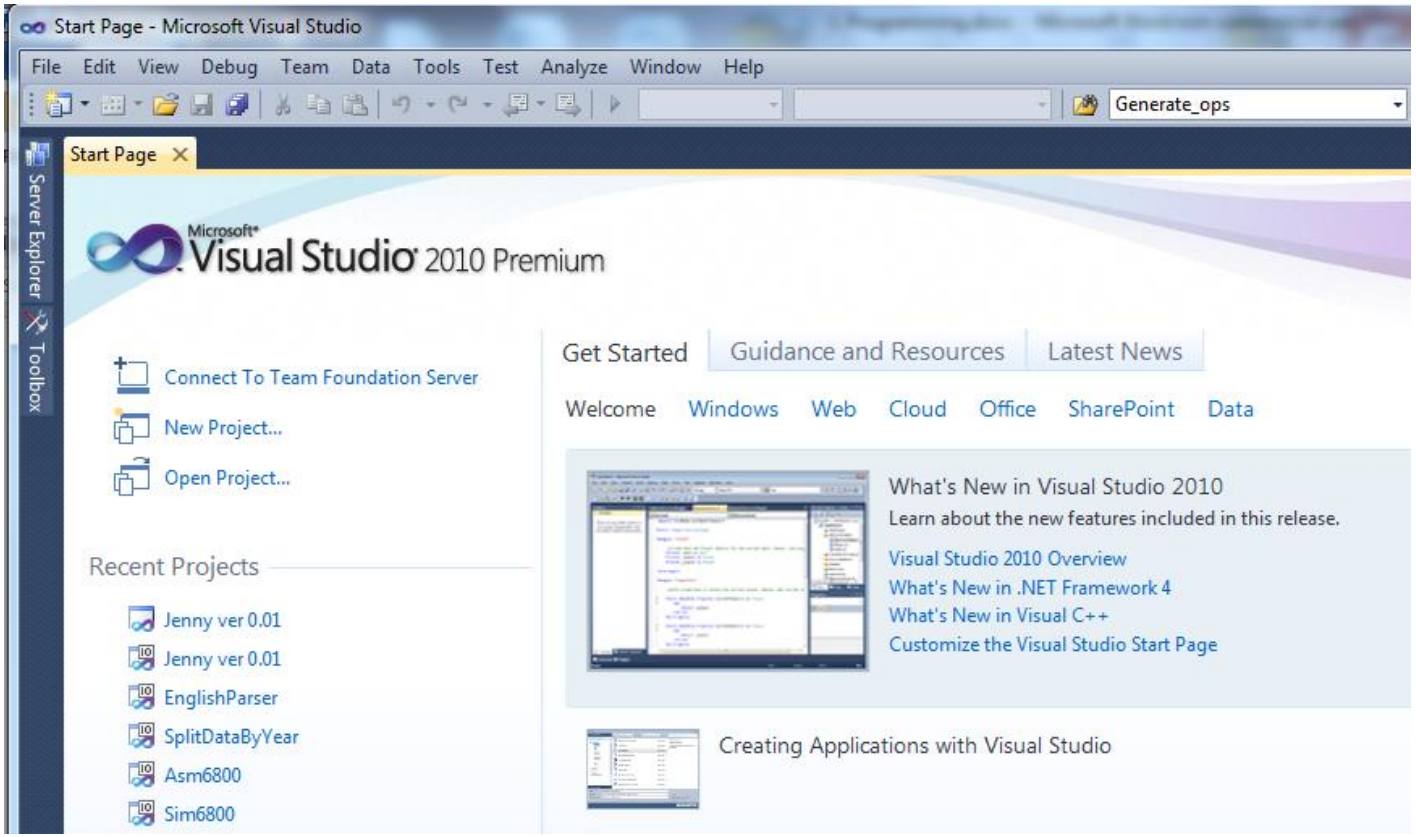
UFCF93-30-1 Computer and Network Systems

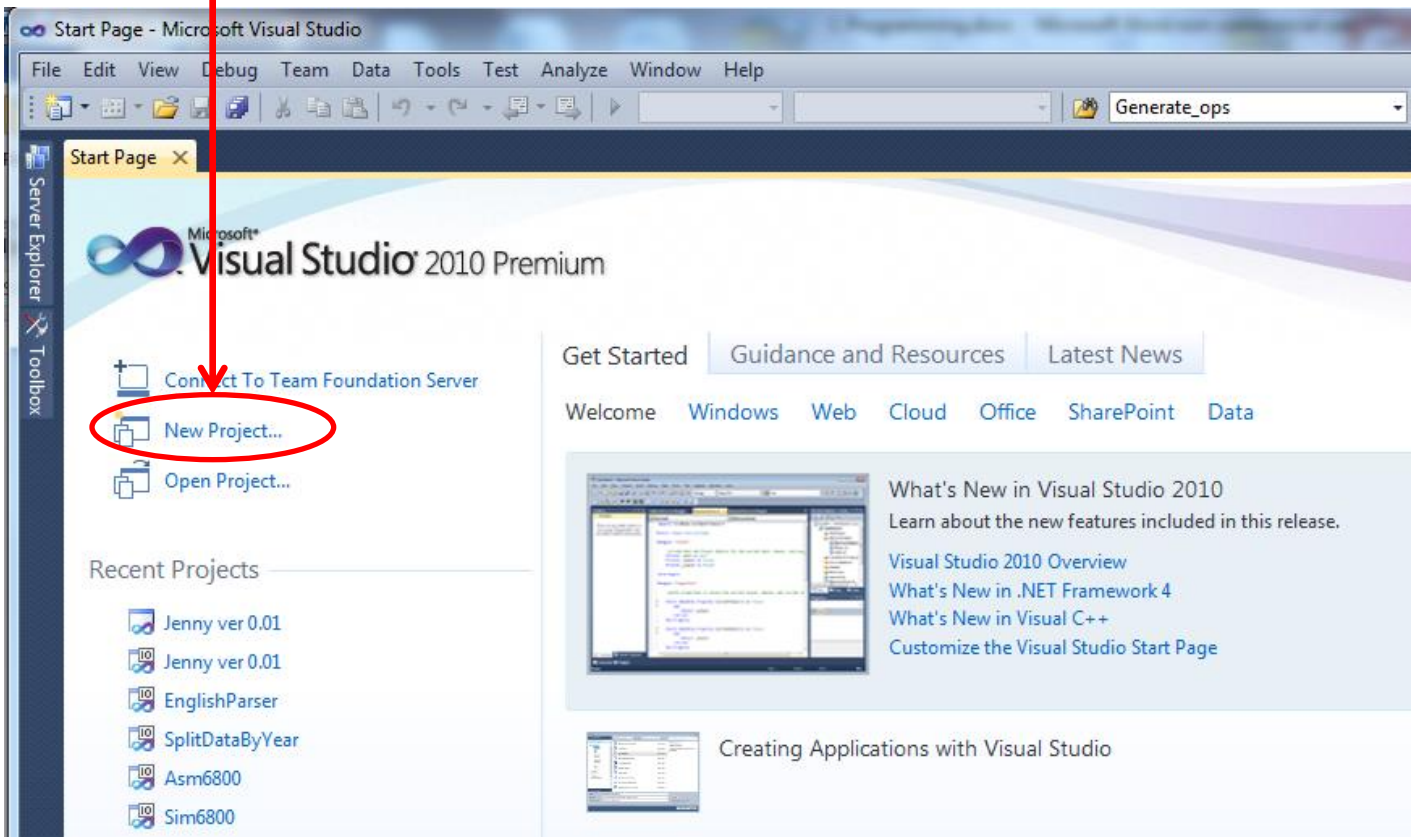
Computer Practical 3 Learning C programming

