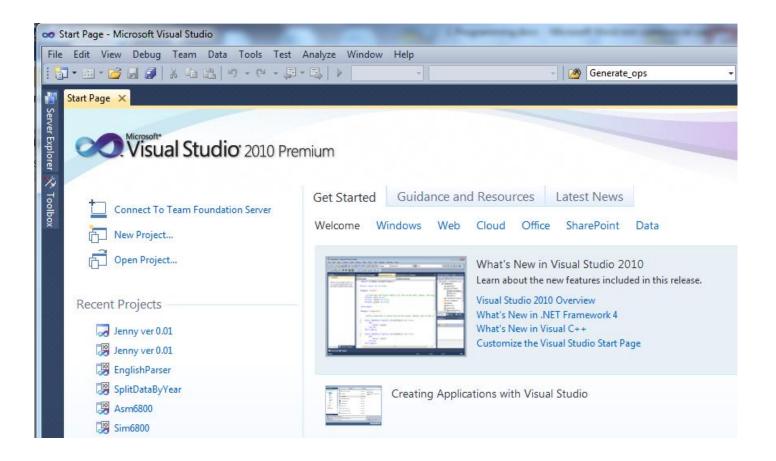
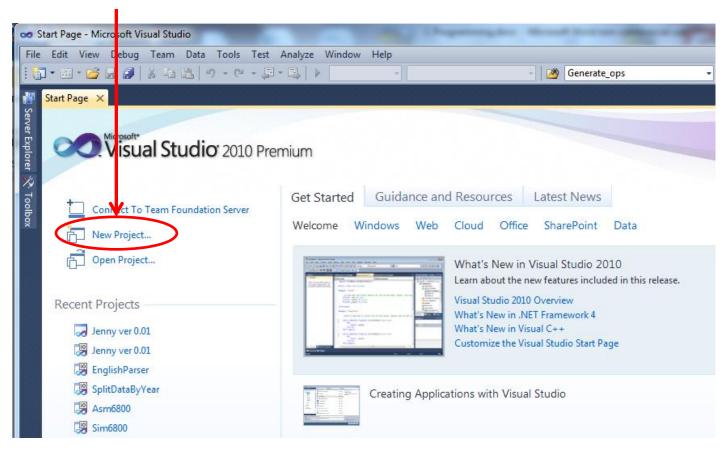
UFCF93-30-1 Computer and Network Systems

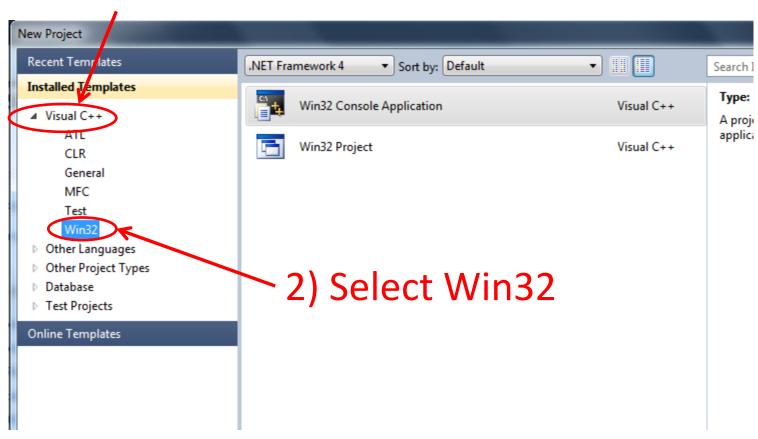
Computer Practical 2 Learning C programming



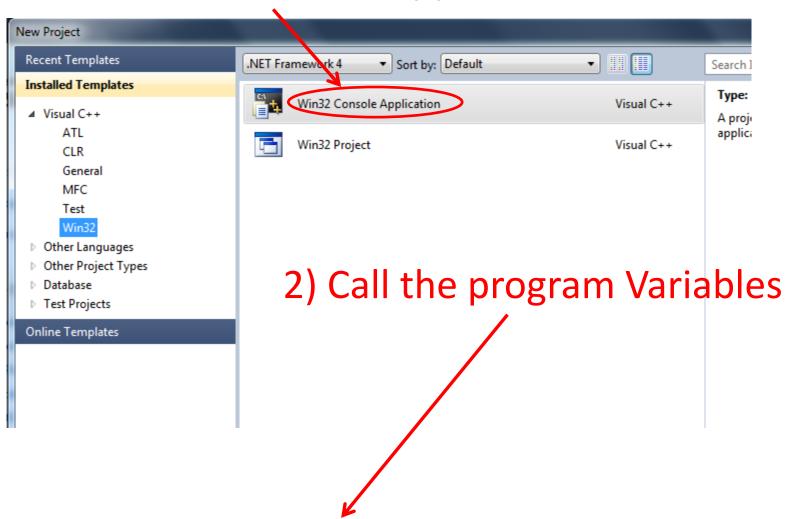
1) Select New Project...

