CSE 1320

Week of 01/28/2019

Instructor : Donna French

ANSI C and Integer Types

`limits.h`

`/usr/include/limits.h`

Contains defines that set the sizes of integer types

```
/* Minimum and maximum values a 'signed int' can hold.  */
#  define INT_MIN       (-INT_MAX - 1)
#  define INT_MAX       2147483647

/* Maximum value an 'unsigned int' can hold.  (Minimum is 0.)  */
#  define UINT_MAX      4294967295U
```

`printf()` – field width specifier

```
printf(control_string, args, ...)
```

```
% [flag] [field width] [.precision] [size] conversion
```

field width

- optional
- a decimal integer constant specifying the minimal field width
- output will be right justified and blanks will be used to pad on the left
- will use more space than designated if more space is necessary to output expression

```
int addend1;
int addend2;
int a;

printf("Enter first  addend ");
scanf("%d", &addend1);
printf("\nEnter second addend ");
scanf("%d", &addend2);

printf("\n\t%5d\n", addend1);
printf("\t\b+%5d\n\t", addend2);


for (a = 0; a < 5; a++)
{
   printf("=");
}

printf("\n\t%5d\n", addend1 + addend2);
```

```
Enter first  addend 12

Enter second addend 1234

          12      12
      + 1234     +1234
       =====      =====
        1246       1246
```

```
Enter first  addend 12345

Enter second addend 0

        12345       12345
      +     0      +0
       =====       =====
       12345       12345
```

Floating Point Types

- `float` – single precision
- `double` – double precision
- `long double` – extra precision

```
float floatVar = 3.14;
double doubleVar = 3.14159;
long double longdoubleVar = 3.1415926535897L;

float.h determines the limits of each type
```

For more details on floating point, check out this video
https://www.youtube.com/watch?v=PZRI1IfStY0

```
float       floatVar;
double      doubleVar;
long double longdoubleVar;

The sizeof(float)           is 4
The sizeof(double)          is 8
The sizeof(long double)     is 16

The sizeof(floatVar)       is 4
The sizeof(doubleVar)      is 8
The sizeof(longdoubleVar) is 16

floatVar      = FLT_MAX;
doubleVar     = DBL_MAX;
longdoubleVar = LDBL_MAX;
```

Assigning

340282346638528859811704183484516925440.000000

to floatVar

Assigning

179769313486231570814527423731704356798070567525844996598917476803157260780028538760589558632766878171540458953514382464234321326889464182768467546703537516986049910576551282076245490090389328944075868508455133942304583236903222948165808559332123348274797826204144723168738177180919299881250404026184124858368.000000

to doubleVar

Assigning

11897314953572317650212638530309702051690633222946242004403237338917370055229707226164102903365288828535456978074955773144274431536702884341981255738537436786735932007069732632019159182829615243655295106467910866143117906321697788388961347865606003991487534332114549111600886798451548665128523401497730376000091254793939662231513836224178385427439178381387178058894875405751682263476592355769748051137256490208848552224947913993775850260117735491800997962260268595085588836081598469002356451323465944763849398592764562845796617729304078066092291027150460853880879593277816229868275478307680800401506949423034117289577771003357140105597752421240573470073862516601108283791196230084692772009651535002084744707924438485459128867230006190851264721119513614675276335195629275979572502780029807959041931396030214709970352764674455309220226796562809914982320833296412410385092391847347861219216972105434842870483534081130425730022164213489173471742348007148807510020643905172342476560047217680964861079949434157034763206435586242074435044243805661360176088374781653890278095769759772868600714870282879555671414046326158326236027628963161739784842544868606099482708679680480787025118589308385465842230409088059962945945862019037660484467909260022254105307759010657606713472001258464069570302571389609837579989269545530523685607586831792231136395194688508807718721047052039575874800131431314442549439199401757531693393923668818561891299317291042529212368351599223220509980016771027840353601408292963981151228777681357060457893435345169653956125404884644716978689321167108722908808277835051822885764606221873970285165508372099234948333443522898475123275372663606621390228126470623407535207172405866507951821730346378263135339370677490195019784169044182473806316282858685774143258116536404021840272491339332094921949842244273042701987304453662035026238695780468200360144729199712309553005720614186697485284685618651483271597448120312194675168637934309618961510733006555242148519520176285859509105183947250286387163249416761380499631979144187025430270675849519200883791516940158174004671147787720145964446117520405945350476472180797576111172084627363927960033967047003761337450955318415007379641260504792325166135484129188421134082301547330475406707281876350361733290800595189632520707167390454777712968226520622565143991937680440029238090311243791261477625596469422198137514696707944687035800439250765945161837981185939204954403611491531078225107269148697980924094677214272701240437718740921675661363493890045123235166814608932240069799317601780533819184998193300841098599393876029260139091141452600372028487213241195542428210183120421610446740462163533690058366460659115629876474552506814500393294140413149540067760295100596225302282300363147382468105964844244132486457313743759509641616804802412935187620466813563687753281467553839888717718365128939471953350618850032676073543886733680020743878496570145760903498575712430451020387304948542567024793393228091105260415385289948492039910919461299124916332899179980943803378795220931314669461497059396641523759492858909604899161219449899863848370224866722491489246784102061833646274169695763076324802355879752452537370354338829608627534277400163334340550835370485073745448197547222289752810830208986826330202852599230841680545396879114182976299889645764827652875045628549242651652177507995162596692291149777889623566709566271384820181913483216879958636526376209782850700993372943967846398790249145142227425270063639423279984839767399871544185542015622441549266530145155046854892586202760857618371297633587612153825651296335381416639495165560002641591865548500570526114319529199188079545223946496276356301785808966922264062353828985358675959906470083856871238103295919264948462507689922584193054807636202150890221492205280698420183508405869384938154989094454619778930291135765167754062322782983140334732766039522316034228247175281818188443048809213218933550869873395861276073670866523755556758031714901084773200964243187800700087973460329062789435537435644488519071916164551411557619393996907674151564028265436640267600950875239455073415561358679330660317447209244465135323666476497354008519670407711036405381500734868917983640495706061895350050898409138268695350900667833244725787121966044152849248400418509328119089636341757398971665960007594878006191640948543387585205711654107226099628815012314437794400874930194474433078438899570184271000480830501217712356062289507626904285680004771889315808935851559386317665294808903126774702966254511086154895839508779675546413794489596052797520987481383976257859210575628440175934932416214833956535018919681138909184379573470326940634289008780584694035245347939808067427323629788710086717580253156130235606487870925986528841635097252953709111431720488774740553905400942537542411931794417513706468964386151771884986701034153254238591108962471088538580868883777725864856414593426212108664758848926003176234596076950884914966244415660441955208681198977024 0.000000

