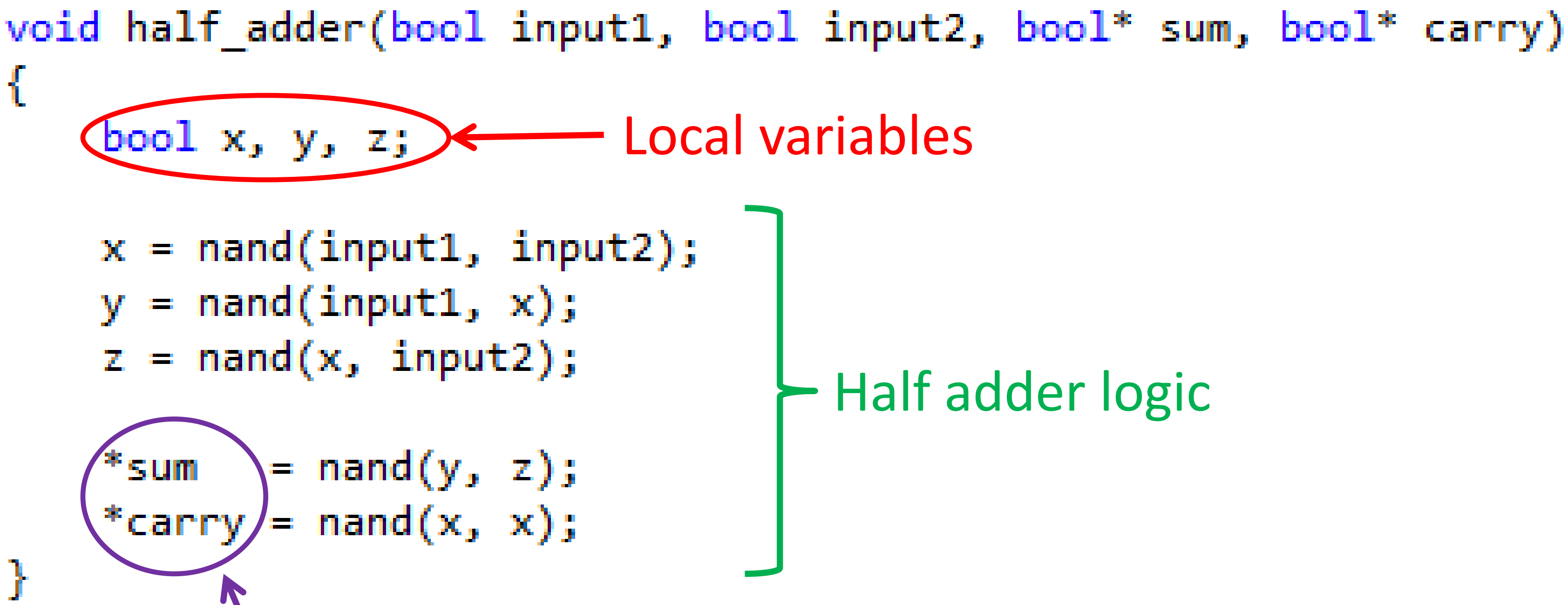


```c
void half_adder(bool input1, bool input2, bool* sum, bool* carry)
{
}
```

The address of the variables, sum and carry, will be passed to this function so that the function can write updated values to them (Note that sum and carry are **pointers**)

C Programming – Adding Circuits

Let’s finish off the half adder…



```
void half_adder(bool input1, bool input2, bool* sum, bool* carry)
{
    bool x, y, z;

    x = nand(input1, input2);
    y = nand(input1, x);
    z = nand(x, input2);

    *sum   = nand(y, z);
    *carry = nand(x, x);
}
```

