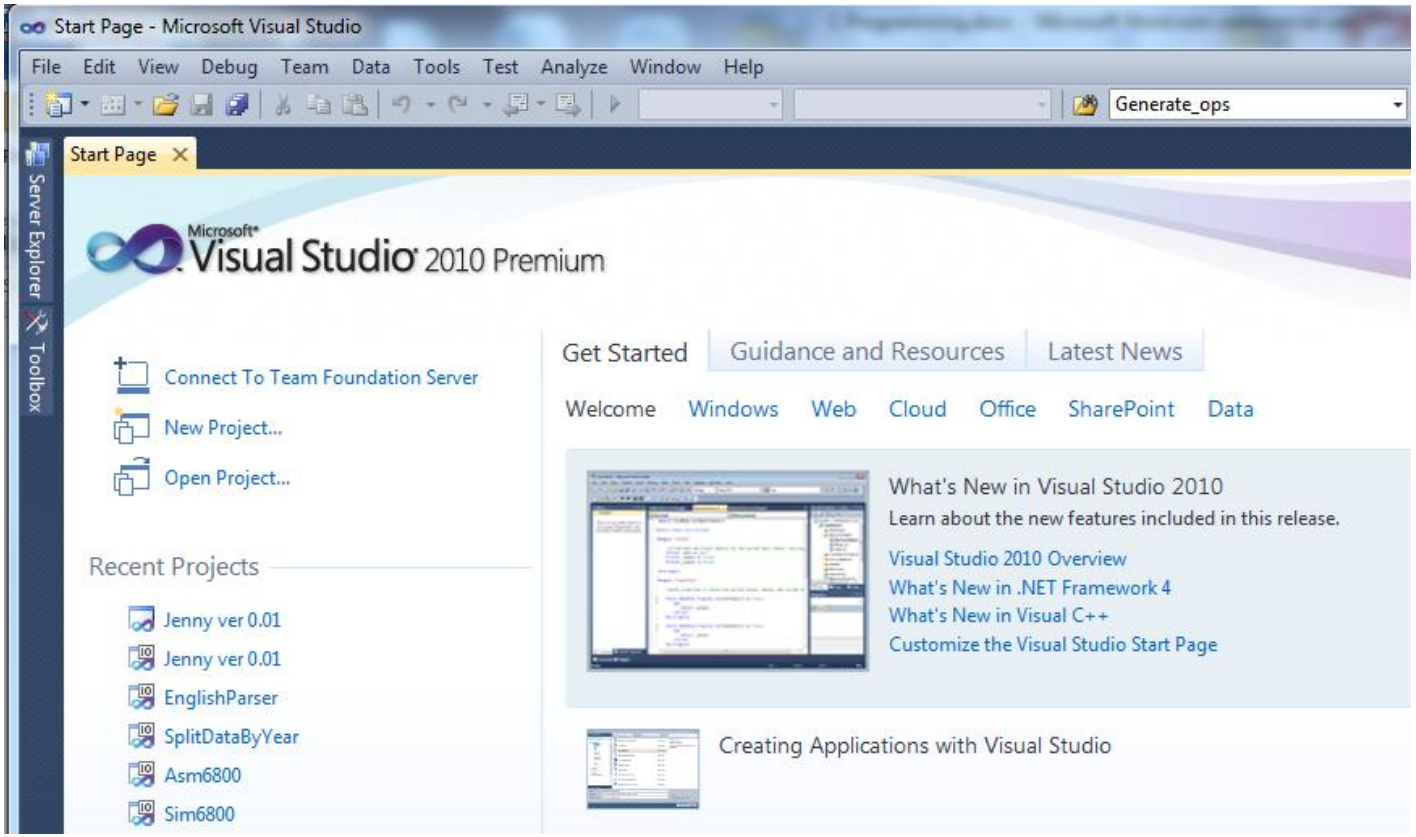
UFCF93-30-1 Computer and Network Systems

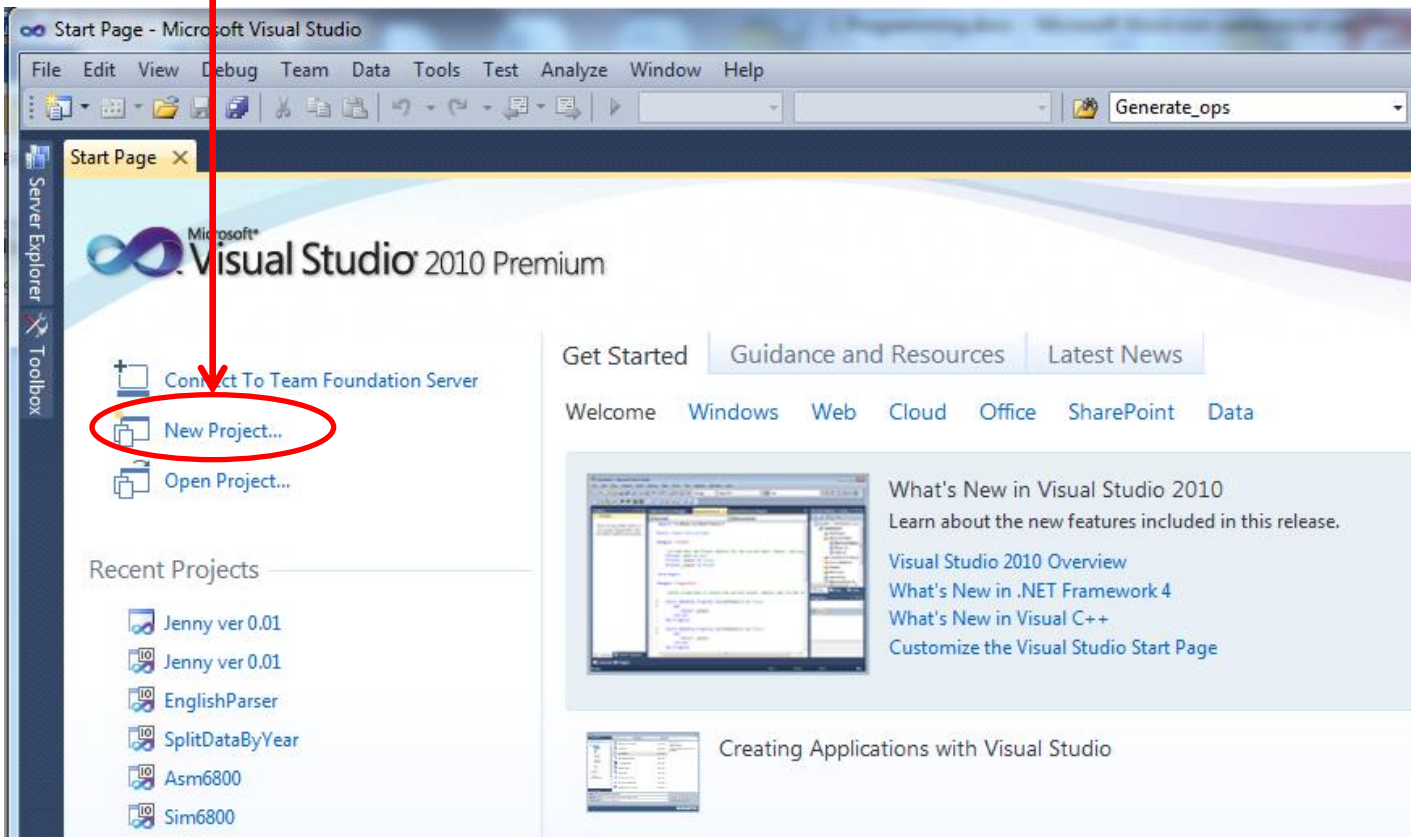
Computer Practical 2 Learning C programming