to longdoubleVar

```
The sizeof(floatVar)      is 4
The sizeof(doubleVar)     is 8
The sizeof(longdoubleVar) is 16
```

The contents of a variable do not change the `sizeof()` that variable.

Floating Point Types

Using operators with floating point types.

arithmetic	+	-	*	/		
relational	==	!=	<	<=	>	>=
logical	!	&&	||			

Expression	Value	Type
`2.5 + 5.7`	`8.2`	`double`
`2.5 <= 3.62`	`1` (true)	`int`
`2.5 == 3.62`	`0` (false)	`int`
`2.5 / 3.62`	`0.6906`	`double`
`2.5 && 3.62`	`1` (true)	`int`
`!2.5`	`0` (false)	`int`
`!0`	`1` (true)	`int`

Input and Output of Floating Point Values

Conversion Specifications for `scanf()`

```
%e %f %g      float
%le %lf %lg   double
%Le %Lf %Lg   long double
```

Conversion Specifications for `printf()`

```
%e   %f   %g   %E   %G     float, double
%Le %Lf %Lg %LE %LG   long double
```

For more about scientific notation

https://www.youtube.com/watch?v=Hmw0wJVud0k

perconDemo.c
psconversionDemo.c

```
Enter a float value for %e              12.3456
Value entered using %e   is             1.234560e+01
Value entered using %.2e is             1.23e+01

Enter a float value for %f              12.3456
Value entered using %f   is             12.345600
Value entered using %.3f is             12.346

Enter a double value for %le            12.3456
Value entered using %le is              1.234560e+01
Value entered using %.4le is            1.2346e+01

Enter a float value for %g              12.3456
Value entered using %g is               12.3456
Value entered using %.2g is             12

Enter a double value %lg                12.3456
Value entered using %lg is              12.3456
Value entered using %.3lg is            12.3

Enter a double long value for %Lg       12.3456
Value entered using %Lg is              12.3456
Value entered using %.4LG is            12.35
```

`printf()` – precision specification

```
printf(control_string, args, ...)
```

```
% [flag] [field width] [.precision] [size] conversion
```

.precision

- optional
- a period followed by a decimal integer specifying the number of digits to be printed in a conversion of a floating point value after the decimal point

```
float f1 = 1;
float f3 = 3;

double d1 = 1;
double d3 = 3;

long double ld1 = 1L;
long double ld3 = 3L;

printf("float version       of 1/3 %.65f\n\n",
       f1/f3);
printf("double version      of 1/3 %.65f\n\n",
       d1/d3);
printf("long double version of 1/3 %.65Lf\n\n",
       ld1/ld3);

printf("sum = %.65Lf\n\n",
       f1/f3 + d1/d3 + ld1/ld3);
```

```
float version       of 1/3
0.33333334326744079589843750
00000000000000000000000000000
0000000000000

double version      of 1/3
0.33333333333333331482961625
62473909929394721984863281
2500000000000

long double version of 1/3
0.33333333333333333423683
51437379203616728773340582
8475952148437 5

sum =
1.00000000993410738858637593
07390807862248038873076438
9038085937500
```

floatpreDemo.c

Types of Expressions

- every expression has an associated type
- operators and operands within the expression determine the expression's type
- in a binary operation, both operands are converted to the dominating type before being evaluated
- result will retain the dominate type
- most to least dominate
  - `long double`
  - `double`
  - `float`
  - `unsigned long`
  - `long`
  - `unsigned`
  - `int`

```
int a;
float b;
float c;

c = a + b;

a would be converted to float
```

This type of conversion is called automatic typecasting.