C Programming



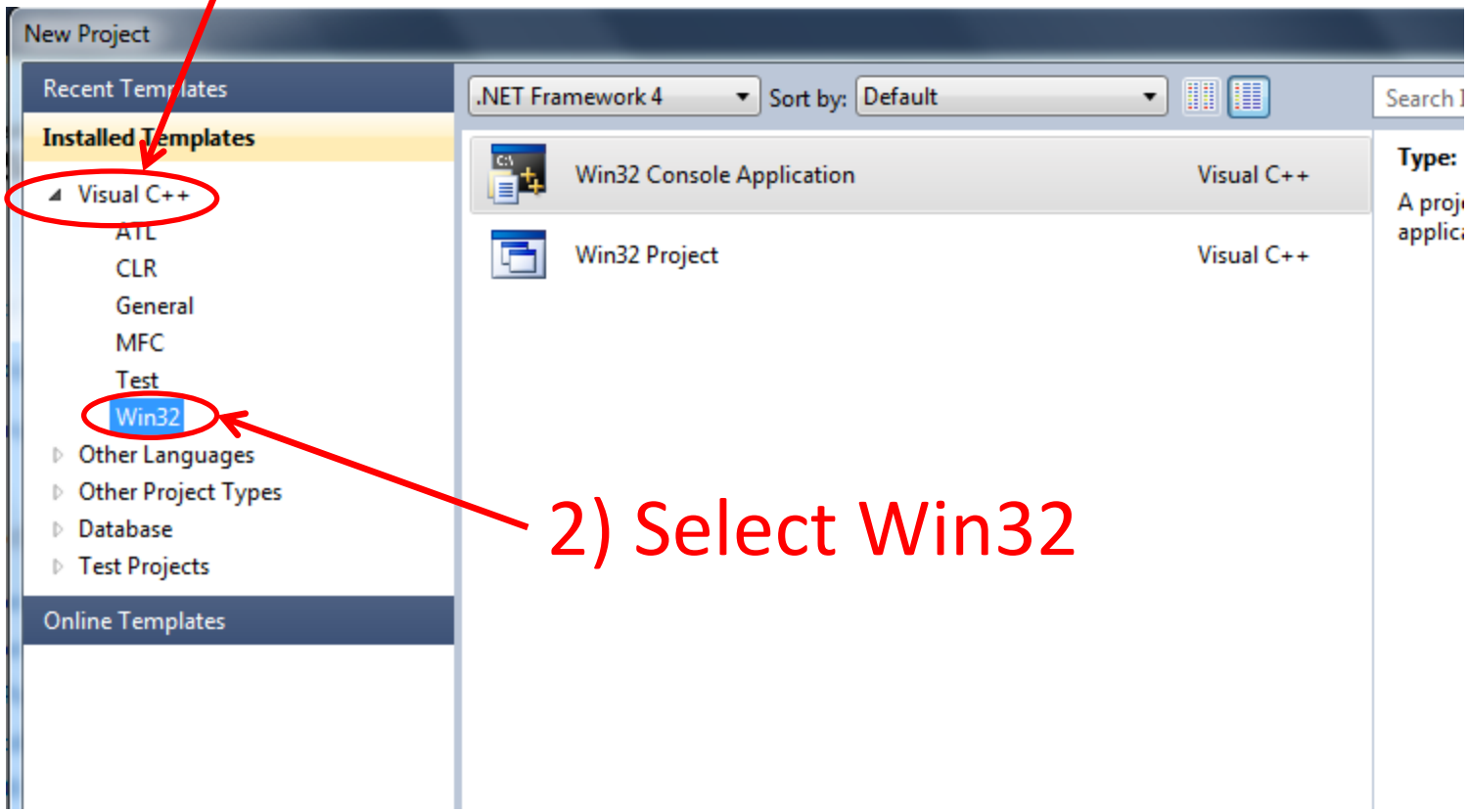
C Programming

1) Select New Project…



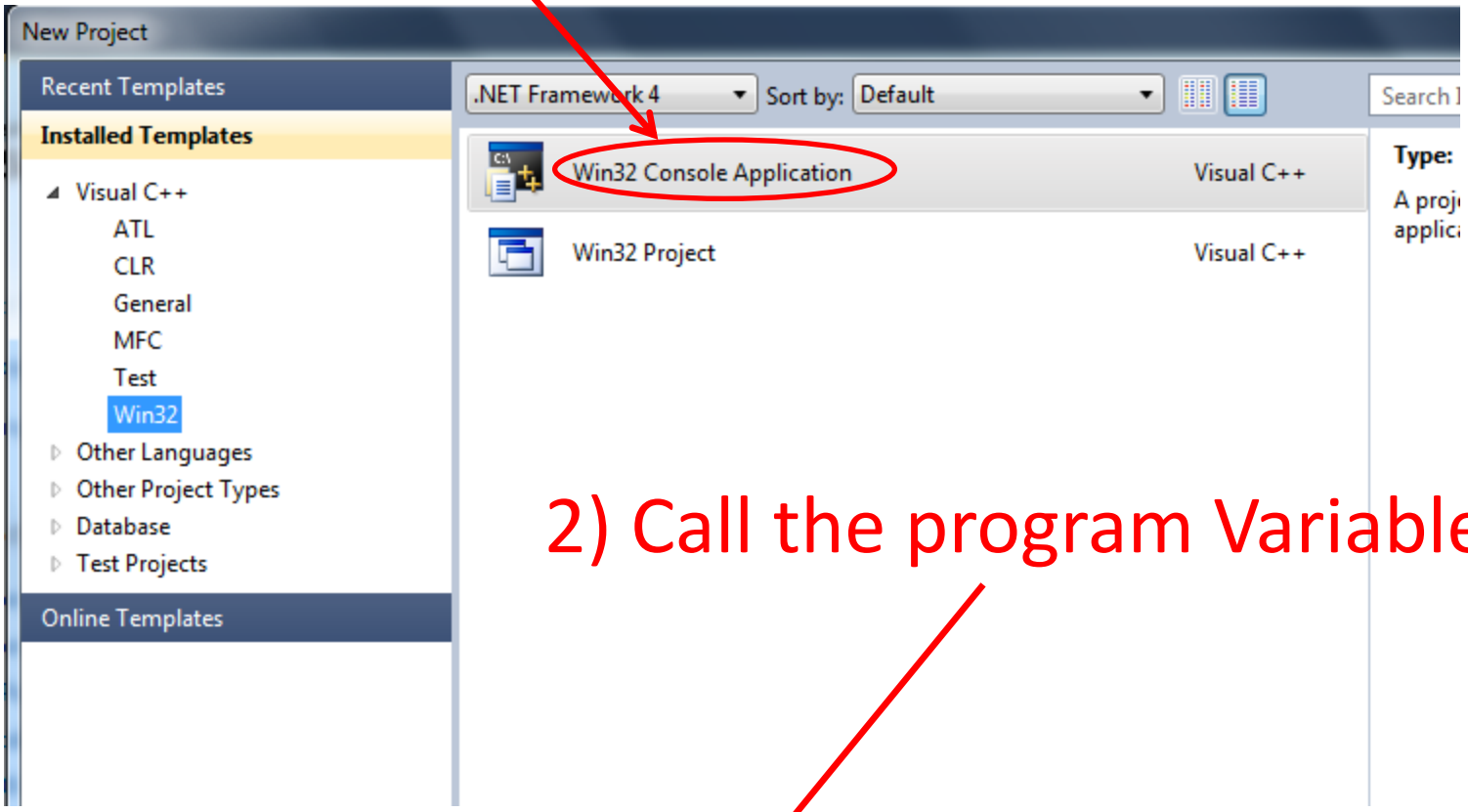
C Programming

1) Select Visual C++



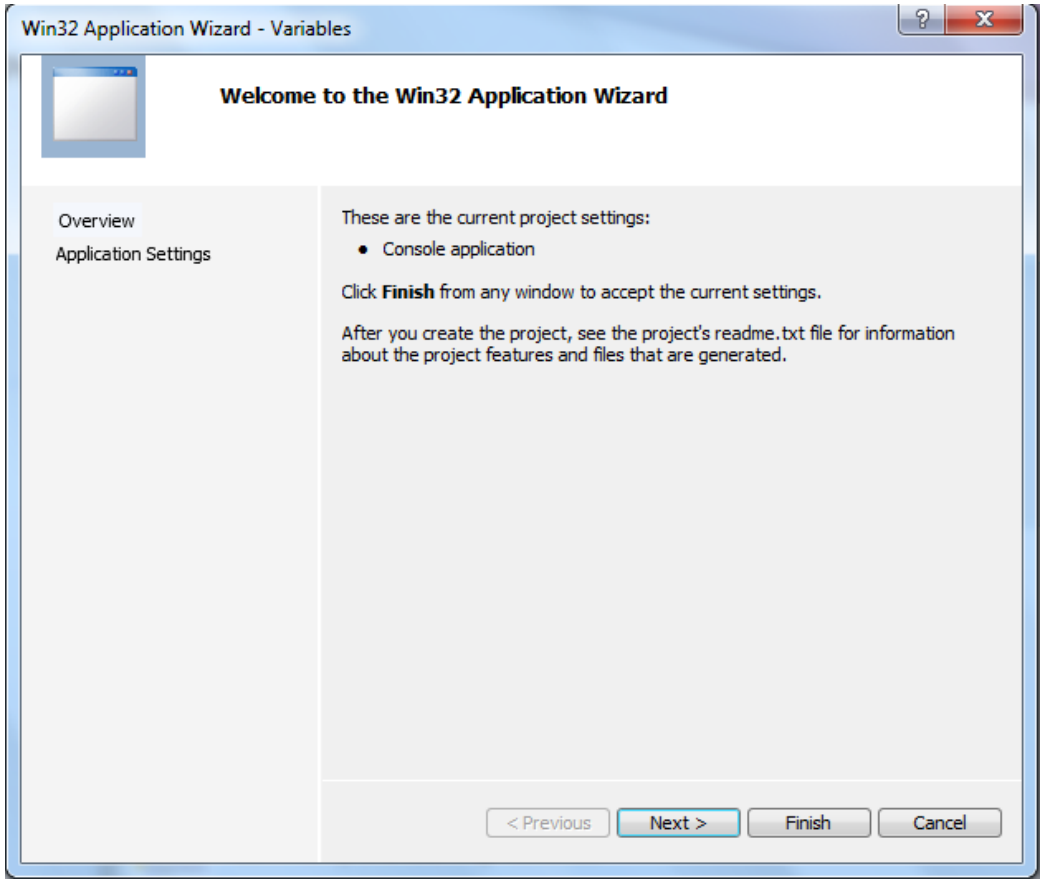
C Programming

1) Select Win32 Console Application

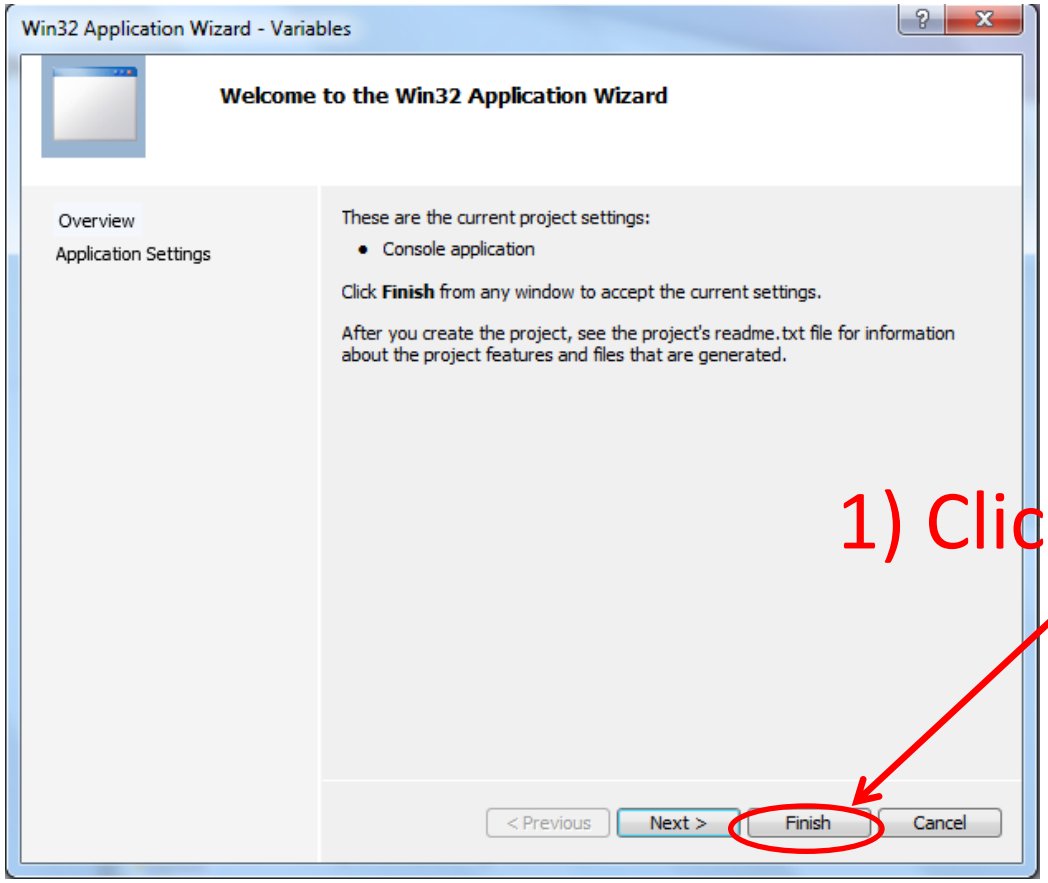


2) Call the program Variables

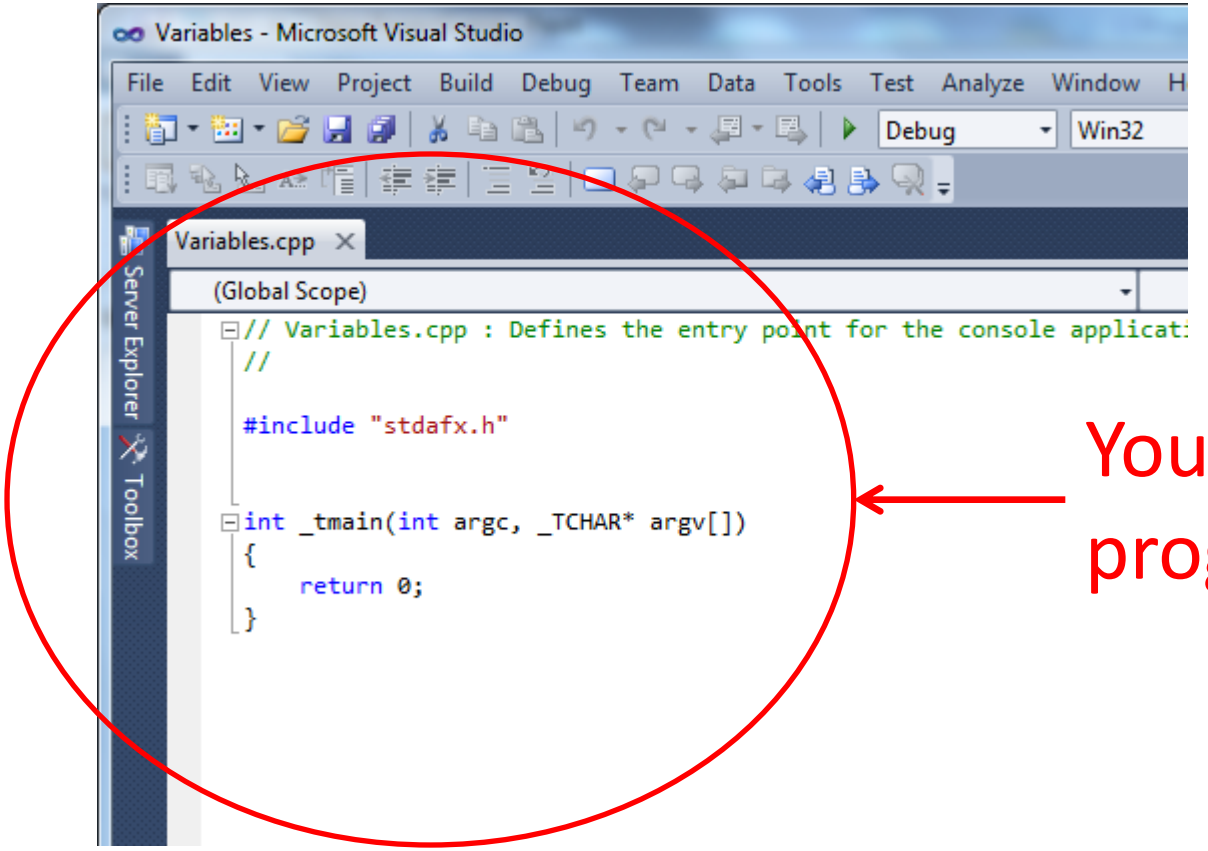
C Programming



C Programming



C Programming

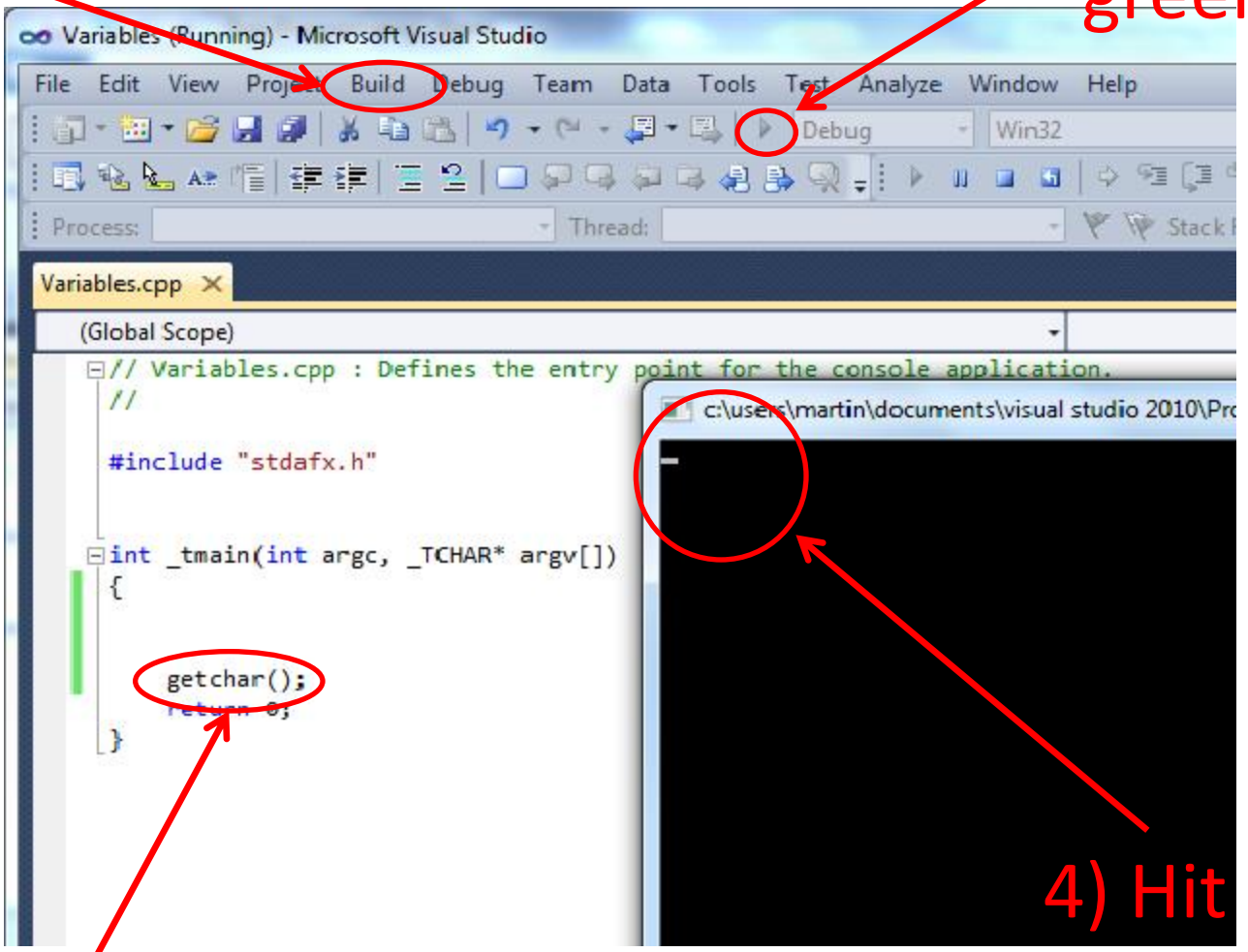


Your new program

C Programming

2) Click Build

3) Click the green arrow

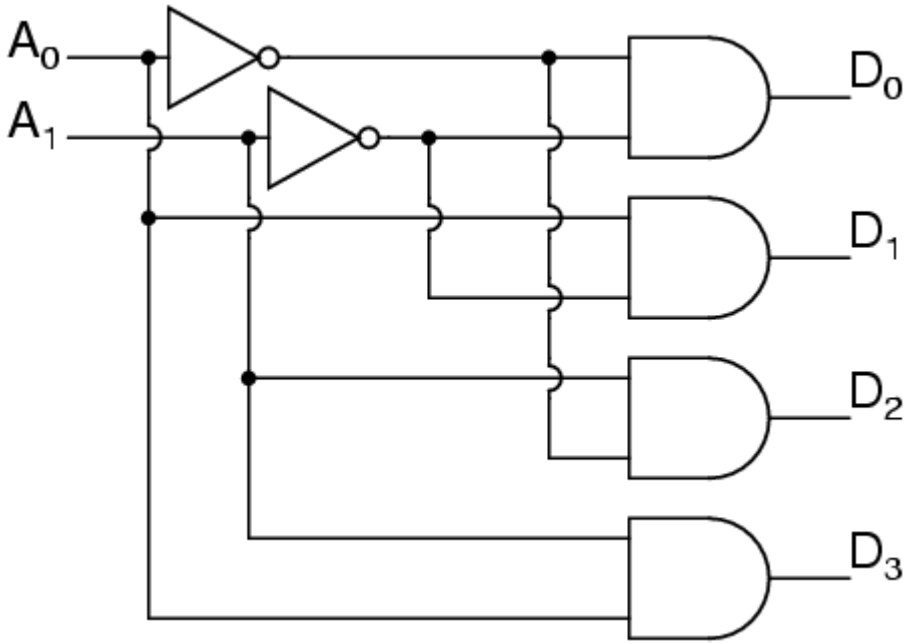


1) Add getchar();

4) Hit any key to close the program

