ISEN
ALL IS DIGITAL!
yncréa

C Language Programming Reminders

V1.0
10/02/2018

ISEN | école d'ingénieurs
ALL IS DIGITAL!

C Language Programming Reminders

Bitmask

Bitmasking

Bitmasking has the following purpose

- ✓ Masking bits to 1
- ✓ Masking bits to 0
- ✓ Querying the status of a bit
- ✓ Toggling bit values

Masking bits to 1

To turn certain bits on, the bitwise OR operation can be used
Following the principle that Y OR 1 = 1 and Y OR 0 = Y

Therefore, to make sure a bit is ON, OR can be used with a ‘1’

To leave a bit unchanged, OR is used with a ‘0’

Example: masking on the higher nibble (bits 4, 5, 6, 7) the lower nibble (bits 0, 1, 2, 3)

	1001 0101
OR	1111 0000
=	1111 0101

	1010 0101
OR	1111 0000
=	1111 0101

Masking bits to 0

To turn certain bits on, the bitwise AND operation can be used

Following the principle that Y AND 0 = 0 and Y AND 1 = Y

Therefore, to make sure a bit is OFF, AND can be used with a ‘0’

To leave a bit unchanged, AND is used with a ‘1’

Example: masking on the higher nibble (bits 4, 5, 6, 7) the lower nibble (bits 0, 1, 2, 3)

	1001 0101
AND	0000 1111
=	0000 0101

	1010 0101
AND	0000 1111
=	0000 0101

Querying Status bit

To check the state of individual bits regardless of the other bits, all the other bits use the bitwise AND to be turned off

Then the value is compared with ‘1’. If it is equal to ‘0’, then the bit was off (‘0’)

	1001 1101
AND	0000 1000
=	0000 1000

	1001 0101
AND	0000 1000
=	0000 0000

Toggling bit values

In order to toggle, you need need to perform an exclusive OR called XOR

	1001 1101
XOR	1111 1111
=	0110 0010

To write arbitrary 1s and 0s to a subset of bits, first write 0s to that subset, then set the high bits
register = (register & ~bitmask) |

ISEN | école d'ingénieurs
ALL IS DIGITAL!

C Language Programming Reminders

Pointers

What is a Pointer?

ISEN ALL IS DIGITAL! | yncréa

- ✓ For a C programmer, the memory of a computer is like a succession of memory cells, each one byte in size, and each with a unique address
- ✓ This way, each cell can be easily located in the memory by means of its unique address (the Cortex M is viewed as a 4GB memory)
- ✓ For example, the memory cell with the address 1776 always follows immediately after the cell with address 1775 and precedes the one with 1777, and is exactly one thousand cells after 776 and exactly one thousand cells before 2776
- ✓ When a variable is declared, the memory needed to store its value is assigned a specific location in memory

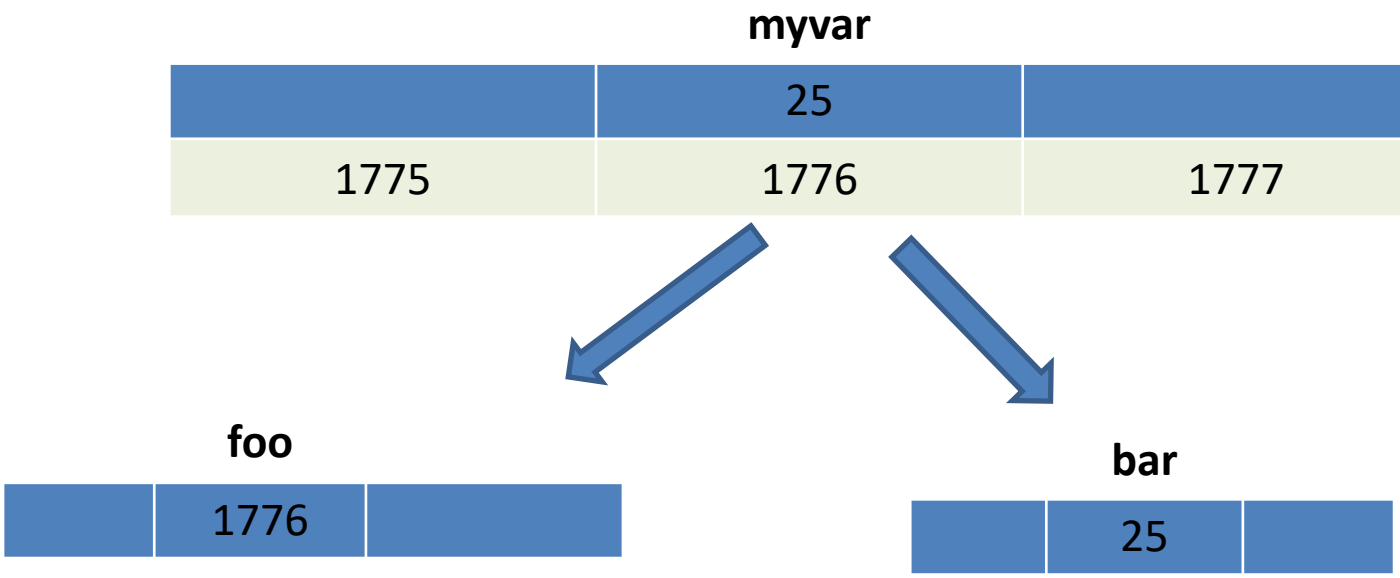
Address of Operator (&)