C Programming



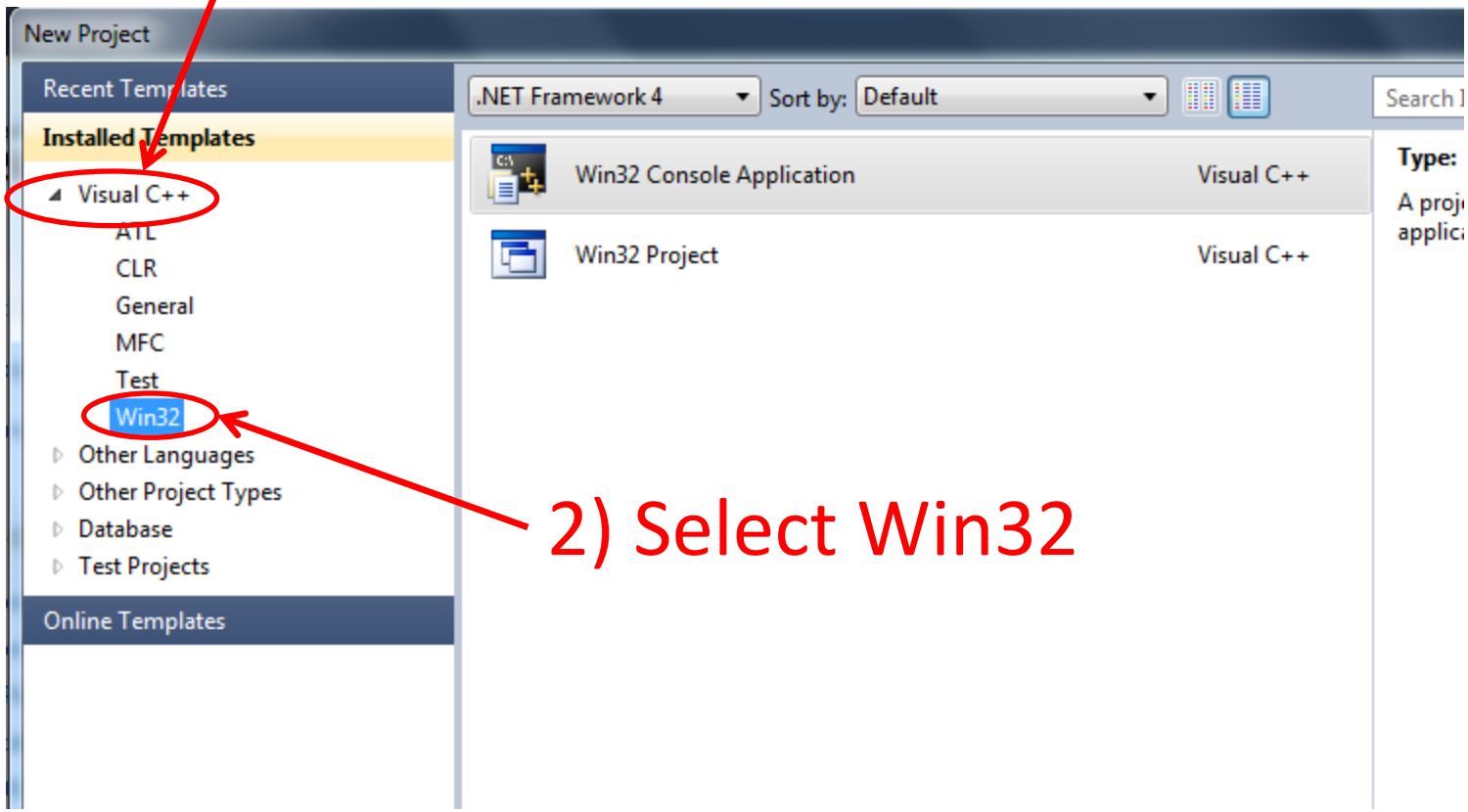
C Programming

1) Select New Project…



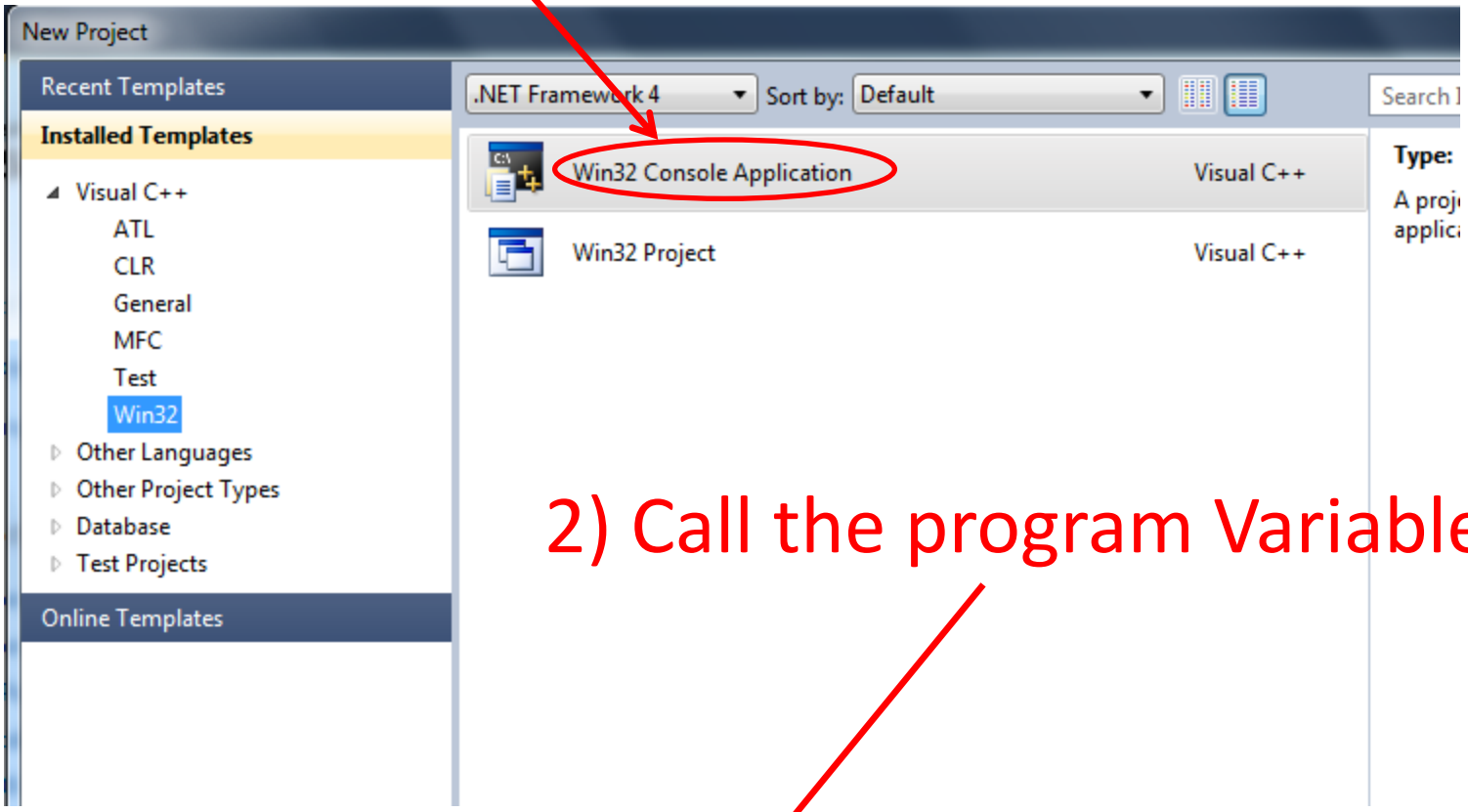
C Programming

1) Select Visual C++



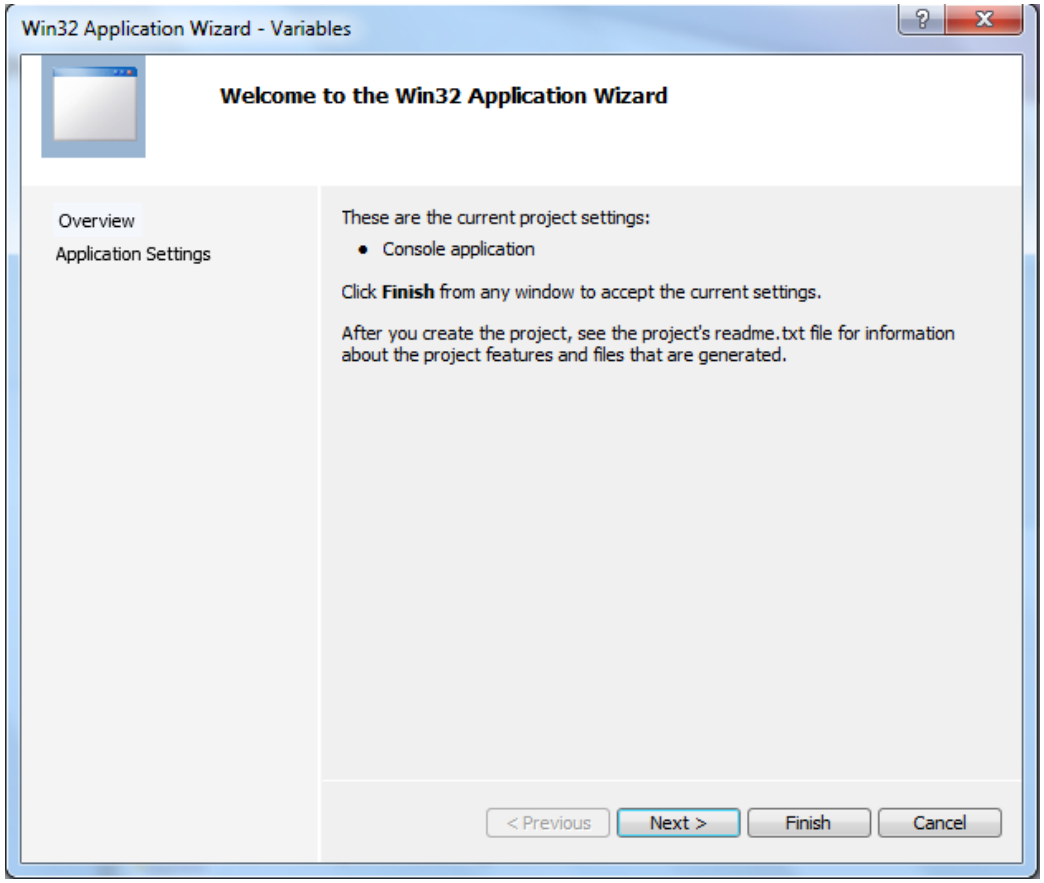
C Programming

1) Select Win32 Console Application

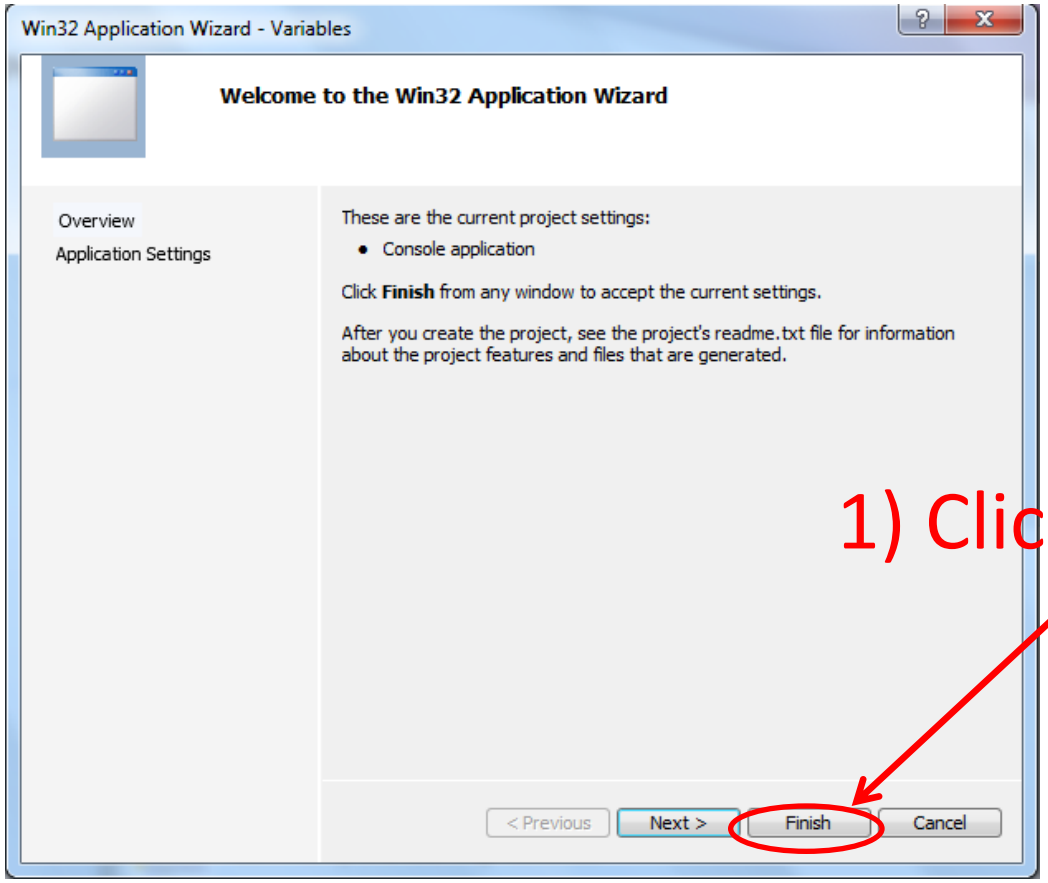


2) Call the program Variables

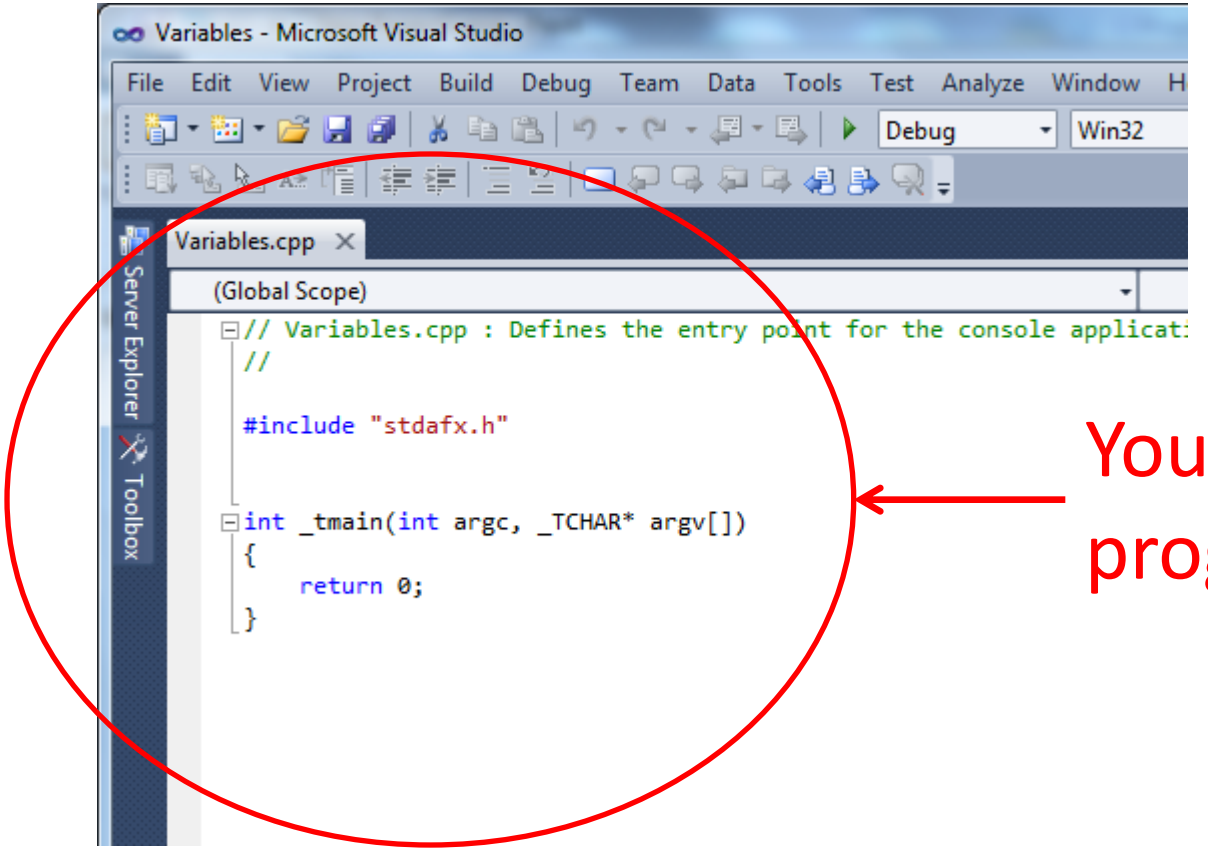