C Programming – Decoding Circuits

In this decoding circuit a 2 bit input (A_0 and A_1) activates one of 4 outputs



Picture source: Internet

C Programming – Decoding Circuits

Lets code this into a C program…

```
int _tmain(int argc, _TCHAR* argv[])
{
    bool a0, a1;
    bool d0, d1, d2, d3;

    a0 = true;
    a1 = true;

    d0 = !a0 && !a1;
    d1 =  a0 && !a1;
    d2 = !a0 &&  a1;
    d3 =  a0 &&  a1;

    if (d0)
    {
        printf_s("y = true\n");
    }
    else
    {
        printf_s("y = false\n");
    }

    getchar();
    return 0;
}
```

Allocating memory space

Initialising input variables

Decode logic

We would need to repeat this four times to output D0 to D3. Repeated code like this we put into a function

C Programming – Decoding Circuits

Lets have a function print our true and false for us…

Add this function above main()

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

int _tmain(int argc, _TCHAR* argv[])
{
    bool a0, a1;
    bool d0, d1, d2, d3;
```

C Programming – Decoding Circuits

Our new function is of type void as it doesn't return a value

Add this function above main()

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

int _tmain(int argc, _TCHAR* argv[])
{
    bool a0, a1;
    bool d0, d1, d2, d3;
```