- ✓ The address of a variable can be obtained by preceding the name of a variable with the sign '&', known as *address-of-operator*
- ✓ Example: foo=&myvar
- ✓ This would assign the address of variable myvar to foo
- ✓ By preceding the name of the variable myvar with the *address-of-operator,* we are no longer assigning the content of the variable itself to foo but its address

Code Snippet

```
myvar = 25;
foo = &myvar
bar = myvar
```

Let's assume myvar was stored at address 1776
The values contained in each variable after the execution of this are shown in the following diagram:



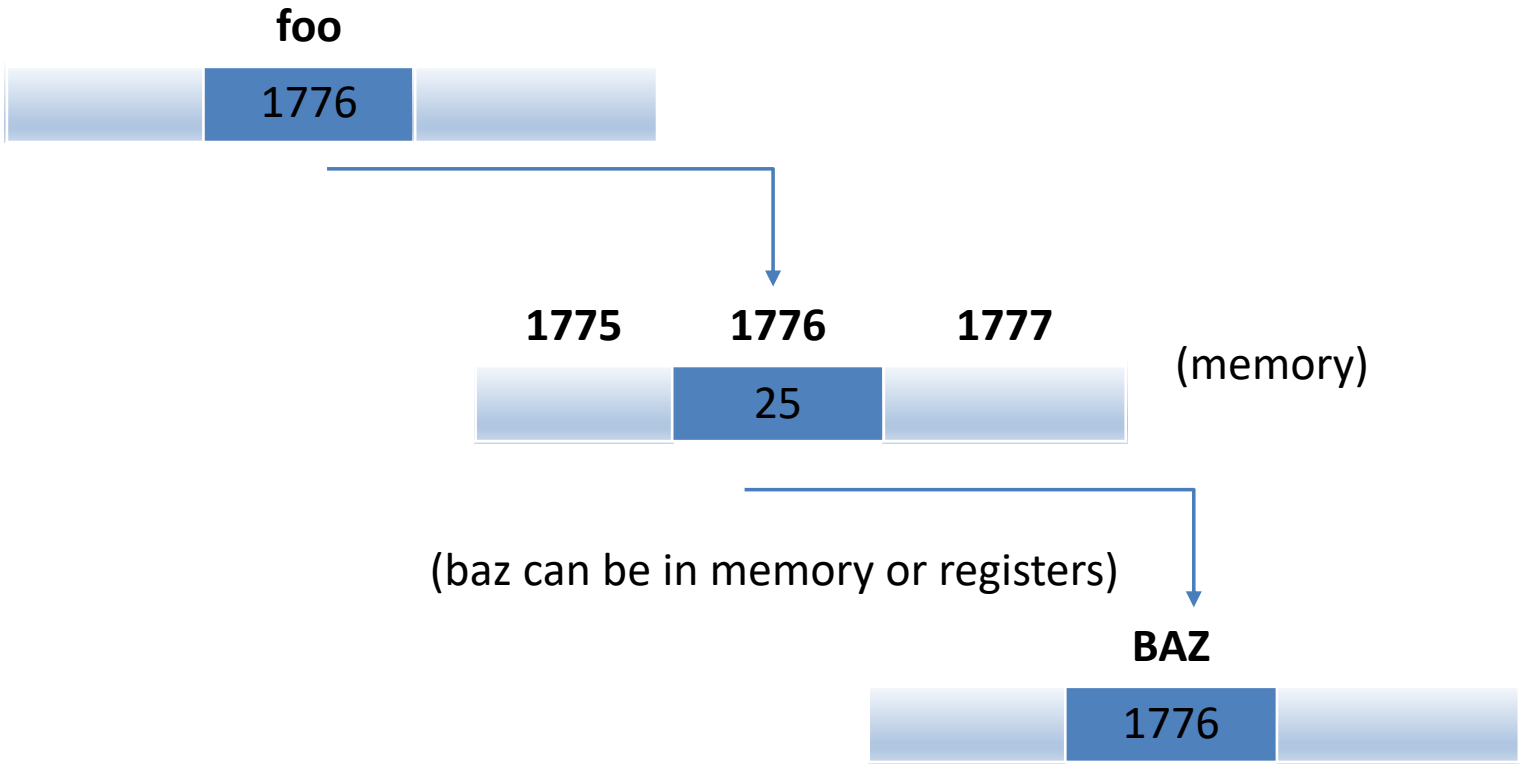
Dereference Operator '*'