C Programming



C Programming



C Programming

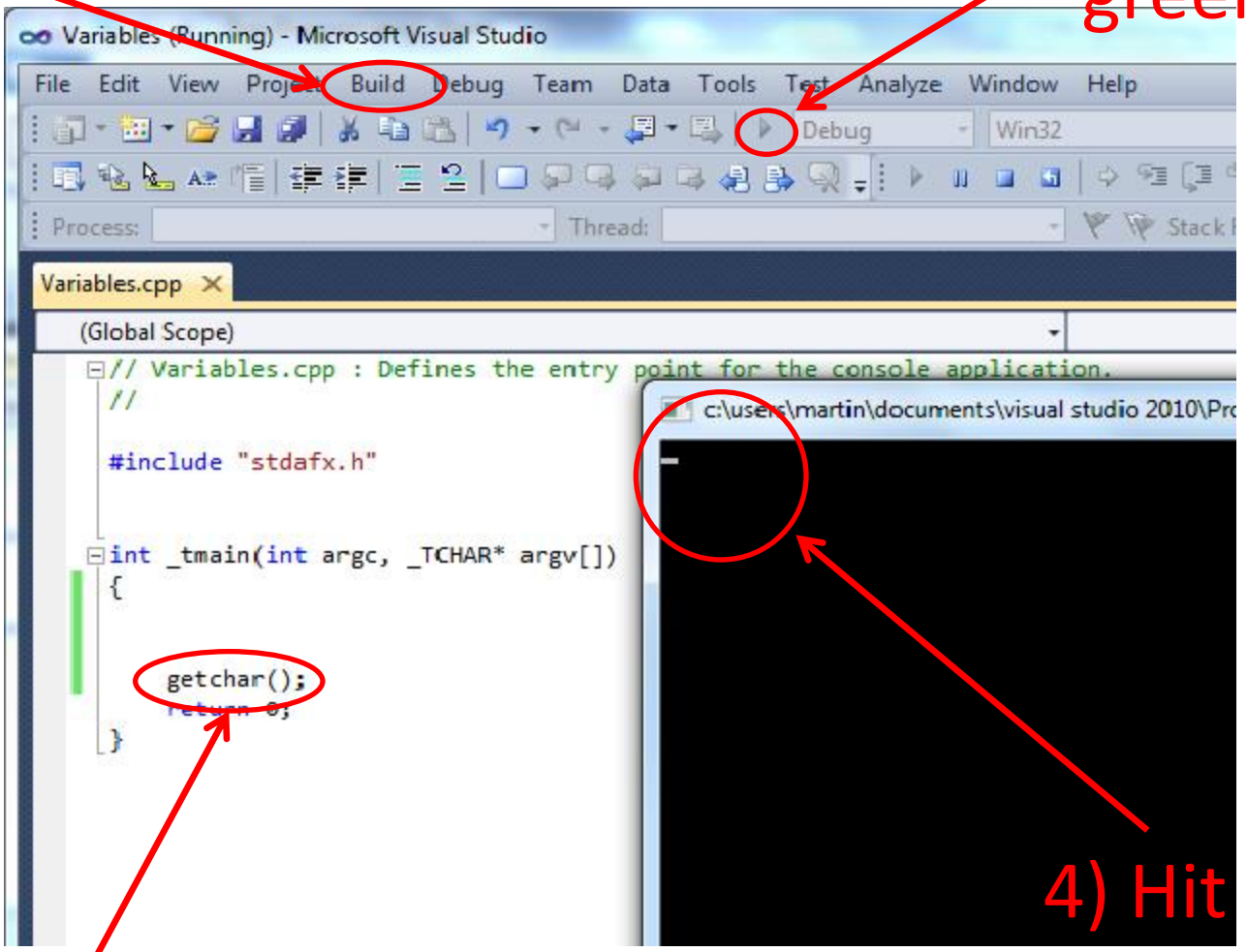


Your new program

C Programming

2) Click Build

3) Click the green arrow



1) Add getchar();

4) Hit any key to close the program

C Programming - Booleans

In the previous practical we looked at different types of **integer** and **floating point** variables