Our new function has two parameters; the first a char and the second a boolean

%c means print out a character

C Programming – Decoding Circuits

Now we can update main() to use our new function…

```
int _tmain(int argc, _TCHAR* argv[])
{
    bool a0, a1;
    bool d0, d1, d2, d3;

    a0 = true;
    a1 = true;

    d0 = !a0 && !a1;
    d1 =  a0 && !a1;
    d2 = !a0 &&  a1;
    d3 =  a0 &&  a1;

    print('0', d0);
    print('1', d1);
    print('2', d2);
    print('3', d3);

    getchar();
    return 0;
}
```

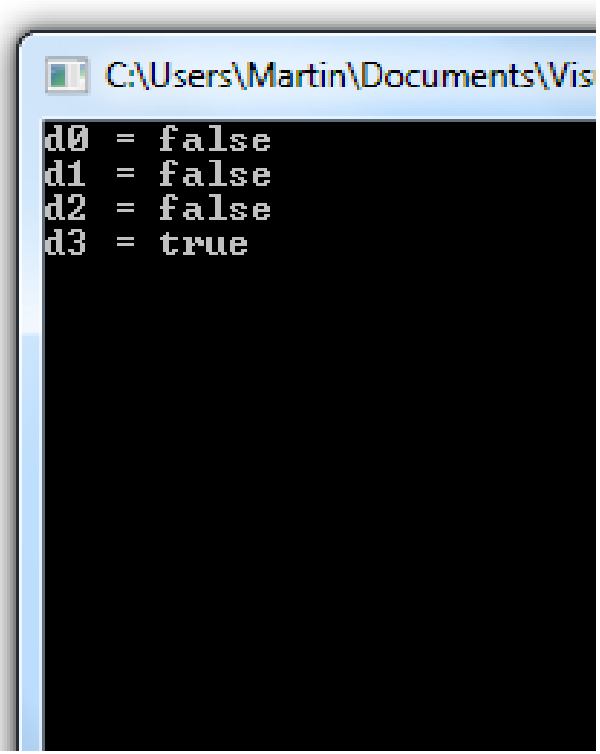
The first parameter is a character that will distinguish which of d0 to d3 we are outputting. Single characters are always in single quotes in C.

The second parameter is the value of d0 to d3

C Programming – Decoding Circuits

Build and run…

```
int _tmain(int argc, _TCHAR* argv[])
{
    bool a0, a1;
    bool d0, d1, d2, d3;

    a0 = true;
    a1 = true;

    d0 = !a0 && !a1;
    d1 =  a0 && !a1;
    d2 = !a0 &&  a1;
    d3 =  a0 &&  a1;

    print('0', d0);
    print('1', d1);
    print('2', d2);
    print('3', d3);
```



C Programming – Decoding Circuits

Modify and run your program to complete this table…

A0	A1	D0	D1	D2	D3
False	False	?	?	?	?
False	True	?	?	?	?
True	False	?	?	?	?
True	True	?	?	?	?

C Programming – Decoding Circuits

In programming you can decode using if statements…

```
if (a1)
{
    if (a0)
    {
    }
    else
    {
    }
}
else
{
    if (a0)
    {
    }
    else
    {
    }
}
```

Code for a1 == true and a0 == true goes in here

Code for a1 == true and a0 == false goes in here

Code for a1 == false and a0 == true goes in here

Code for a1 == false and a0 == false goes in here

C Programming – Decoding Circuits

In programming you can decode using a case statement (combine a1 and a0 into one integer)…

```
switch (a)
{
case 3:

    break;

case 2:

    break;

case 1:

    break;

case 0:

    break;

default:

    break;
}
```

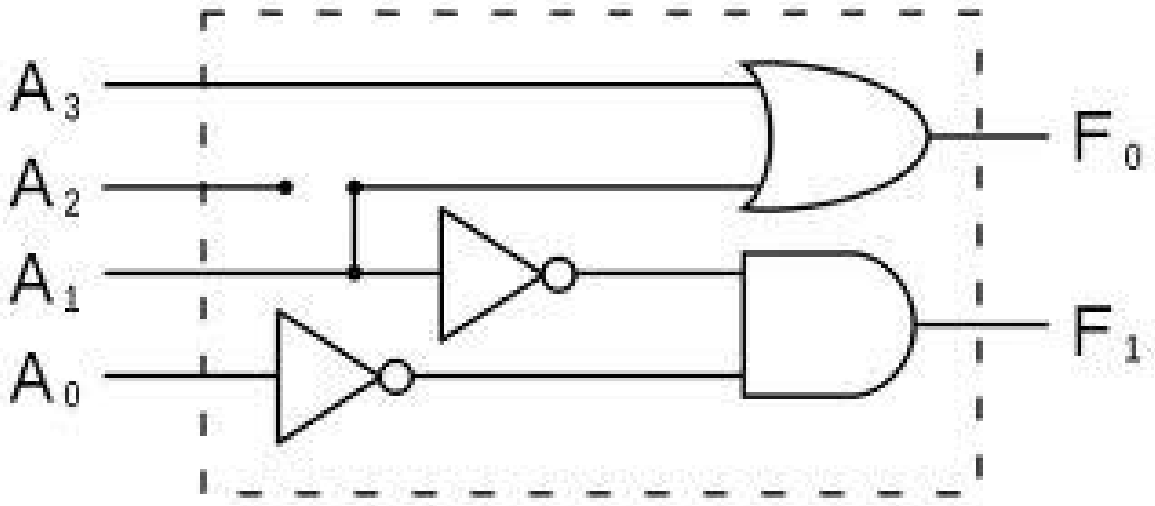
Code for a1 == true and a0 == true goes in here