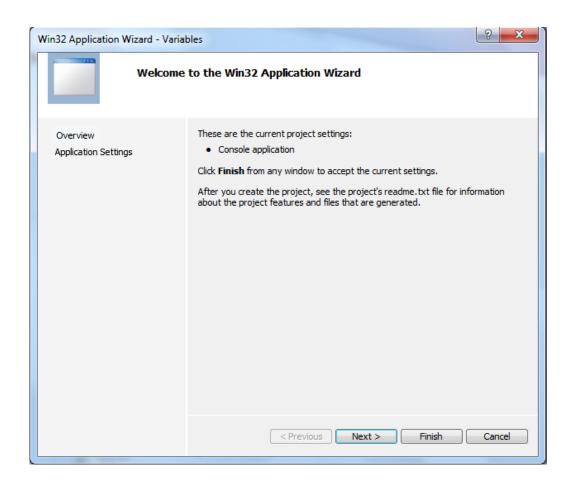


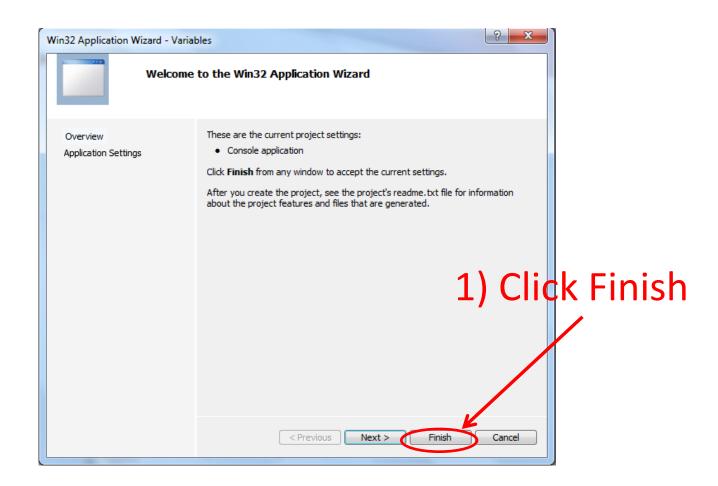
1) Select Visual C++



1) Select Win32 Console Application



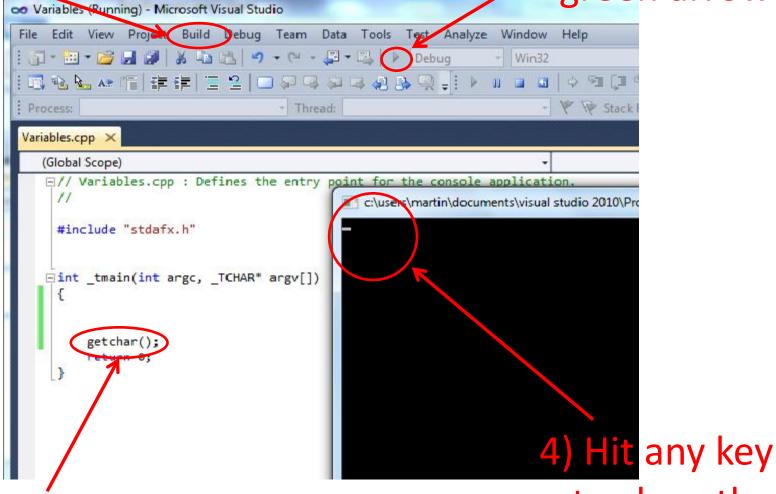




```
Variables - Microsoft Visual Studio
File Edit View Project Build Debug Team Data Tools Test Analyze Window H
  🛅 🕶 🛅 🗲 📴 🔛 🦸 🐰 🛅 🖺 🕑 🗸 🖰 🗸 🗗 Debug
                                                          ▼ Win32
    Variables.cpp X
      (Global Scope)
      □// Variables.cpp : Defines the entry point for the console applicat:
        #include "stdafx.h"
                                                             Your new
      □int _tmain(int argc, _TCHAR* argv[])
                                                             program
           return 0;
```

2) Click Build

_, 3) Click the green arrow



1) Add getchar();

to close the program

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**

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 !)