In this lecture we look an **boolean** variables

A **boolean** variable can take the values of **true** or **false**

C Programming - Booleans

Integer and **floating point** variables have a set of operators associated with them; **add**, **subtract**, **multiply** and **divide** (**+**, **-**, * and **/**)

Boolean variables have their own set of logical operators; **and**, **or**, **xor** and **not** (**&&**, **||**, **^^** and **!**)

C Programming - Booleans

How are boolean variables stored?

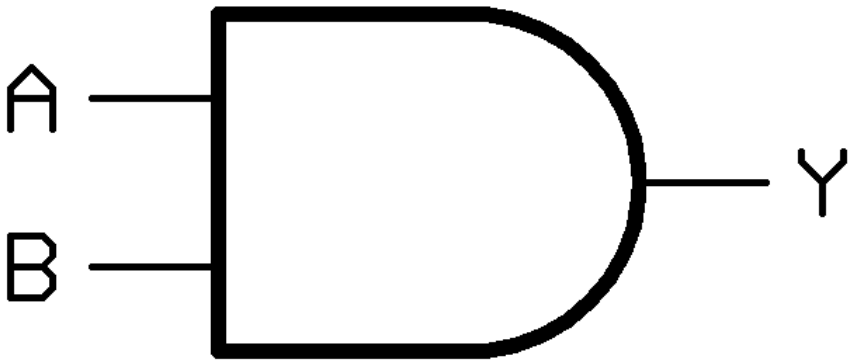
C Programming - Booleans

How are boolean variables stored?

Even though they only require one bit of storage that are typically stored as an integer and so could take up 8, 16, 32 or 64 bits of memory