Code for a1 == true and a0 == false goes in here

Code for a1 == false and a0 == true goes in here

Code for a1 == false and a0 == false goes in here

Encoding circuits work the other way around to decoding circuits



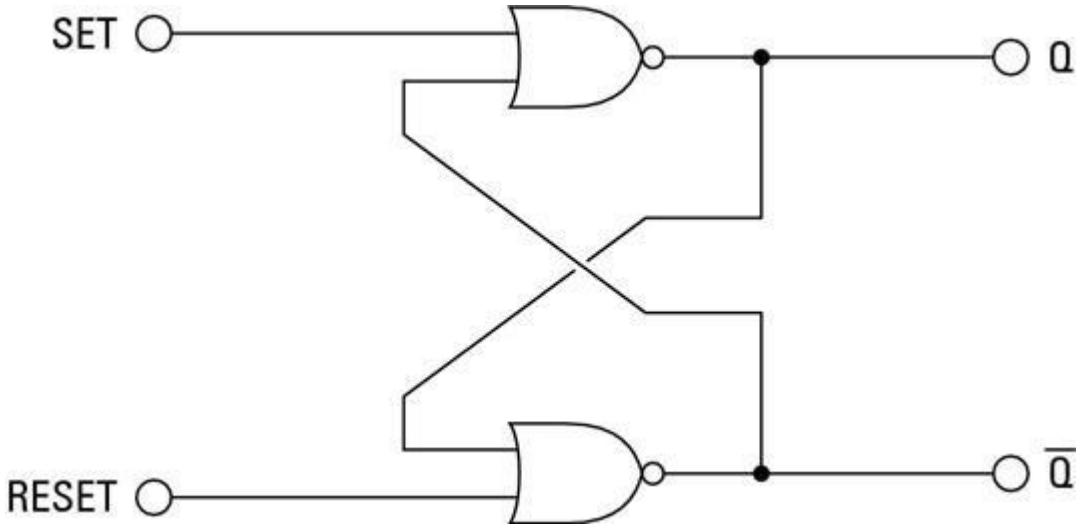
Picture source: Internet

The encoder circuit would behave like this…

Inputs	Outputs
0001	00
0010	01
0100	10
1000	11

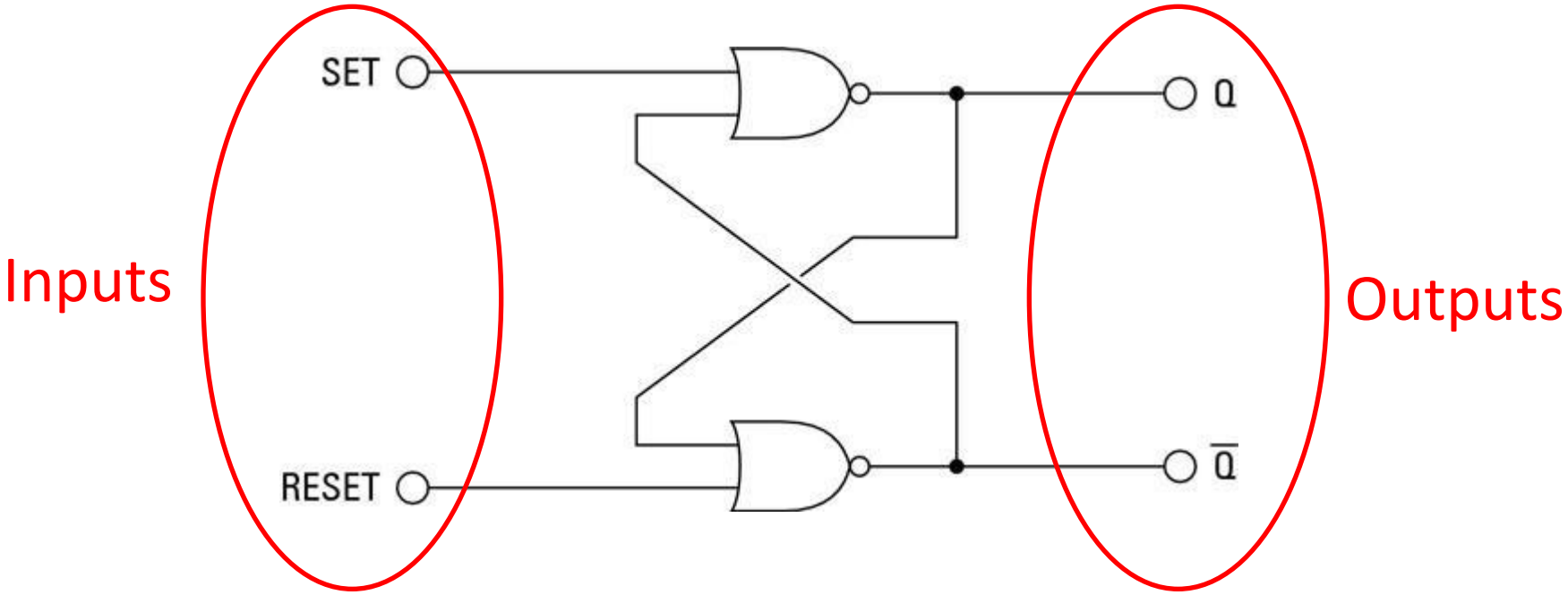
C Programming – memory circuits

We are going to look at a simple **memory circuit**… a **basic latch circuit** made from **NOR** gates



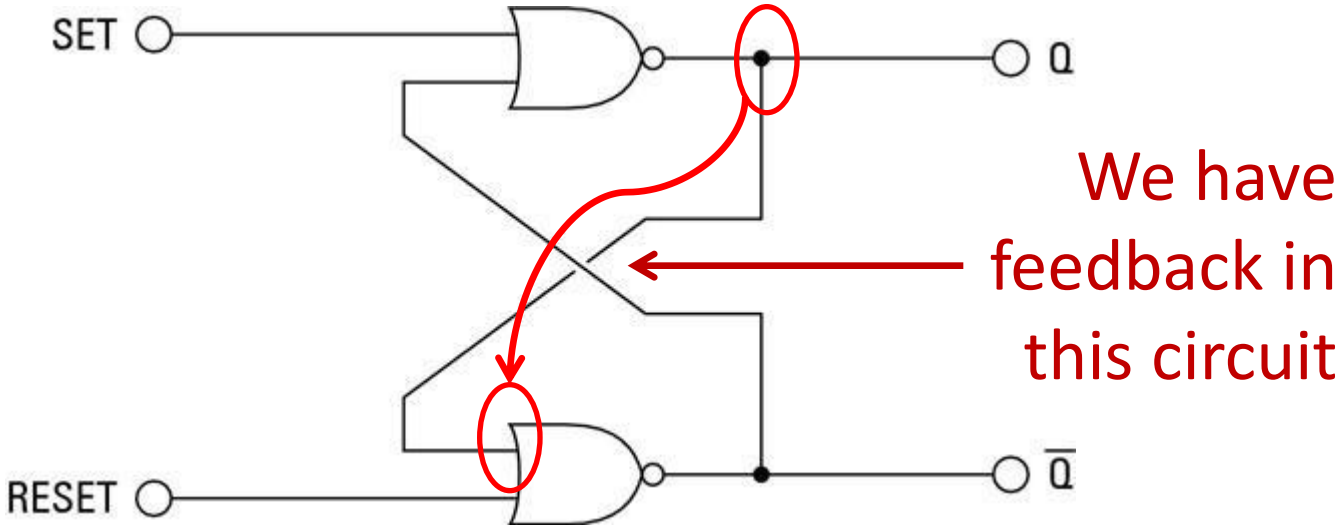
Picture source: Internet

We are going to look at a simple **memory circuit**… a **basic latch circuit** made from **NOR** gates



Picture source: Internet

We are going to look at a simple **memory circuit**… a **basic latch circuit** made from **NOR** gates



Picture source: Internet

C Programming – memory circuits

Lets write some C code to emulate this…

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

```
int _tmain(int argc, _TCHAR* argv[])
{
    bool set;
    bool reset;
    bool q;
    bool q_bar;

    q     = false;  // Set to their initial values
    q_bar = false;

    set   = true;
    reset = true;

    print ("q    ", q);
    print ("q_bar", q_bar);

    getchar();
    return 0;
}
```

C Programming – memory circuits

char* is a pointer to a variable of type char and in the C programming language this is how we handle strings of text

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

%s means print out a string

name is a pointer to a string of characters

```
int _tmain(int argc, _TCHAR* argv[])
{
    bool set;
    bool reset;
    bool q;
    bool q_bar;

    q     = false;  // Set to their initial values
    q_bar = false;

    set   = true;
    reset = true;

    print ("q    ", q);
    print ("q_bar", q_bar);

    getchar();
    return 0;
}
```

In C a string is an array of characters. The string is always in double quotes

C Programming – memory circuits

Strings…

An array of char is a string in C

Initialise the string; the string contains chars CaNS and '\0'

```
char text_array[10] = {"CaNS"};
char* name;

name = &text_array[0];
```

Set the pointer name to the address of the first char in the array; & means address of

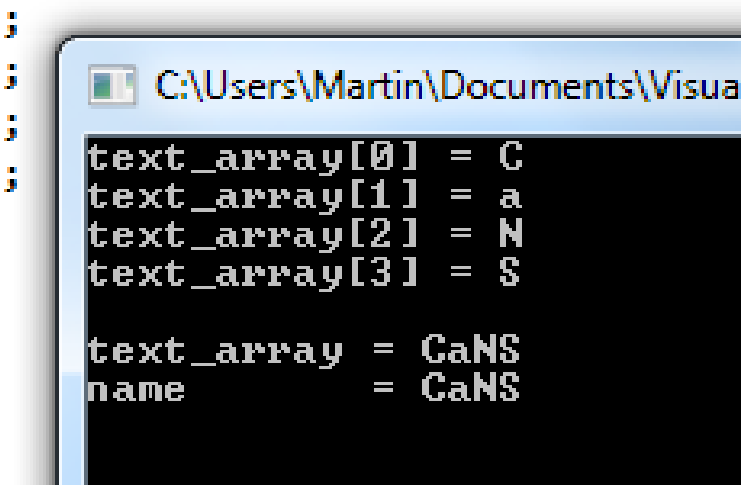
```
printf("text_array[0] = %c\n", text_array[0]);
printf("text_array[1] = %c\n", text_array[1]);
printf("text_array[2] = %c\n", text_array[2]);
printf("text_array[3] = %c\n", text_array[3]);
printf("\n");
printf("text_array = %s\n", text_array);
printf("name       = %s\n", name);
printf("\n");
printf("\n");
```