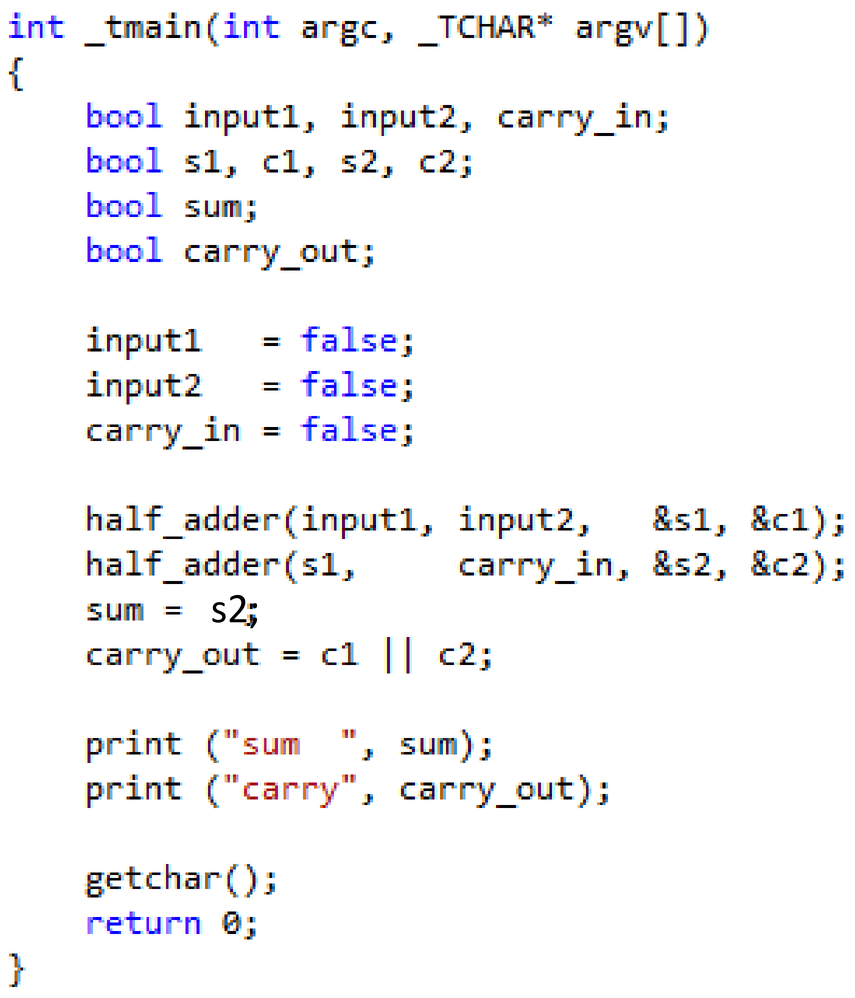
Local variables

Half adder logic

Write to what these pointers point at (known as dereferencing)

C Programming – Adding Circuits

Let’s update main()…



```
int _tmain(int argc, _TCHAR* argv[])
{
    bool input1, input2, carry_in;
    bool s1, c1, s2, c2;
    bool sum;
    bool carry_out;

    input1   = false;
    input2   = false;
    carry_in = false;

    half_adder(input1, input2,   &s1, &c1);
    half_adder(s1,     carry_in, &s2, &c2);
    sum =  s2;
    carry_out = c1 || c2;

    print ("sum  ", sum);
    print ("carry", carry_out);

    getchar();
    return 0;
}
```

Full adder logic

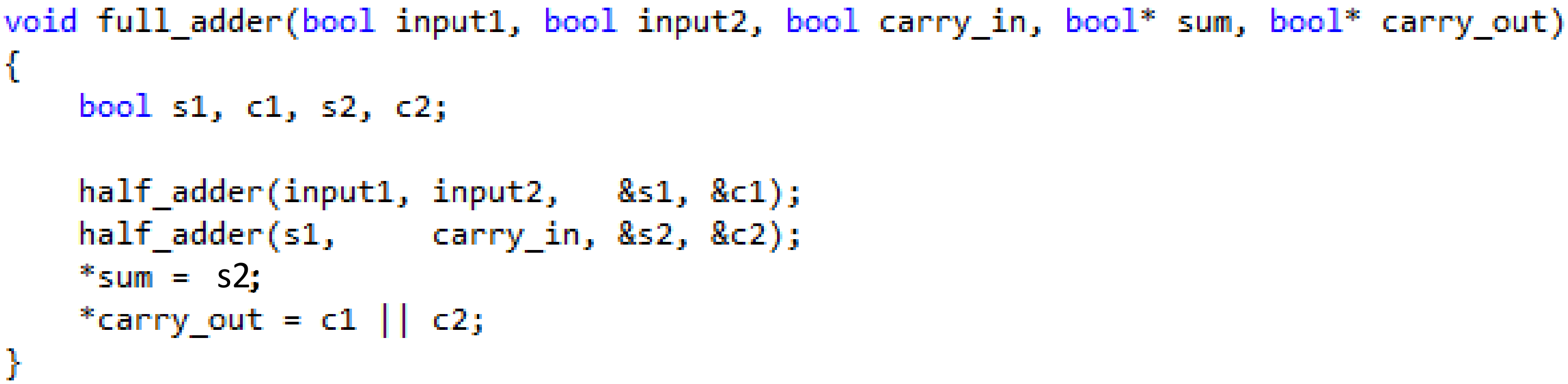
C Programming – Adding Circuits

Modify the program to fill in this table…

Input 1	Input 2	Carry in	Sum	Carry out
False	False	False	?	?
False	False	True	?	?
False	True	False	?	?
False	True	True	?	?
True	False	False	?	?
True	False	True	?	?
True	True	False	?	?
True	True	True	?	?