- ✓ A variable which stores the address of another variable is called a pointer. Pointers are said to « point to » the variable whose address they store
- ✓ Pointers can be used to access the variable they point to directly. This is done by preceding the pointer name with the dereference operator '*'. The operator itself can be read as « value pointed to by ».

Code snippet `baz = * foo;`

- ✓ This can be read as: baz equal to value pointed to by foo, and the statement assigns the value 25 to baz; since foo is 1776.

Overview


foo
1776
1775
1776
1777
25
(memory)
(baz can be in memory or registers)
BAZ
1776

Summary dereference

ISEN ALL IS DIGITAL! | yncréa

Baz = foo; // baz equal to foo (1776)
Baz = *foo; // baz equal to value pointed to by foo (25)

After this snippet:

```
myvar = 25;
foo = &myvar
bar = myvar
```

The following statements are true:

```
myvar == 25;
&myvar == 1776
foo == 1776
*foo = 25
```

Declaring Pointers

- ✓ Due to the ability of a pointer to directly refer to the value it points to, a pointer has different properties when it points to a char than when it points to an int or a float
- ✓ Once dereferenced, the type needs to be known. And for that, the declaration of a pointer needs to include the data type the pointer is going to point to
- ✓ The declaration of pointers follows this syntax

Type* name

Example of declarations

```
int * number;
char * character;
double * decimals;
```

ISEN
ALL IS DIGITAL!
yncréa

Structures

Pointers and Arrays

The concept of arrays is related to pointers. In fact, arrays work very much like pointers to their first elements
An array can always be implicitly converted to the pointer of the proper type. For example, let's consider these two declarations:

Int myarray[20]

Int * mypointer

The following assignment operation is valid

mypointer = myarray

After that, mypointer and myarray would be equivalent and would have very similar properties. The main difference being that mypointer can be assigned a different address, whereas myarray can never be assigned anything, and will always represent the same block of 20 elements of type int. The following statement is not valid: myarray=mypointer;

Pointer Initialization

Pointers can be initialized to point to specific locations at the very moment they are defined

```
int myvar;
int * myptr = &myvar;
```

The resulting state of variables after this code is the same after

```
int myvar;
int * myptr;
Myptr = &myvar;
```

When pointers are initialized, what is initialized is the address they point to (i.e., myptr), never the value being pointed (i.e. * myptr).