Forced Type Conversions

type cast

- the type of an expression can be temporarily changed with a type cast
- pair of parentheses enclosing a type specifier
- can be constructed with any of the basic types in C
- no restrictions on the use of type casts
- any type in C can be cast to any other type
  - data may be lost

C:\Users\Donna\Desktop\UTA\Programs\CSE1320\typecast3Demo.c - Notepad++

File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?

typecast2Demo.c typecast3Demo.c

```c
/* typecast 3 demo */

#include <stdio.h>

int main(void)
{
    int int1 = 1;
    int int2 = 2;
    int int3 = 4;

    printf("Enter a value for int1 ");
    scanf("%d", &int1);

    printf("Enter a value for int2 ");
    scanf("%d", &int2);

    printf("Enter a value for int3 ");
    scanf("%d", &int3);

    printf("Sum = %d\n", int1+int2+int3);
    printf("Average = %d\n", (int1+int2+int3)/3);

    return 0;
}
```

C source file | length : 407 | lines : 25 | Ln : 2 | Col : 1 | Sel : 0 | 0 | Windows (CR LF) | UTF-8 | INS

emo.c

XOR

- Only TRUE if one or the other is true but not both.
- One or the other but not both.

p	q	p XOR q
T	T	F
T	F	T
F	T	T
F	F	F

You can have ice cream or pie for dessert but not both.

You can go to sleep at 8AM or go to work at 8AM but not both.

Bit Operations on the Integer Types

Bit operations

~	bitwise negation
>>	shift right
<<	shift left
&	bitwise and
^	bitwise xor
|	bitwise or

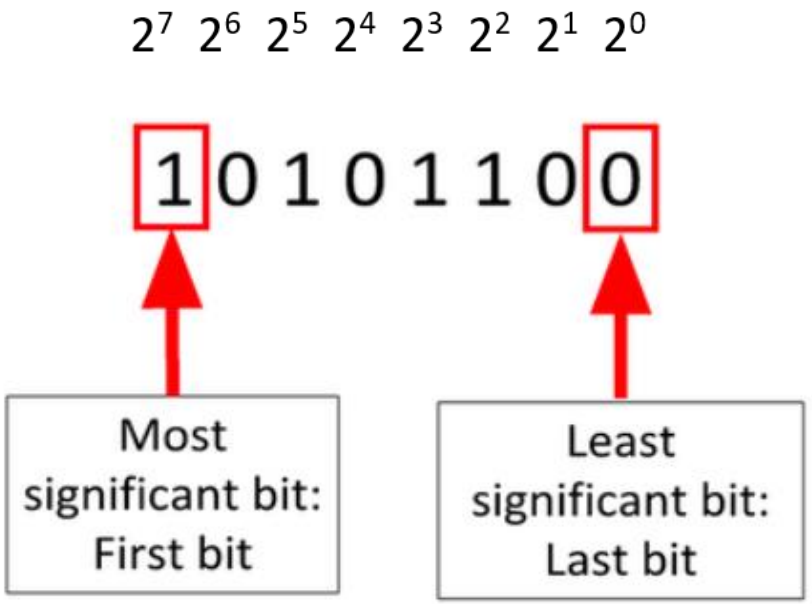
How To Read Bits

Least Significant Bit (LSB)

the bit in a binary number that is of the lowest numerical value

Most Significant Bit (MSB)

the bit in a binary number that is of the highest numerical value



Bit Operations on the Integer Types

How are they used? Why are we learning this?

Gaming software

- performance
  - deciphering online game protocols
- image masking – when one image needs to be placed over another
- 3D games to determine distances

IP addresses

- specify what is permitted and what is denied

Image compression/decompression

Bit Operations on the Integer Types

bitwise negation

~expression

where expression has an integer type

replaces all the 0 bits by 1 and all of the 1 bits by 0

a short will be represented by 16 bits/2 bytes

0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**

Bitwise Negation

~0000000000000001

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

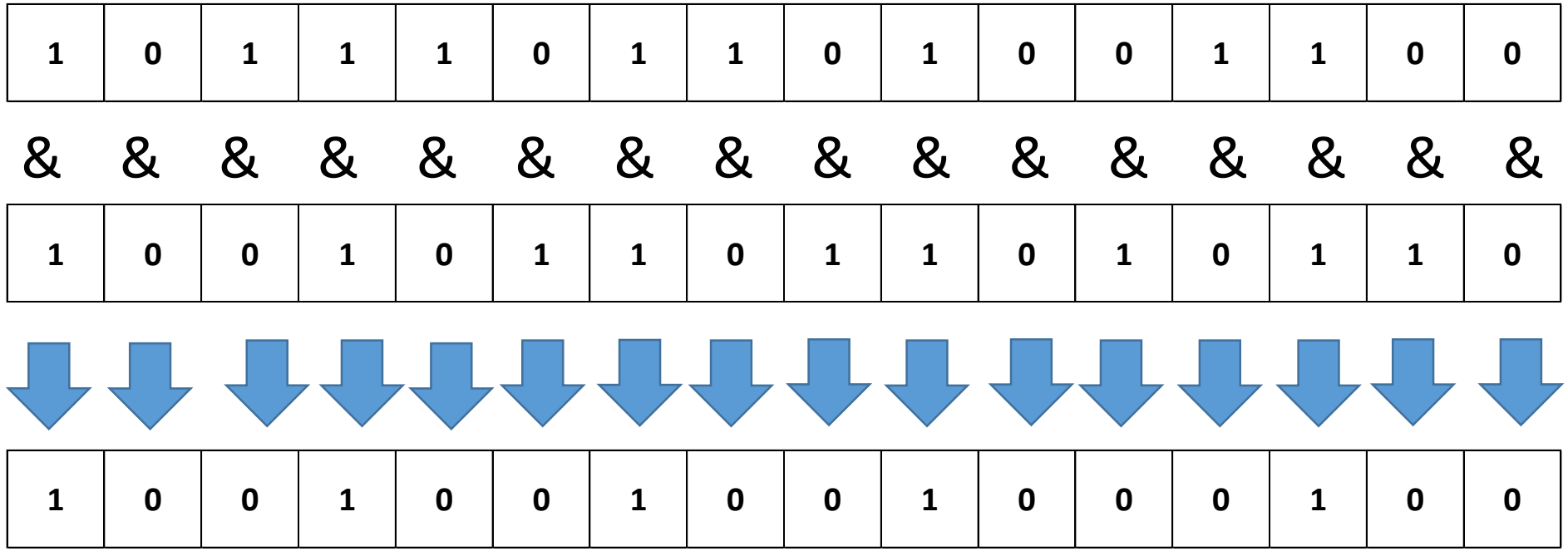
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0