C Programming – Adding Circuits

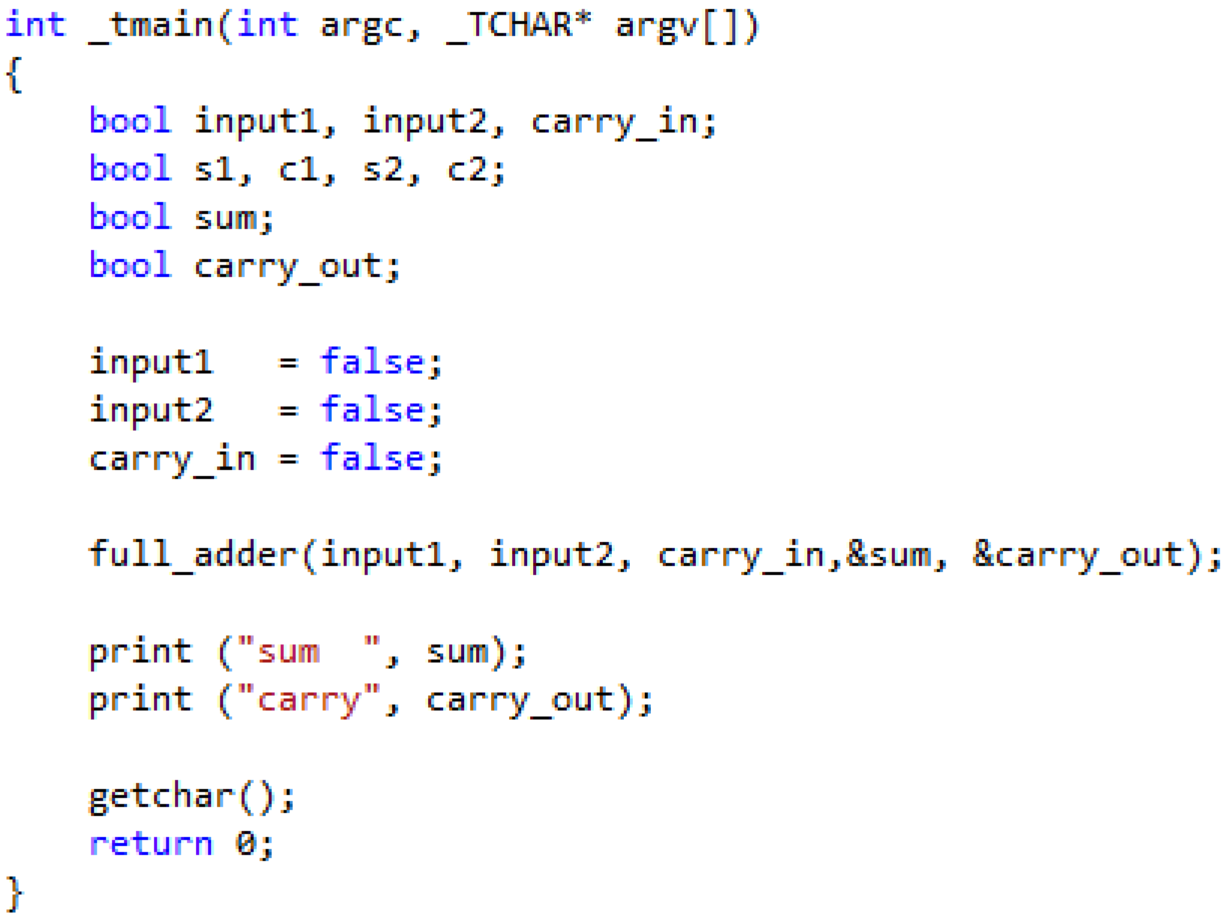
Let’s make a full adder function…



```
void full_adder(bool input1, bool input2, bool carry_in, bool* sum, bool* carry_out)
{
    bool s1, c1, s2, c2;

    half_adder(input1, input2,   &s1, &c1);
    half_adder(s1,     carry_in, &s2, &c2);
    *sum =  s2;
    *carry_out = c1 || c2;
}
```