Pointer arithmetics

ISEN ALL IS DIGITAL! | yncréa

To conduct arithmetical operation on pointers is a little different than to conduct them on regular integer types. Only addition and subtraction operations are allowed. The others make no sense

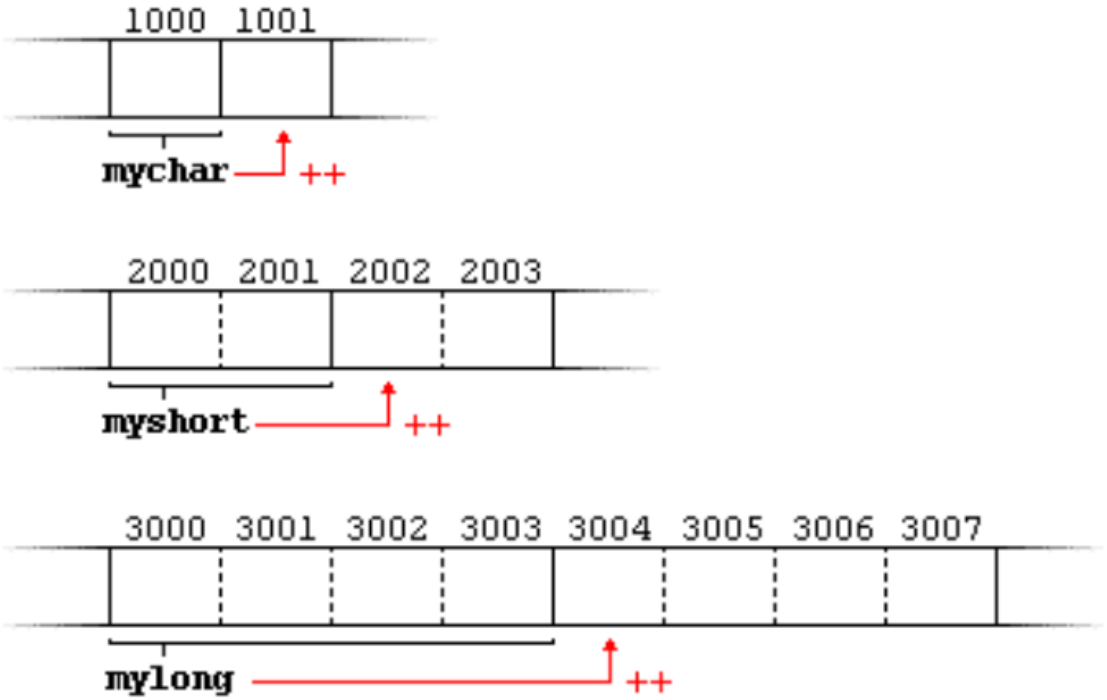
But both addition and subtraction have a slightly different behavior with pointers, according to the size of the data type to which they point (it depends on size of the type)

Pointer arithmetics Snippet

Effect of the below snippet

```
char *mychar;
short * myshort
long *mylong

++mychar
++myshort
++mylong
```



Increment and Decrement

Postfix operators have higher precedence than prefix operators, such as the dereference operator (*)

- ✓ The expression *p++ is equivalent to *(p++). Increment pointer, and dereference unincremented address
- ✓ *++p is equivalent to *(++p): Increment pointer and dereference incremented address
- ✓ ++*p is equivalent to ++(*p): Dereference pointer, and increment the value it points to
- ✓ (*p)++ // dereference pointer, and post increment the value it points to
- ✓ *p++ = *q++ is equivalent to

```
*p = *q;
p++
p++
```

To reduce confusion and add legibility, it is recommended to add parentheses (and avoid mistakes when used in #define expressions)

Pointers and const (1/2)

ISEN ALL IS DIGITAL! | yncréa

```
int x;
int y = 10;
const int * p = &y;
x = *p;          // ok: reading p
*p = x;          // error: modifying p, which is const-qualified
```

Pointers can be used to access a variable by its address, and this access may include modifying the value pointed

But it is also possible to declare pointers that can access the pointed value to read it; but not to modify it. For this, it is enough with qualifying the type pointed to by the pointer as const

Pointers and const (2/2)

```
int x;
      int *       p1 = &x;  // non-const pointer to non-const int
const int *       p2 = &x;  // non-const pointer to const int
      int * const p3 = &x;  // const pointer to non-const int
const int * const p4 = &x;  // const pointer to const int
```

One of the use cases of pointers to const elements is as function parameters: a function that takes a pointer to non const as parameter can modifye the value passed as argument, while a function that takes a pointer to const as parameter cannot

Void Pointers

The *void* type of pointer is a special type of pointer. Void represents the absence of type. Therefore, void pointers are pointer that point to a value that has no type (and thus also an undetermined length and undetermined dereferencing properties)