1	0	0	1
0	1	1	0
1	0	0	1

0	1	1	0
1	0	0	1
0	1	1	0

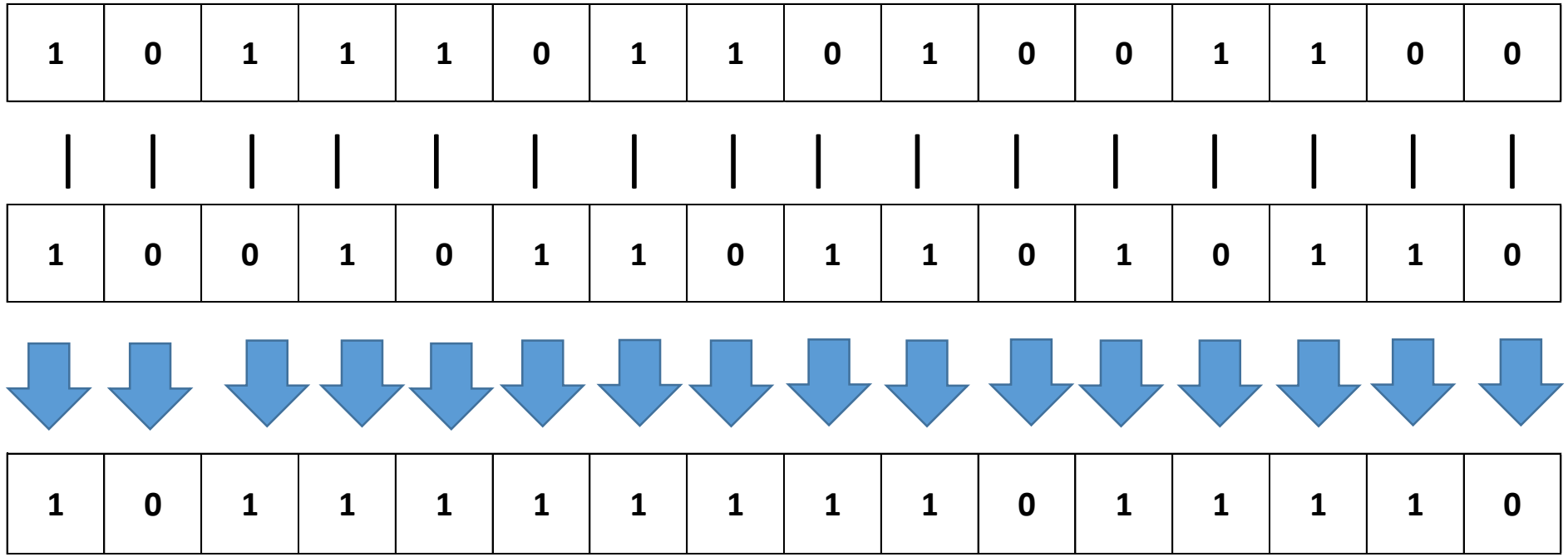
Bitwise AND

1011101101001100 & 1001011011010110



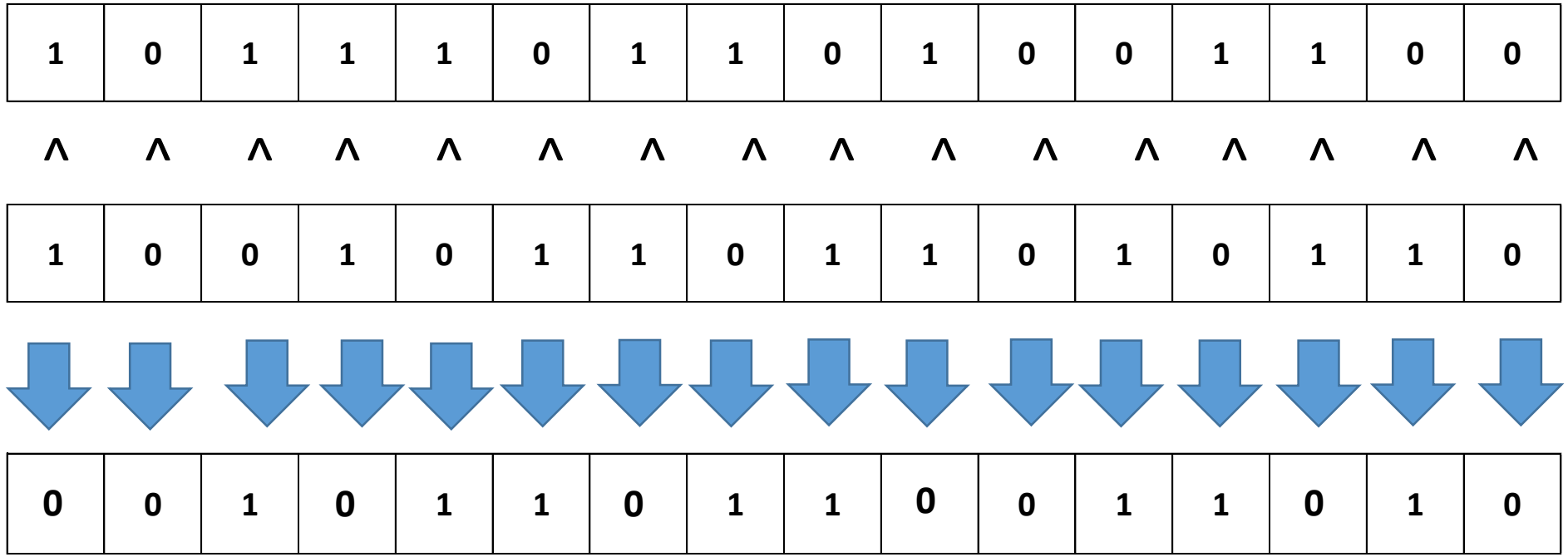
Bitwise OR

1011101101001100 | 1001011011010110



Bitwise XOR

1011101101001100 ^ 1001011011010110



Precedence of Bitwise Operators

bitwise negation

bitwise and

bitwise xor

bitwise or

Associate from left to right

Using Masks

Bit masks can be used

- to detect whether or not a certain bit is on or off
- to turn a bit on or off

Individual bits can be used as flags (0 has a certain interpretation and 1 has a different interpretation).

Masks can be used to evaluate and manipulate each bit.

Question – is the 4th bit on or off in 345?

```
number   345      0000000101011001
&        &        &
mask     16       0000000000010000
         ------   ----------------
         16       0000000000010000
```

Question – is the 2nd bit on or off in 123?

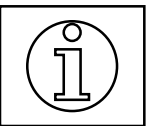
```
number   123      0000000001111011
&        &        &
mask     4        0000000000000100
         ------   ----------------