C Programming – Adding Circuits

Let’s update main()…



```
int _tmain(int argc, _TCHAR* argv[])
{
    bool input1, input2, carry_in;
    bool s1, c1, s2, c2;
    bool sum;
    bool carry_out;

    input1   = false;
    input2   = false;
    carry_in = false;

    full_adder(input1, input2, carry_in,&sum, &carry_out);

    print ("sum  ", sum);
    print ("carry", carry_out);

    getchar();
    return 0;
}
```

C Programming – Adding Circuits

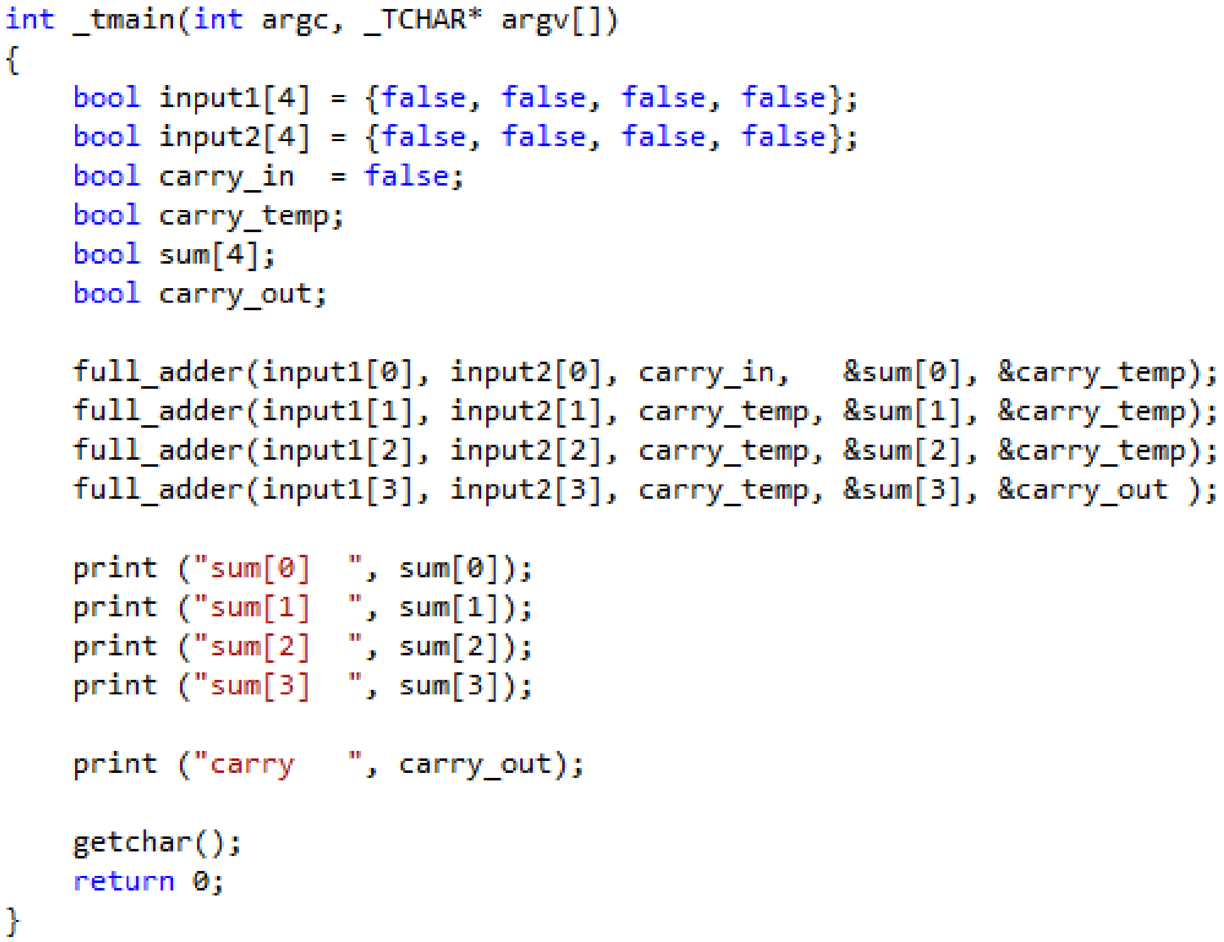
Modify the program to fill in this table…

Input 1	Input 2	Carry in	Sum	Carry out
False	False	False	?	?
False	False	True	?	?
False	True	False	?	?
False	True	True	?	?
True	False	False	?	?
True	False	True	?	?
True	True	False	?	?
True	True	True	?	?