Print individual characters

Print strings

C Programming – memory circuits

```
char  text_array[10] = {"CaNS"};
char* name;

name = &text_array[0];

printf("text_array[0] = %c\n", text_array[0]);
printf("text_array[1] = %c\n", text_array[1]);
printf("text_array[2] = %c\n", text_array[2]);
printf("text_array[3] = %c\n", text_array[3]);
printf("\n");
printf("text_array = %s\n", text_array);
printf("name       = %s\n", name);
printf("\n");
printf("\n");
```



C Programming – memory circuits

Back to our basic latch, add the following code…

1) Move q and q_bar to the top of the program, it makes them **global** variables

```
#include "stdafx.h"

bool q;
bool q_bar;
```

2) Add a latch function below the print function and above main()

```
void latch(bool set, bool reset)
{
    q =     !( set   || q_bar );
    q_bar = !( reset || q );

    print ("q    ", q);
    print ("q_bar", q_bar);
    printf_s("\n");
}
```

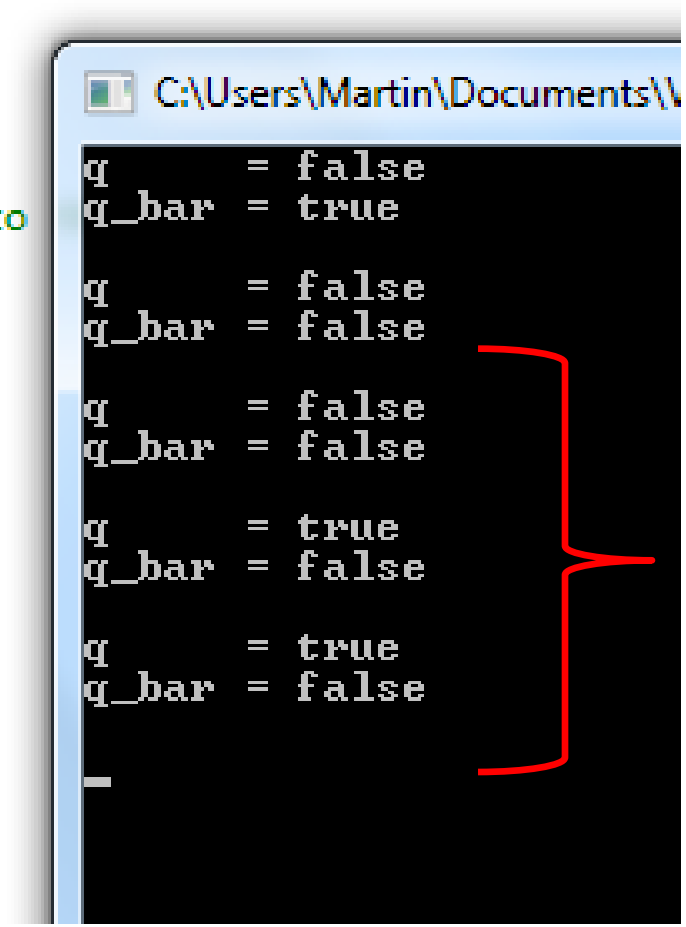
C Programming – memory circuits

Re-write main()…

```
int _tmain(int argc, _TCHAR* argv[])
{
    bool set;
    bool reset;

    q     = false;  // Set to their initial values
    q_bar = true;

    //    set    reset
    latch(false, false);
    latch(false, true );
    latch(true,  true );
    latch(false, true );
    latch(false, false);

    getchar();
    return 0;
}
```

C Programming – memory circuits

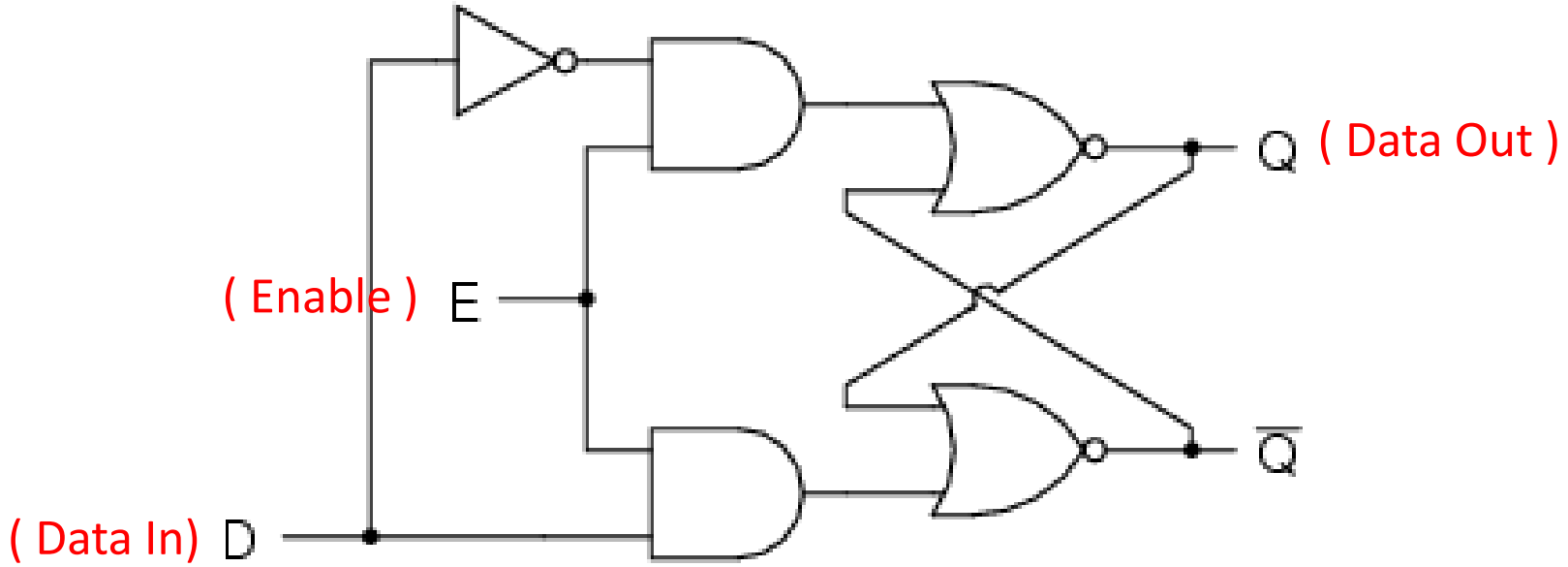
```
int _tmain(int argc, _TCHAR* argv[])
{
    bool set;
    bool reset;

    q     = false;  // Set to
    q_bar = true;

    //    set    reset
    latch(false, false);
    latch(false, true );
    latch(true,  true );
    latch(false, true );
    latch(false, false);

    getchar();
    return 0;
}
```



When set is toggled while reset is true the value of reset is remembered

C Programming – memory circuits

Latch circuits are not very easy to use so we add some more logic to create a create a **memory circuit**…



Picture source: Internet

C Programming – memory circuits

Update the program…

```
void memory(bool data_in, bool enable)
{
    bool set;
    bool reset;

    set   = (!data_in) && enable;
    reset =   data_in  && enable;

    q =     !( set   || q_bar );
    q_bar = !( reset || q );

    print ("data_out", q);
    printf_s("\n");
}
```

```
int _tmain(int argc, _TCHAR* argv[])
{
    bool set;
    bool reset;

    q     = false;  // Set to their initial values
    q_bar = true;

    //     data_in enable
    memory(false,  false);
    memory(true,   false);
    memory(true,   true);
    memory(true,   false);
    memory(false,  false);
    memory(false,  true);
    memory(false,  false);
    memory(false,  false);

    getchar();
    return 0;
}
```