         0        0000000000000000
```

Question – is the 4th bit on or off in 200?

```
number   200      0000000011001000
&        &        &
mask     16       0000000000010000
         ------   ----------------
         0        0000000000000000
```

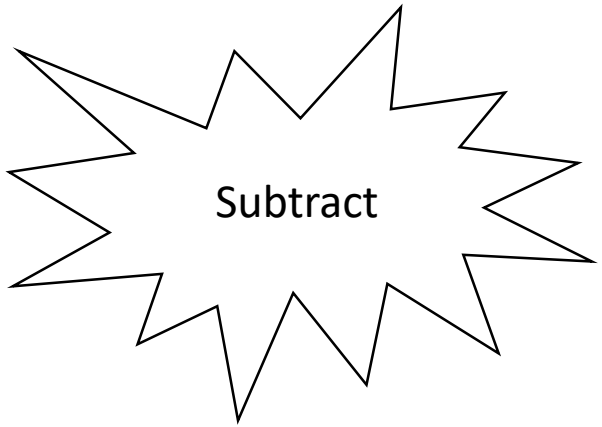
Question – is the 6th bit on or off in 124?

```
number   124      0000000001111100
&        &        &
mask     64       0000000001000000
         ------   ----------------
         64       0000000001000000
```



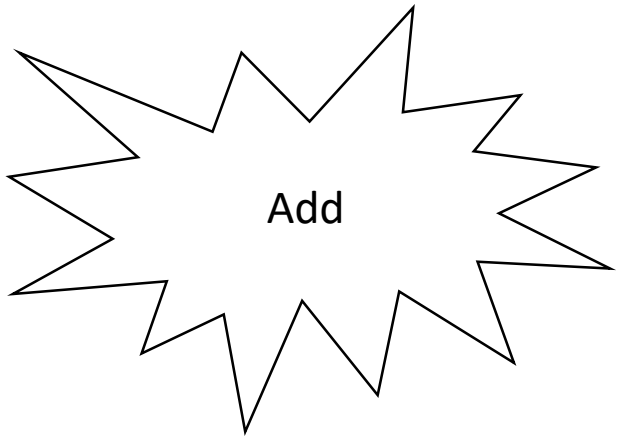
Question – what happens when the 4th bit is turned off in 345?

```
number   345      0000000101011001
^        ^        ^
mask     16       0000000000010000
         ------   ----------------
         329      0000000101001001
```



Question – what happens when the 4th bit is turned on in 200?

```
number   200      0000000011001000
^        ^        ^
mask     16       0000000000010000
         ------   ----------------
         216      0000000011011000
```



Question – what happens if we use a non power of 2 bit mask?