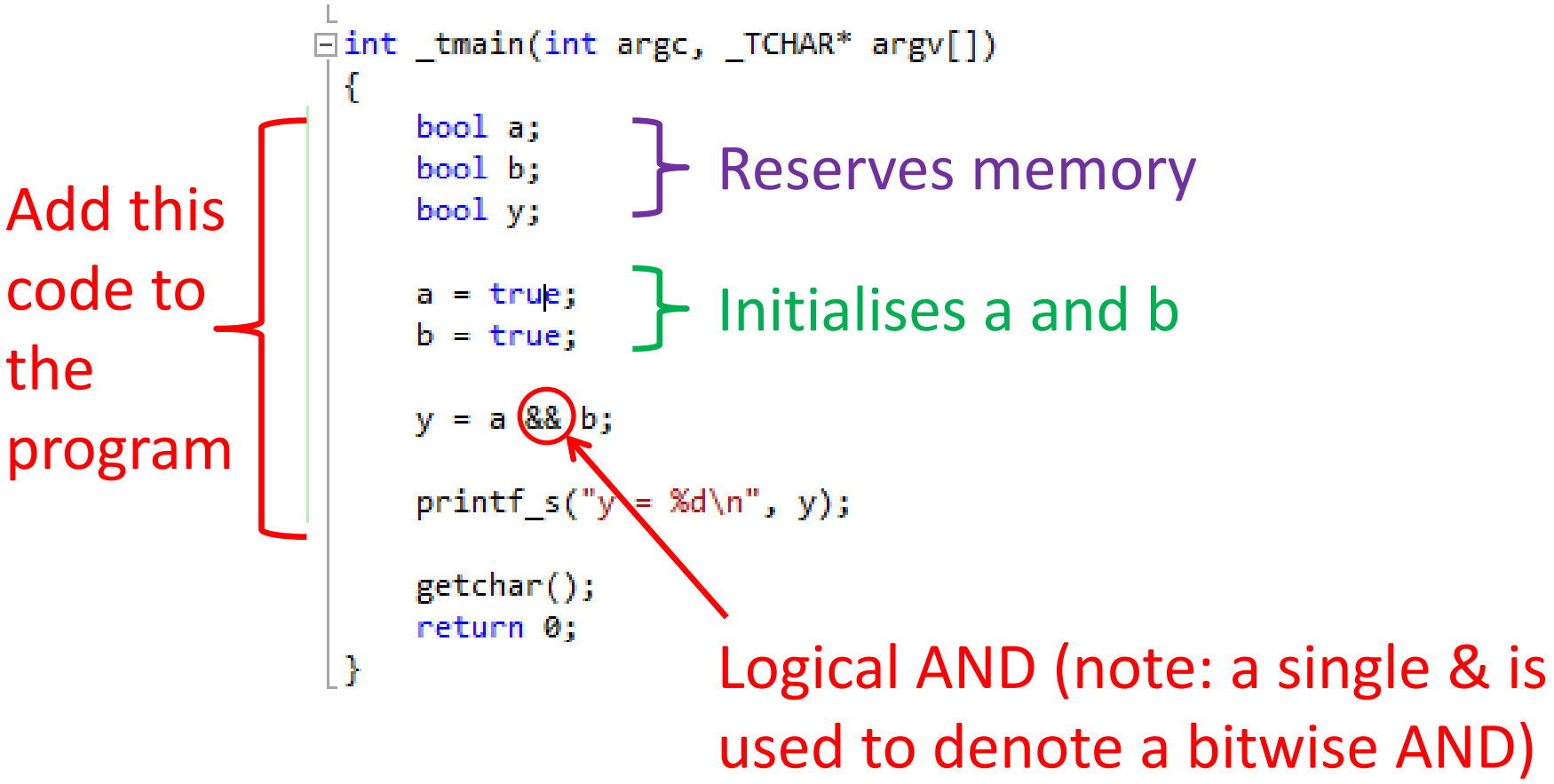
False is represented by 0 and true by -1 (although in Visual Studio by 1)

In electronics an AND gate is represented by the following symbol



C Programming - Booleans

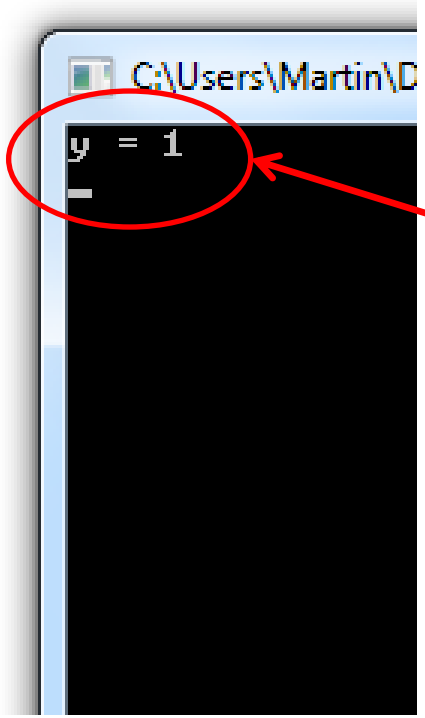
Lets implement this in a C program…



C Programming - Booleans

Build and run…

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

    a = true;
    b = true;

    y = a && b;

    printf_s("y = %d\n", y);

    getchar();
    return 0;
}
```



Remember in Visual Studio false is 0 and true is 1 (it will be different with other compilers)

C Programming - Booleans

We can alter the program to print out true or false…

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

    a = true;
    b = true;

    y = a && b;

    if (y)
    {
        printf_s("y = true\n");
    }
    else
    {
        printf_s("y = false\n");
    }

    getchar();
    return 0;
}
```

Change the print statement to this

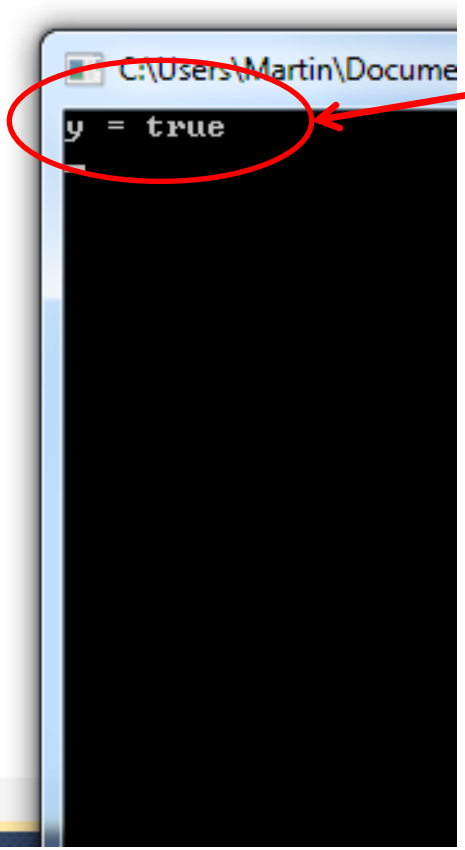
If y is true then print "y = true"

If y is false then print "y = false"

C Programming - Booleans

Build and run…

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

    a = true;
    b = true;

    y = a && b;