This gives void pointer a great flexibility, by being able to point to any data type

The data pointed to by the, cannot be directly dereferenced (logical, as there is no type and we don't know the length), therefore any address in a void pointer need to be transformed into some other type pointer that points to a concrete data type before being dereferenced

Use of Void Pointers

One of possible use of void pointers is to pass generic parameters to a function.

```
// increaser
#include <iostream>
using namespace std;

void increase (void* data, int psize)
{
  if ( psize == sizeof(char) )
  { char* pchar; pchar=(char*)data; ++(*pchar); }
  else if (psize == sizeof(int) )
  { int* pint; pint=(int*)data; ++(*pint); }
}

int main ()
{
  char a = 'x';
  int b = 1602;
  increase (&a,sizeof(a));
  increase (&b,sizeof(b));
  cout << a << ", " << b << '\n';
  return 0;
}
```

```
y, 1603
```

Pointer to a Function

Pointers to functions is to used to pass a function as an argument to another function

```
// pointer to functions
#include <iostream>
using namespace std;

int addition (int a, int b)
{ return (a+b); }

int subtraction (int a, int b)
{ return (a-b); }

int operation (int x, int y, int (*functocall)(int,int))
{
  int g;
  g = (*functocall)(x,y);
  return (g);
}

int main ()
{
  int m,n;
  int (*minus)(int,int) = subtraction;

  m = operation (7, 5, addition);
  n = operation (20, m, minus);
  cout <<n;
  return 0;
}
```

8

Pointers to fonctions are declared with the same syntax as a regular function declaration, except that the nale of the function is enclosed between parentheses() and an asterisk (*) is inserted before the name.

ISEN | école d'ingénieurs
ALL IS DIGITAL!

C Language Programming Reminders

Structures

Structure Introduction

- ✓ Arrays allow to define type of variables that can hold several data items of the same kind
- ✓ Structure is another user defined data type available in C that allows to combine data items of different kinds

Structure Purpose

Structures are used to represent a record. Suppose you want to keep track of your books in a library. You might want to track the following attributes about each book

- Tile
- Author
- Subject
- Book ID

These four items can be part of one structure

Defining a Structure

To define a structure, you must use the **struct** statement. The struct statement defines a new data type, with more than one member. The format of the stuct statements is as follow

```
struct [structure tag] {

   member definition;
   member definition;
   ...
   member definition;
} [one or more structure variables];
```

Structure Elements

```
struct [structure tag] {

   member definition;
   member definition;
   ...
   member definition;
} [one or more structure variables];
```

The structure tag is optional and each member definition is a norma variable definition, such as int i or float f or any other valid variable definition. At the end of the structure's definition, you can specify or or more structure variables but it is optional

Structure for Books

ISEN ALL IS DIGITAL! | yncréa

```
struct Books {
   char  title[50];
   char  author[50];
   char  subject[100];
   int   book_id;
} book;
```

Accessing Structure Members

ISEN ALL IS DIGITAL! | yncréa

To access any member of a structure, we use the member access operator (.)

```
struct Books {
   char  title[50];
   char  author[50];
   char  subject[100];
   int   book_id;
};

int main( ) {

   struct Books Book1;        /* Declare Book1 of type Book */
   struct Books Book2;        /* Declare Book2 of type Book */

   /* book 1 specification */
   strcpy( Book1.title, "C Programming");
   strcpy( Book1.author, "Nuha Ali");
   strcpy( Book1.subject, "C Programming Tutorial");
   Book1.book_id = 6495407;
```

Pointers to Structures

ISEN ALL IS DIGITAL! | yncréa

You can define pointers to structures in the same way as you define pointer to any other variable

```
struct Books *struct_pointer;
```

You can store the address of a structure variable in the above defined pointer variable. To find the address of a structure variable, plave the '&' operator before the structure's name

```
struct_pointer = &Book1;
```

Pointers access to structures

To access the members of a structure pointed by a pointer, the structure must be dereferenced followed by the member access operator:

(*struct_pointer).title

This is used so often and as it is a cumbersome method, C provide a shorthand operator for indirect access: the points to operator (->). The above statement is then rewritten as

```
struct_pointer->title;
```