C Programming – memory circuits

'Pass by value', the value of the variable is passed

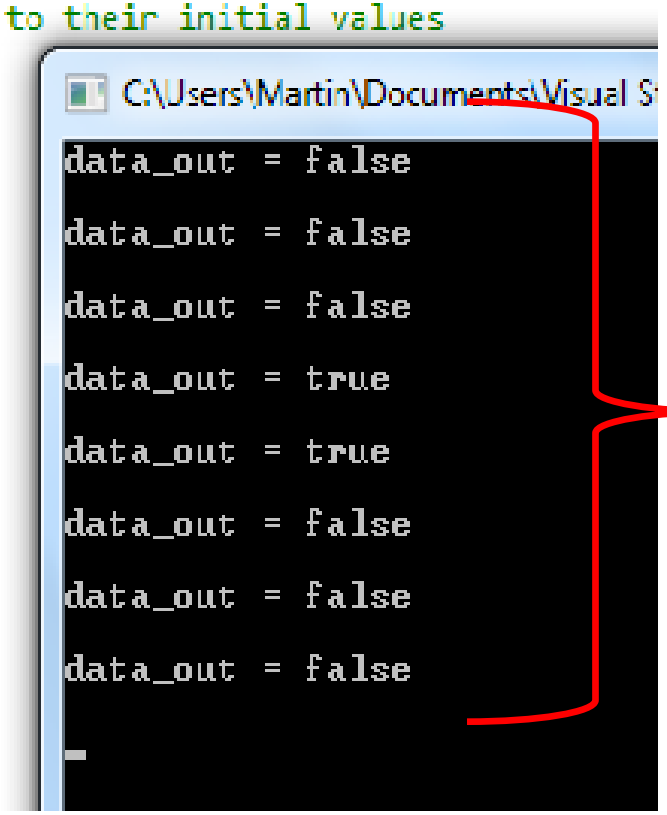
'Pass by reference', the address of the variable is passed

```
void memory(bool data_in, bool enable)
{
    bool set;
    bool reset;

    set   = (!data_in) && enable;
    reset =   data_in  && enable;

    q =     !( set   || q_bar );
    q_bar = !( reset || q );

    print ("data_out", q);
    printf_s("\n");
}
```

```
int _tmain(int argc, _TCHAR* argv[])
{
    bool set;
    bool reset;

    q     = false;  // Set to their initial values
    q_bar = true;

    //     data_in enable
    memory(false,  false);
    memory(true,   false);
    memory(true,   true);
    memory(true,   false);
    memory(false,  false);
    memory(false,  true);
    memory(false,  false);
    memory(false,  false);

    getchar();
    return 0;
}
```

C Programming – memory circuits

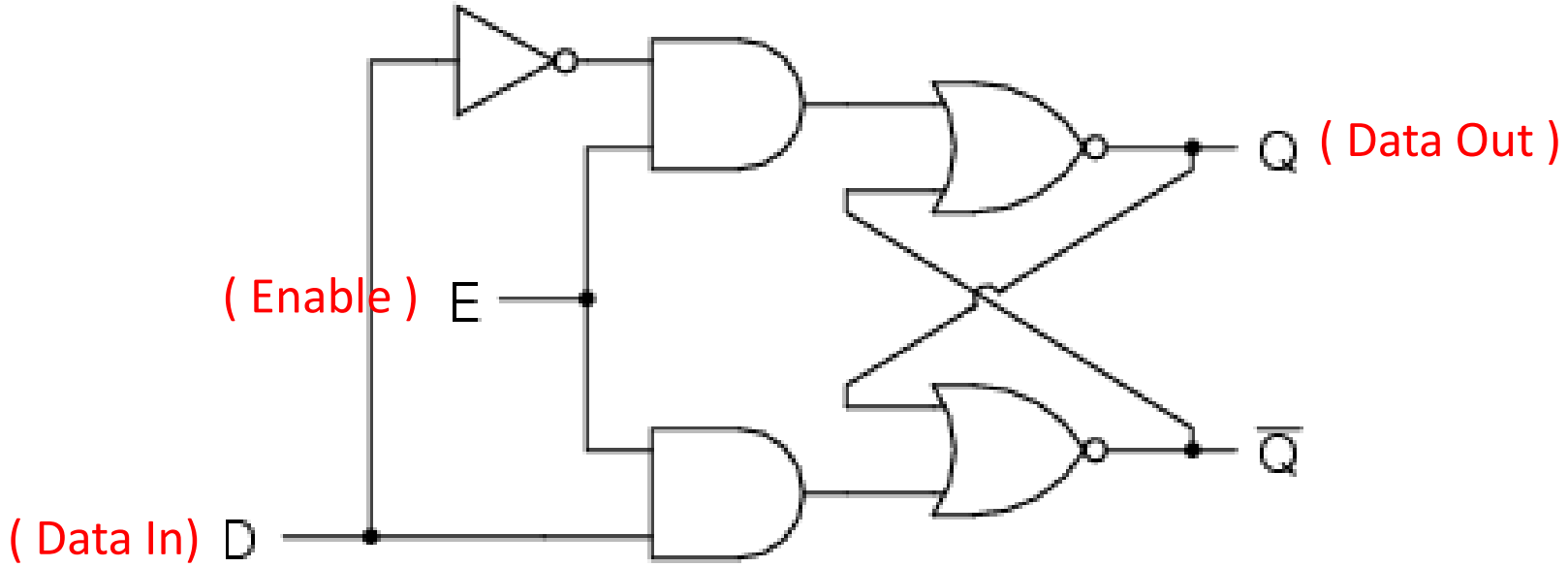
```
int _tmain(int argc, _TCHAR* argv[])
{
    bool set;
    bool reset;

    q     = false;  // Set to their initial values
    q_bar = true;

    //     data_in enable
    memory(false,  false);
    memory(true,   false);
    memory(true,   true);
    memory(true,   false);
    memory(false,  false);
    memory(false,  true);
    memory(false,  false);
    memory(false,  false);

    getchar();
    return 0;
}
```



When enable is toggled the value of data_in is saved into memory (and can be read as data_out)

C Programming – memory circuits

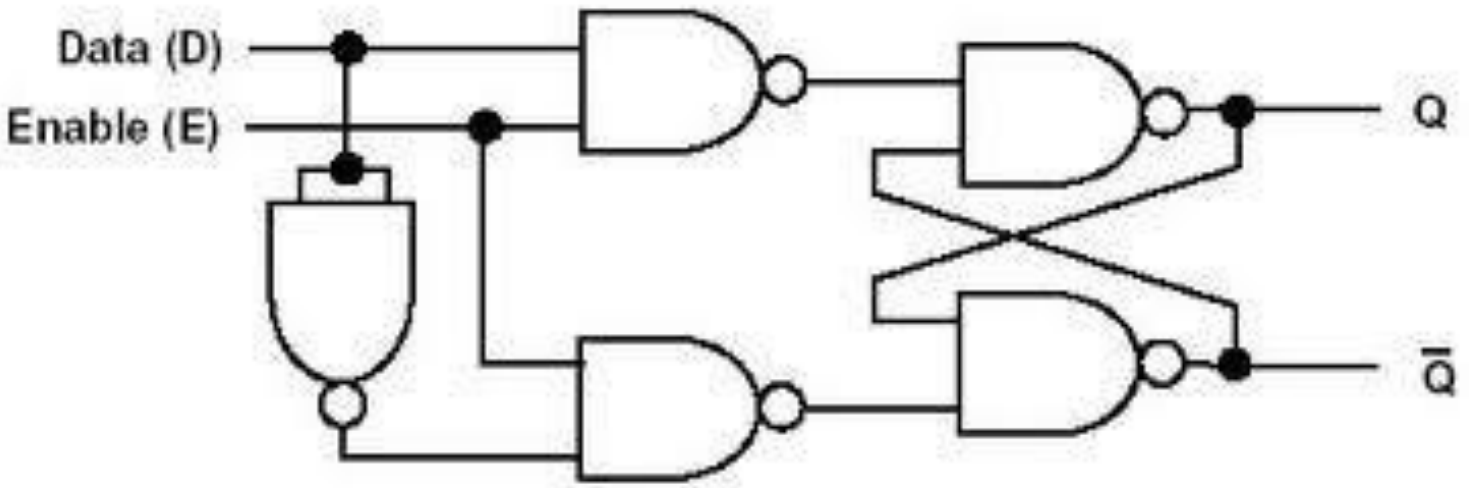
This **memory circuit** can store a single bit



Picture source: Internet

C Programming – memory circuits

Typically a computer circuit will be made up of either NAND or NOR gates, here is a memory circuit made of NAND gates…



Picture source: Internet

The End