    if (y)
    {
        printf_s("y = true\n");
    }
    else
    {
        printf_s("y = false\n");
    }
```



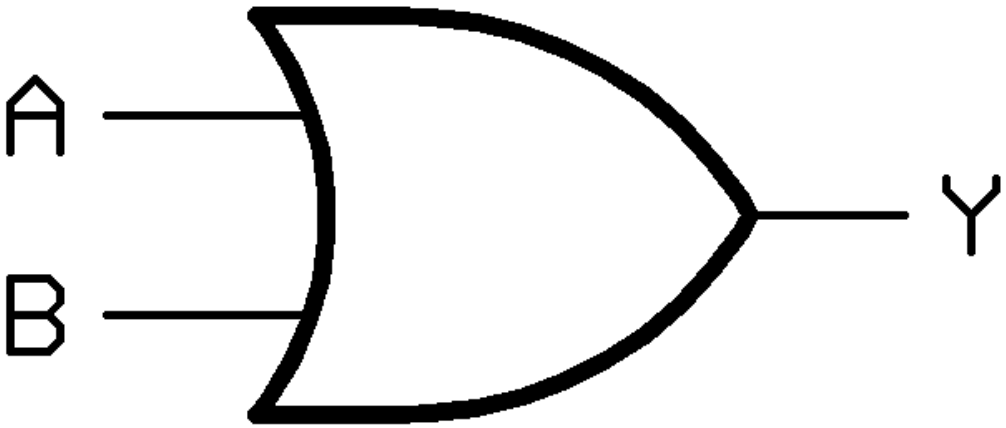
Now it says
y is true

C Programming - Booleans

Modify and run your program to complete the truth table for the AND gate

A	B	Y
True	True	?
True	False	?
False	True	?
False	False	?

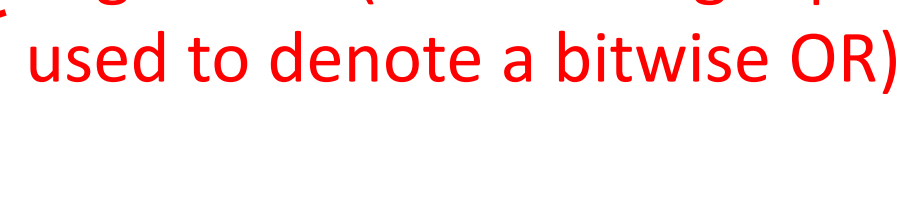
In electronics an OR gate is represented by the following symbol



C Programming - Booleans

Modify your program to emulate the OR gate…

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

    a = true;
    b = true;

    y = a || b;

    if (y)
    {
        printf_s("y = true\n");
    }
    else
    {
        printf_s("y = false\n");
    }

    getchar();
    return 0;
}
```



C Programming - Booleans

Build and run…

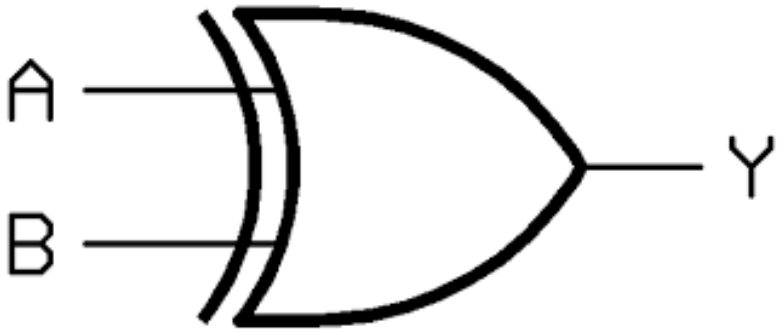
```
int _tmain(int argc, _TCHAR* argv[])
{
    bool a;
    bool b;
    bool y;

    a = true;
    b = true;

    y = a || b;

    if (y)
    {
        printf_s("y = true\n");
    }
    else
    {
        printf_s("y = false\n");
    }
```

C Programming - Booleans

Modify and run your program to complete the truth table for the OR gate

A	B	Y
True	True	?
True	False	?
False	True	?
False	False	?

In electronics an XOR gate is represented by the following symbol



C Programming - Booleans

Modify your program to emulate the XOR gate…

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

    a = true;
    b = true;

    y = a ^ b;

    if (y)
    {
        printf_s("y = true\n");
    }
    else
    {
        printf_s("y = false\n");
    }

    getchar();
    return 0;
}
```

Bitwise XOR (note: a logical ^^ is not valid)

C Programming - Booleans

Build and run…

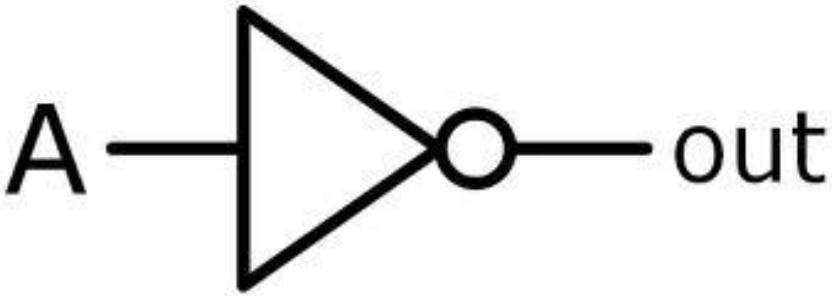
```
int _tmain(int argc, _TCHAR* argv[])
{
    bool a;
    bool b;
    bool y;

    a = true;
    b = true;

    y = a ^ b;

    if (y)
    {
        printf_s("y = true\n");
    }
    else
    {
        printf_s("y = false\n");
    }
```

C Programming - Booleans

Modify and run your program to complete the truth table for the XOR gate

A	B	Y
True	True	?
True	False	?
False	True	?
False	False	?

In electronics an NOT gate is represented by the following symbol



C Programming - Booleans

Lets implement this in a C program…

Modify this code

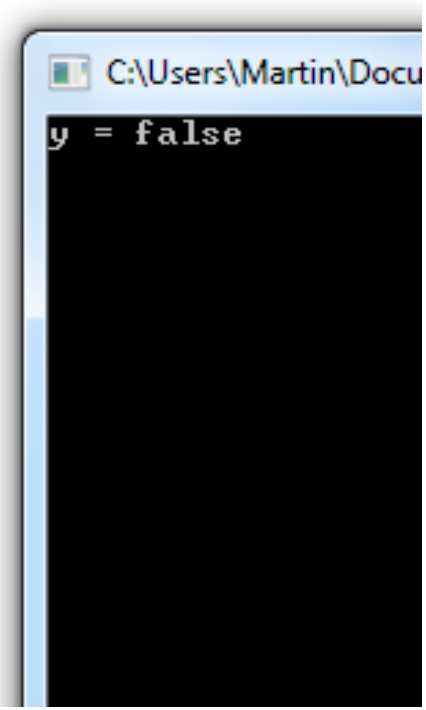
```
int _tmain(int argc, _TCHAR* argv[])
{
    bool a;
    bool out;

    a = true;

    out = !a;

    if (out)
    {
        printf_s("y = true\n");
    }
    else
    {
        printf_s("y = false\n");
    }

    getchar();
    return 0;
}
```

out becomes equal to not a

C Programming - Booleans

Build and run…