…is it the same as before?

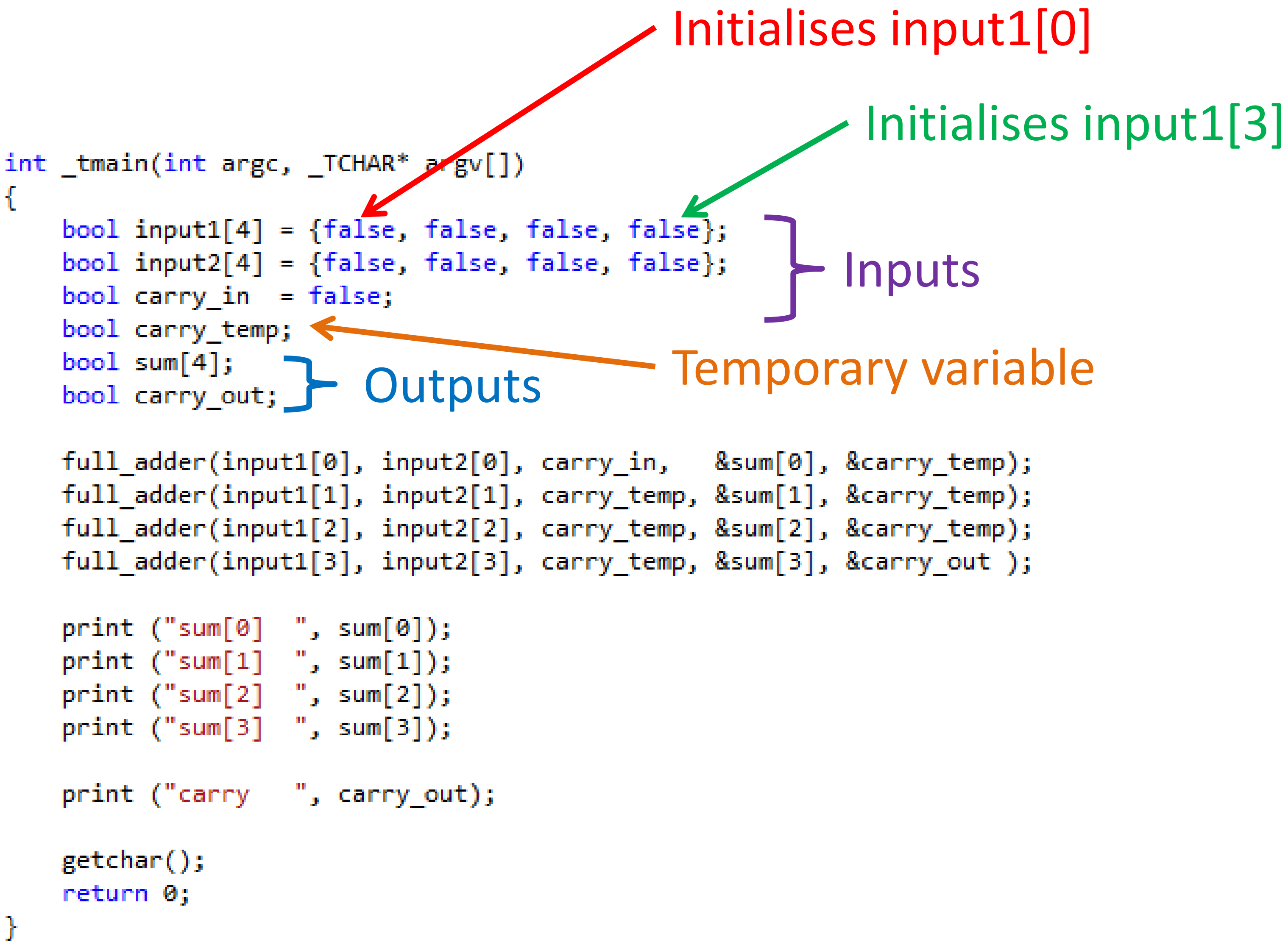
C Programming – Adding Circuits

Let’s do some 4-bit maths…



```
int _tmain(int argc, _TCHAR* argv[])
{
    bool input1[4] = {false, false, false, false};
    bool input2[4] = {false, false, false, false};
    bool carry_in  = false;
    bool carry_temp;
    bool sum[4];
    bool carry_out;

    full_adder(input1[0], input2[0], carry_in,   &sum[0], &carry_temp);
    full_adder(input1[1], input2[1], carry_temp, &sum[1], &carry_temp);
    full_adder(input1[2], input2[2], carry_temp, &sum[2], &carry_temp);
    full_adder(input1[3], input2[3], carry_temp, &sum[3], &carry_out );

    print ("sum[0]  ", sum[0]);
    print ("sum[1]  ", sum[1]);
    print ("sum[2]  ", sum[2]);
    print ("sum[3]  ", sum[3]);

    print ("carry   ", carry_out);

    getchar();
    return 0;
}
```

C Programming – Adding Circuits



```
int _tmain(int argc, _TCHAR* argv[])
{
    bool input1[4] = {false, false, false, false};
    bool input2[4] = {false, false, false, false};
    bool carry_in  = false;
    bool carry_temp;
    bool sum[4];
    bool carry_out;

    full_adder(input1[0], input2[0], carry_in,   &sum[0], &carry_temp);