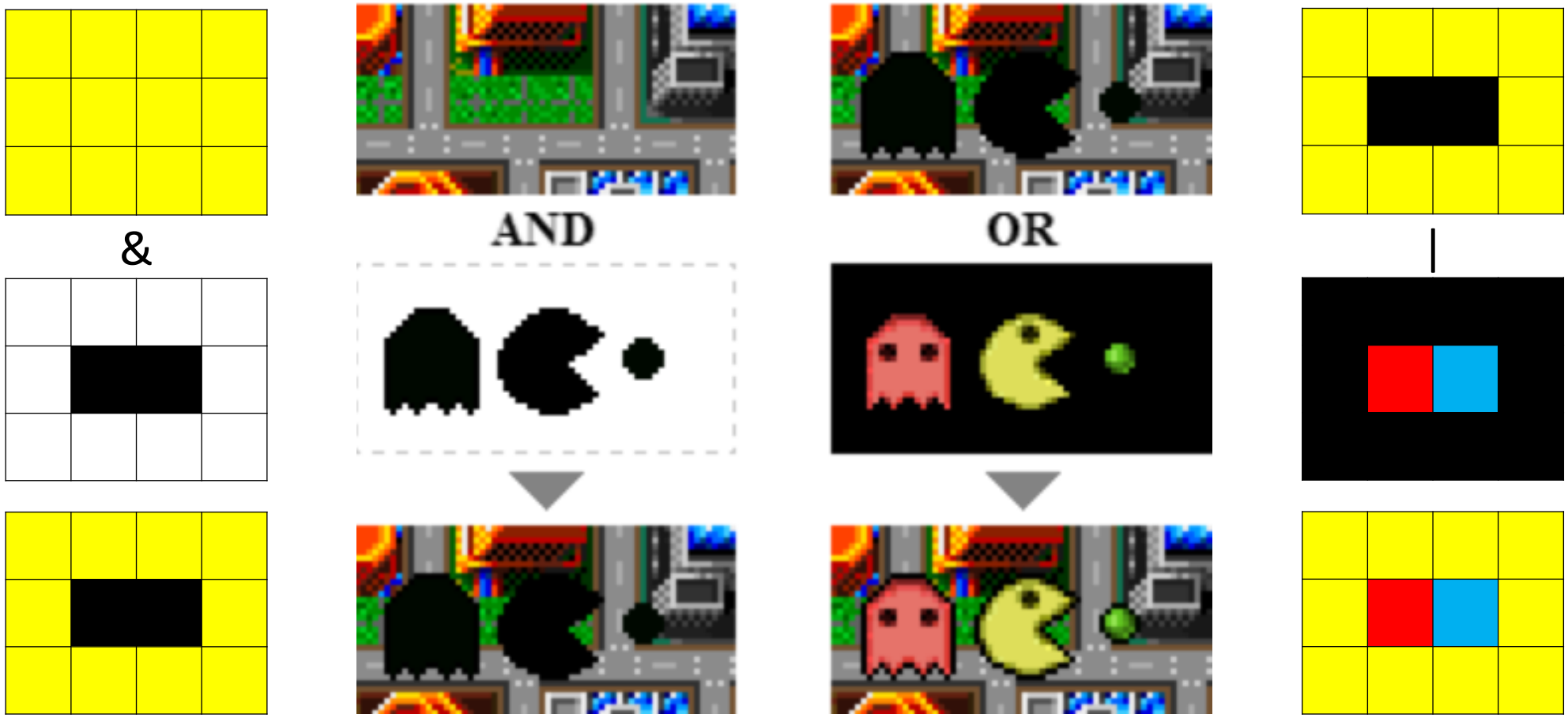
```
number   345      0000000101011001
^        ^        ^
mask     14       000000000000111 0
         ------   ----------------
         343      0000000101010111
```

Did it add? Did it subtract? How did we get from 345 to 343?

It added and subtracted!

Turning a bit on adds and turning a bit off subtracts the power of 2 for that bit.