```
int _tmain(int argc, _TCHAR* argv[])
{
    bool a;
    bool out;

    a = true;

    out = !a;

    if (out)
    {
        printf_s("y = true\n");
    }
    else
    {
        printf_s("y = false\n");
    }

    getchar();
    return 0;
```

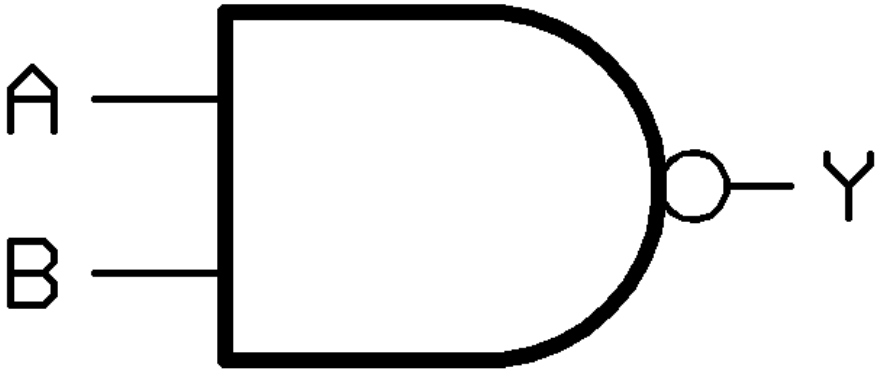


C Programming - Booleans

Modify and run your program to complete the truth table for the NOT gate

A	out
True	?
False	?

In electronics a NAND gate is represented by the following symbol



C Programming - Booleans

Modify your program to emulate the NAND gate…

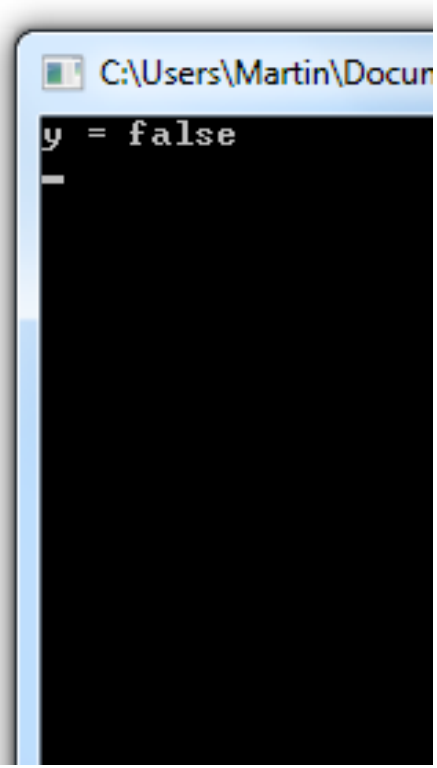
```
int _tmain(int argc, _TCHAR* argv[])
{
    bool a;
    bool b;
    bool y;

    a = true;
    b = true;

    y = !( a && b );

    if (y)
    {
        printf_s("y = true\n");
    }
    else
    {
        printf_s("y = false\n");
    }

    getchar();
    return 0;
}
```

NOT (!) is applied to the result of a and b (a && b)

C Programming - Booleans

Build and run…

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

    a = true;
    b = true;

    y = !( a && b );

    if (y)
    {
        printf_s("y = true\n");
    }
    else
    {
        printf_s("y = false\n");
    }
```

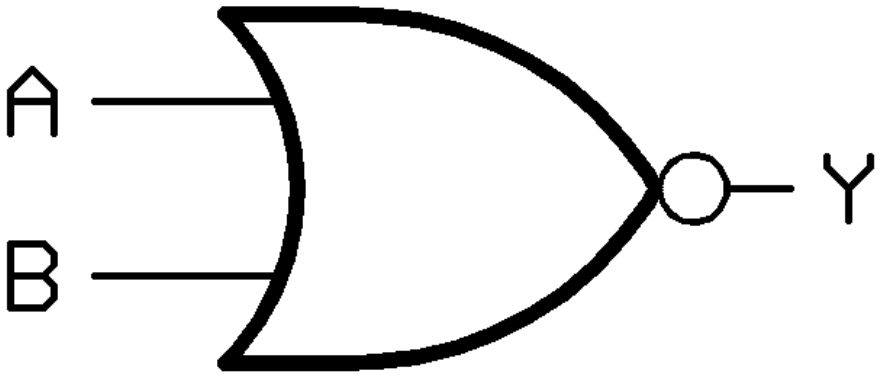


C Programming - Booleans

Modify and run your program to complete the truth table for the NAND gate

A	B	Y
True	True	?
True	False	?
False	True	?
False	False	?

In electronics a NOR gate is represented by the following symbol



C Programming - Booleans

Modify your program to emulate the NOR gate…

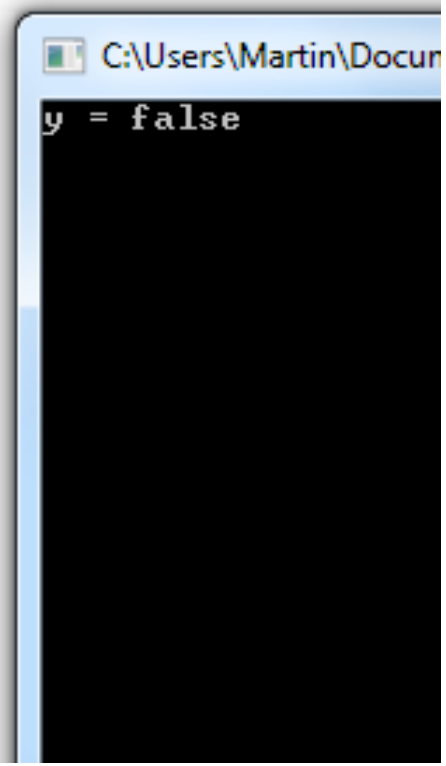
```
int _tmain(int argc, _TCHAR* argv[])
{
    bool a;
    bool b;
    bool y;

    a = true;
    b = true;

    y = !( a || b );

    if (y)
    {
        printf_s("y = true\n");
    }
    else
    {
        printf_s("y = false\n");
    }

    getchar();
    return 0;
}
```

NOT (!) is applied to the result of a or b (a || b)

C Programming - Booleans

Build and run…

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

    a = true;
    b = true;

    y = !( a || b );

    if (y)
    {
        printf_s("y = true\n");
    }
    else
    {
        printf_s("y = false\n");
    }
```



C Programming - Booleans

Modify and run your program to complete the truth table for the NOR gate

A	B	Y
True	True	?
True	False	?
False	True	?
False	False	?

C Programming - Booleans

Note that the NOR program…

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

    a = true;
    b = true;

    y = !( a || b );

    if (y)
    {
        printf_s("y = true\n");
    }
    else
    {
        printf_s("y = false\n");
    }

    getchar();
    return 0;
}
```

C Programming - Booleans

Could be rewritten as…

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

    a = true;
    b = true;

    if ( !( a || b ) )
    {
        printf_s("y = true\n");
    }
    else
    {
        printf_s("y = false\n");
    }

    getchar();
    return 0;
}
```

The End