    full_adder(input1[1], input2[1], carry_temp, &sum[1], &carry_temp);
    full_adder(input1[2], input2[2], carry_temp, &sum[2], &carry_temp);
    full_adder(input1[3], input2[3], carry_temp, &sum[3], &carry_out );

    print ("sum[0]  ", sum[0]);
    print ("sum[1]  ", sum[1]);
    print ("sum[2]  ", sum[2]);
    print ("sum[3]  ", sum[3]);

    print ("carry   ", carry_out);

    getchar();
    return 0;
}
```

Initialises input1[0]

Initialises input1[3]

Inputs

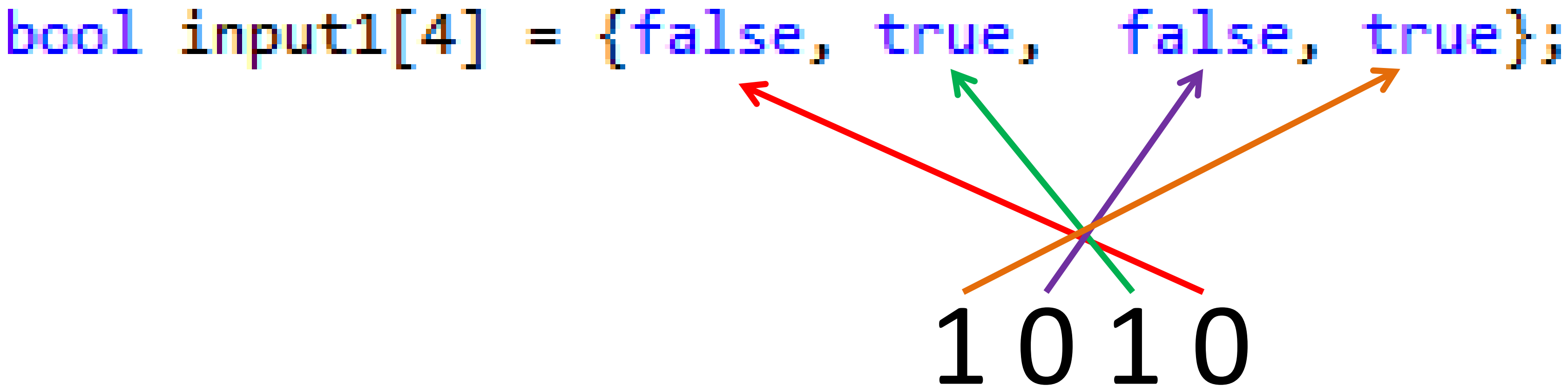
Temporary variable

Outputs

C Programming – Adding Circuits

Remember that the 4-bit inputs are initialised in the order of bit 0 to 3

So, to set input1 to 1010 in binary we write…



```
bool input1[4] = {false, true,  false, true};
```

1 0 1 0

C Programming – Adding Circuits

Modify the program to fill in this table…

Input 1	Input 2	Carry in	Sum	Carry out
1010	0000	0	?	?
1010	0101	0	?	?
1010	0101	1	?	?
1111	0000	0	?	?
1111	0000	1	?	?
1111	1111	0	?	?
1111	1111	1	?	?
1011	0111	0	?	?

C Programming – Adding Circuits

Hopefully you can see how simple **NAND** and **NOR** gates can be used to do **integer addition**

The End