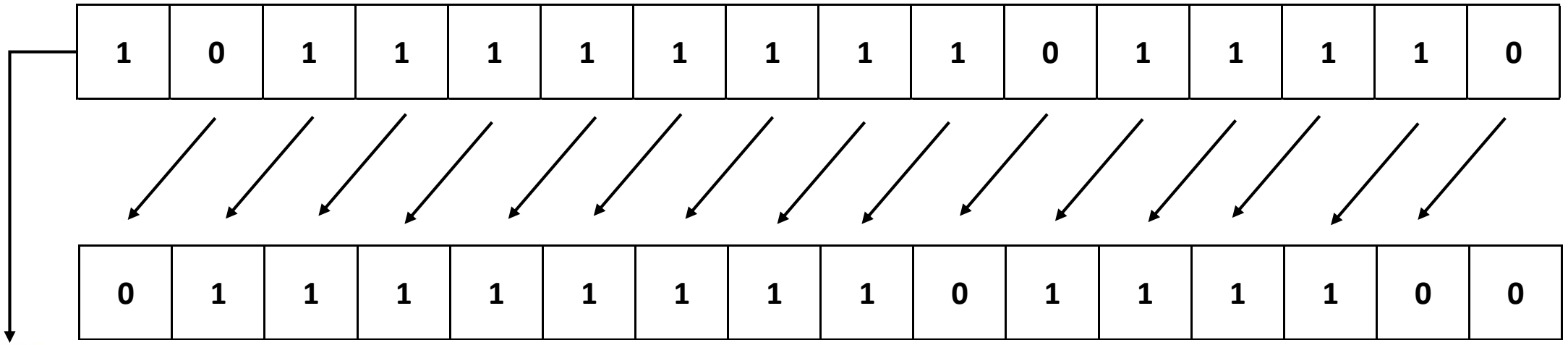
Bitwise AND and Bitwise OR



Bit Shifting

left shift

```
expression1 << expression 2
```

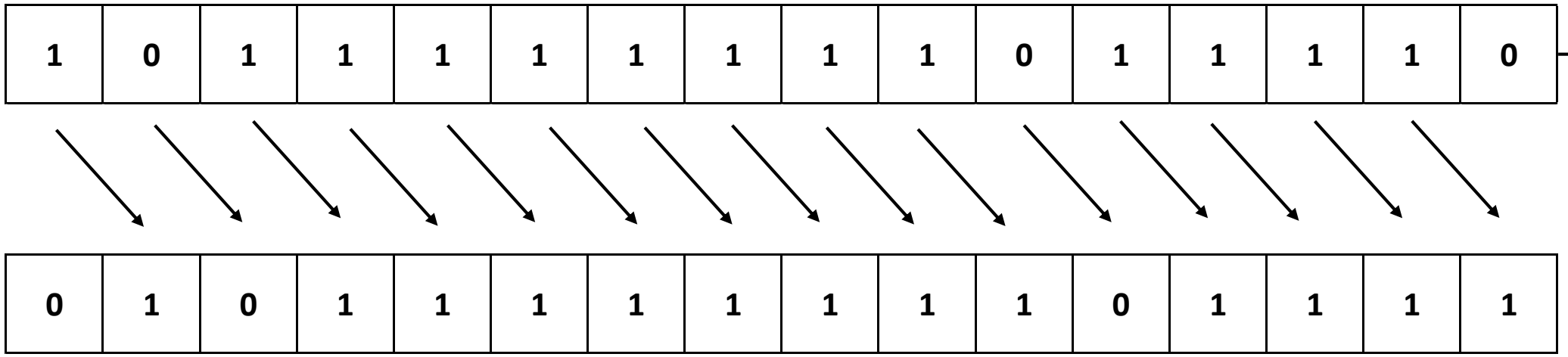


Bitbucket

Bit Shifting

right shift

```
expression1 >> expression 2
```



Bitbucket

frenchdm@omega:~

```
[frenchdm@omega ~]$ a.out
```

Type here to search

12:04 PM
1/25/2019

Using >> for Division

>> 0	2^0	Divide by 1	100 >> 0 = 100
>> 1	2^1	Divide by 2	100 >> 1 = 50
>> 2	2^2	Divide by 4	100 >> 2 = 25
>> 3	2^3	Divide by 8	100 >> 3 = 12
>> 4	2^4	Divide by 16	100 >> 4 = 6
>> 5	2^5	Divide by 32	100 >> 5 = 3
>> 6	2^6	Divide by 64	100 >> 6 = 1
>> 7	2^7	Divide by 128	100 >> 7 = 0

Using << for Multiplication

<< 0	2^0	Multiple by 1	1 << 0 = 1
<< 1	2^1	Multiple by 2	1 << 1 = 2
<< 2	2^2	Multiple by 4	1 << 2 = 4
<< 3	2^3	Multiple by 8	1 << 3 = 8
<< 4	2^4	Multiple by 16	1 << 4 = 16
<< 5	2^5	Multiple by 32	1 << 5 = 32
<< 6	2^6	Multiple by 64	1 << 6 = 64
<< 7	2^7	Multiple by 128	1 << 7 = 128

Compiler Warning and Overflow

cw1Demo.c

```
#include <stdio.h>
#include <limits.h>

int main(void)
{
  short VarA;

  printf("Enter a value ");
  scanf("%d", &VarA);

  if (VarA > SHRT_MAX)
  {
    printf("VarA is larger than a max short\n");
  }

  printf("\nYou entered %d\n\n", VarA);
}
```

SHRT_MAX is 32767

Function Definitions

function definition includes

- function type
- function name
- names, types and number of formal parameters
- executable statements for the function

```
type function_name(int param1, int param2)
{
   /* function code goes here */
}

void MyFunction(int DecNum)
{
   print("DecNum is %d\n", DecNum);
}
```

Function Types

default type for C is type `int`

- any other type must be explicitly declared

type of the function matches the type of the expression associated with its return value

void can be used to not return a value

- no return statement
- return without an expression

```
int main(void)
{
  int addend1;
  int addend2;

  system("clear");

  printf("Enter first  addend ");
  scanf("%d", &addend1);
  printf("\nEnter second addend ");
  scanf("%d", &addend2);

  PrintSum(addend1, addend2);

  return 0;
}
```

```
void PrintSum(int Add1, int Add2)
{
  int a;

  printf("\n\t%5d\n", Add1);
  printf("\t\b+%5d\n\t", Add2);

  for (a = 0; a < 5; a++)
  {
    printf("=");
  }

  printf("\n\t%5d\n", Add1 + Add2);

  return;
}
```

function1Demo.c

```c
int main(void)
{
  int addend1;
  int addend2;

  system("clear");

  printf("Enter first  addend ");
  scanf("%d", &addend1);
  printf("\nEnter second addend ");
  scanf("%d", &addend2);

  printf("\n\t%5d\n", PrintSum(addend1, addend2));

  return 0;
}
```

```c
int PrintSum(int Add1, int Add2)
{
  int a;

  printf("\n\t%5d\n", Add1);
  printf("\t\b+%5d\n\t", Add2);

  for (a = 0; a < 5; a++)
  {
    printf("=");
  }

  return Add1 + Add2;
}
```