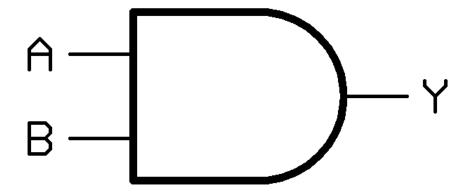
How are boolean variables stored?

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



Lets implement this in a C program...

```
☐ int tmain(int argc, TCHAR* argv[])

                  bool a;
                               Reserves memory
                  bool b;
Add this
                  bool y;
code to
                               Initialises a and b
the
program
                  printf_s("y) = %d\n", y);
                  getchar();
                  return 0;
                               Logical AND (note: a single & is
                               used to denote a bitwise AND)
```

Build and run...

```
□int tmain(int argc, _TCHAR* argv[])
     bool a;
                                                     Remember in
     bool b;
                               C:\Users\Martin\D
     bool y;
                                                     Visual Studio
     a = true;
                                                     false is 0 and
     b = true;
                                                   true is 1 (it will
     y = a & b;
                                                       be different
     printf s("y = %d\n", y);
                                                        with other
     getchar();
                                                         compilers)
     return 0;
```

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

```
⊡int tmain(int argc, TCHAR* argv[])
     bool a;
     bool b;
     bool y;
                                     If y is true then print
     a = true;
     b = true;
                                                      "y = true"
    y = a & b;
     if (y)
        printf s("y = true\n");
    else
        printf s("y = false\n");
                                         If y is false then print
     getchar();
                                                          "y = false"
     return 0;
```

Change the print statement to this

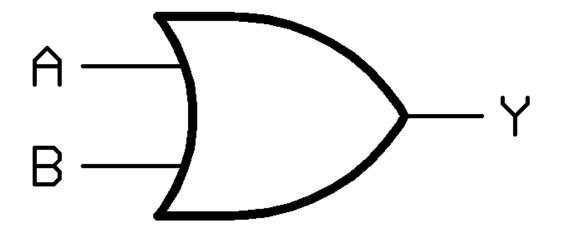
Build and run...

```
□int _tmain(int argc, _TCHAR* argv[])
                                                                     Now it says
     bool a;
                                        C:\Users\Martin\Docume
                                                                          y is true
     bool b;
                                         true
     bool y;
     a = true;
     b = true;
     y = a & b;
     if (y)
         printf_s("y = true\n");
     else
         printf_s("y = false\n");
```

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

Α	В	Υ
True	True	?
True	False	?
False	True	Ś
False	False	Ş

In electronics an OR gate is represented by the following symbol



Modify your program to emulate the OR gate...

```
□int tmain(int argc, TCHAR* argv[])
    bool a;
    bool b;
                                   Logical OR (note: a single | is
    bool y;
                                   used to denote a bitwise OR)
    a = true;
    b = true;
    y = a(||)b;
    if(y)
        printf_s("y = true\n");
    else
        printf s("y = false\n");
    getchar();
    return 0;
```

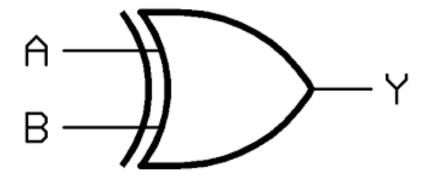
Build and run...

```
int _tmain(int argc, _TCHAR* argv[])
      bool a;
      bool b;
                                         C:\Users\Martin
      bool y;
                                             true
     a = true;
     b = true;
     y = a \mid\mid b;
      if (y)
         printf_s("y = true\n");
      else
         printf_s("y = false\n");
```

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

Α	В	Y
True	True	Ş
True	False	?
False	True	Ś
False	False	Ś

In electronics an XOR gate is represented by the following symbol



Modify your program to emulate the XOR gate...

```
int tmain(int argc, _TCHAR* argv[])
    bool a;
    bool b;
                                    Bitwise XOR (note: a logical ^^
    bool y;
    a = true;
                                                                  is not valid)
    b = true;
    if (y)
        printf s("y = true\n");
    else
        printf s("y = false\n");
    getchar();
    return 0;
```

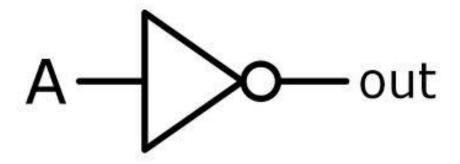
Build and run...

```
∃int _tmain(int argc, _TCHAR* argv[])
     bool a;
                                        C:\Users\Martin\Documents'
     bool b;
     bool y;
                                          = false
     a = true;
     b = true;
     y = a ^ b;
     if (y)
         printf_s("y = true\n");
     else
         printf_s("y = false\n");
```

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

Α	В	Υ
True	True	Ş
True	False	?
False	True	Ś
False	False	Ś

In electronics an NOT gate is represented by the following symbol



Lets implement this in a C program...

```
□int tmain(int argc, TCHAR* argv[])
                   bool a;
                   bool out;
Modify
                   a = true;
                                             out becomes equal to
this
                   out = !a;
                                             not a
code
                   if (out)
                       printf s("y = true\n");
                   else
                       printf s("y = false\n");
                   getchar();
                   return 0;
```

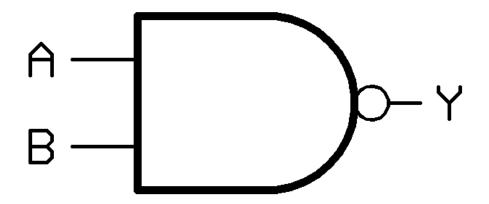
Build and run...

```
∃int _tmain(int argc, _TCHAR* argv[])
     bool a;
     bool out;
                                        C:\Users\Martin\Docu
     a = true;
                                          = false
     out = !a;
     if (out)
         printf_s("y = true\n");
     else
         printf_s("y = false\n");
     getchar();
     return 0;
```

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

Α	out
True	?
False	?

In electronics a NAND gate is represented by the following symbol



Modify your program to emulate the NAND gate...

```
∃int tmain(int argc, TCHAR* argv[])
    bool a;
    bool b;
    bool y;
                                      NOT (!) is applied to the
    a = true;
    b = true;
                                    result of a and b (a && b)
   y = !( a && b );
    if (y)
        printf s("y = true\n");
    else
        printf s("y = false\n");
    getchar();
    return 0;
```

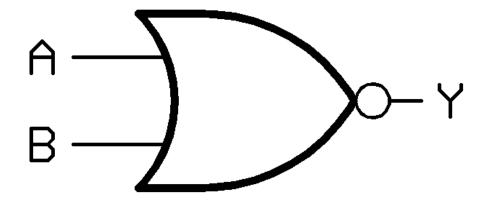
Build and run...

```
□int _tmain(int argc, _TCHAR* argv[])
     bool a;
     bool b;
                                         C:\Users\Martin\Docun
     bool y;
                                           = false
     a = true;
     b = true;
     y = !(a \&\& b);
     if (y)
         printf_s("y = true\n");
     else
        printf_s("y = false\n");
```

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

Α	В	Y
True	True	Ş
True	False	?
False	True	Ś
False	False	Ś

In electronics a NOR gate is represented by the following symbol



Modify your program to emulate the NOR gate...

```
∃int tmain(int argc, TCHAR* argv[])
    bool a;
    bool b;
    bool y;
                                      NOT (!) is applied to the
    a = true;
    b = true;
                                        result of a or b (a | b)
    y = !(a || b);
    if (y)
        printf s("y = true\n");
    else
        printf_s("y = false\n");
    getchar();
    return 0;
```

Build and run...

```
int _tmain(int argc, _TCHAR* argv[])
     bool a;
     bool b;
                                            C:\Users\Martin\Docun
     bool y;
                                         y = false
     a = true;
     b = true;
     y = !( a || b );
     if (y)
         printf_s("y = true\n");
     else
         printf_s("y = false\n");
```

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

Α	В	Y
True	True	Ş
True	False	?
False	True	Ś
False	False	Ś

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;
```

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