function2Demo.c

```
int main(void)
{
   int addend1;
   int addend2;
   int ValidAddend;
   int RunAgain = 1;

   system("clear");

   printf("Addends must be <= %d\n\n", SHRT_MAX);

   while (RunAgain)
   {
      printf("Enter first  addend ");
      scanf("%d", &addend1);

      if (CheckAddend(addend1))
      {
         printf("\nEnter second addend ");
         scanf("%d", &addend2);

         if (ValidAddend = CheckAddend(addend2))
         {
            printf("\n\t%5d\n", PrintSum(addend1, addend2));
         }
      }

      RunAgain = AskToRunAgain();
   }

   return 0;
}
```

```
int PrintSum(short Add1, short Add2)
{
   int a = 1;

   printf("\n\t%5hu\n", Add1);
   printf("\t\b+%5hu\n\t", Add2);

   for (a = 0; a < 5; a++)
   {
      printf("=");
   }

   return Add1 + Add2;
}

int CheckAddend(int Input)
{
   if (Input > SHRT_MAX)
   {
      printf("\nAddend %d is too large\n", Input);
      return FALSE;
   }

   return TRUE;
}

int AskToRunAgain(void)
{
   int Again = 0;

   printf("\nDo you want to add two more numbers? (0=NO/1=YES) ");
   scanf("%d", &Again);

   if (Again)
      system("clear");

   return Again;
}
```

function3Demo.c

Function Prototypes

A function prototype is a declaration of a function that tells the compiler the function's name, its return type and the types of its parameters.

The function prototype is the *same* as the first line of the corresponding function definition, but ends with a *required* semicolon.

Function Definition

```
int PrintSum(short Add1, short Add2)
```

Function Prototype

```
int PrintSum(short Add1, short Add2);
int PrintSum(short, short);
```

Variable names are optional

Function Prototypes

The compiler uses the prototype to

- Ensure that the function definition matches the function prototype.
- Check that the function call contains the correct number and types of arguments and that the types of the arguments are in the correct order.
- Ensure that the value returned by the function can be used correctly in the expression that called the function—for example, for a function that returns void you cannot call the function on right side of an assignment.
- Ensure that each argument is consistent with the type of the corresponding parameter—for example, a parameter of type double can receive values like 7.35, 22 or –0.03456, but not a string like "hello".
- If the arguments passed to a function do not match the types specified in the function's prototype, the compiler attempts to convert the arguments to those types.

Function Prototypes

What happens if we move `main()` to the top of the program but do not add prototypes?

*C:\Users\Donna\Desktop\UTA\Programs\CSE1320\function3Demo.c - Notepad++

File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?

passarray2Demo.c | function3Demo.c | new 1

```c
/* function 3 demo */

#include <stdio.h>
#include <limits.h>

#define TRUE  1
#define FALSE 0

int PrintSum(short Add1, short Add2)
{
    int a = 1;

    printf("\n\t%5hu\n", Add1);
    printf("\t\b+%5hu\n\t", Add2);

    for (a = 0; a < 5; a++)
    {
        printf("=");
    }

    return Add1 + Add2;
}

int CheckAddend(int Input)
{
    if (Input > SHRT_MAX)
    {
        printf("\nAddend %d is too large\n", Input);
        return FALSE;
    }

    return TRUE;
```

Function Prototypes

```
[frenchdm@omega ~]$ gcc function3Demo.c
function3Demo.c:43: error: conflicting types for ‘PrintSum’
function3Demo.c:43: note: an argument type that has a default
promotion can’t match an empty parameter name list declaration
function3Demo.c:32: error: previous implicit declaration of
‘PrintSum’ was here
```

```
32                  printf("\n\t%5d\n", PrintSum(addend1, addend2));
33              }
34          }
35
36          RunAgain = AskToRunAgain();
37      }
38
39      return 0;
40  }
41
42  int PrintSum(short Add1, short Add2)
43  {
```

One Dimensional Arrays

Arrays

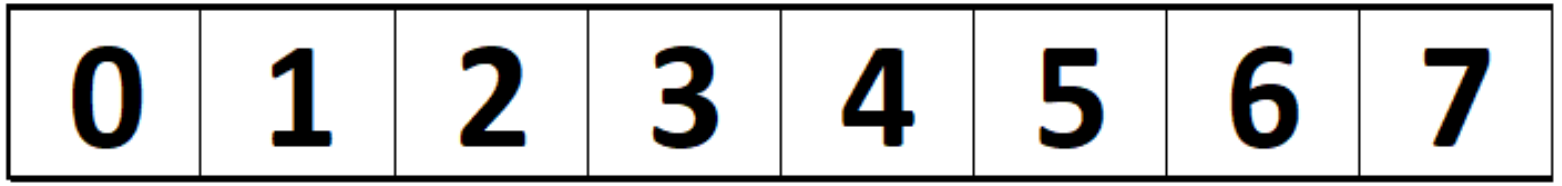
- aggregate type
- used to store collections of related data
- multiple values of the same data type can be stored with one variable name

One Dimensional Arrays

each data object occupies one cell of the array

spice rack

index




```
int spice_rack[8]
```

- each cell is type `int`
- 8 cells
- array name is `spice_rack`

- indicies are 0 through 7
  - must be integers
  - first cell is always 0
  - last cell is 1 less than the total number of cells

Initialization of Arrays

Arrays can be initialized 2 ways

Method 1 - the array declaration

- comma separated list enclosed in braces
- initial values can be constants or expression using declared and initialized variables

```
int MyArray[10] = {12,42,63,48,59,62,77,82,91,10};
char MyCharArray[10] = {'A','B','C','D','E','F','G','H','I','J'};
char MyCharArray[] = {"ABCDEFGHIJ"};
```

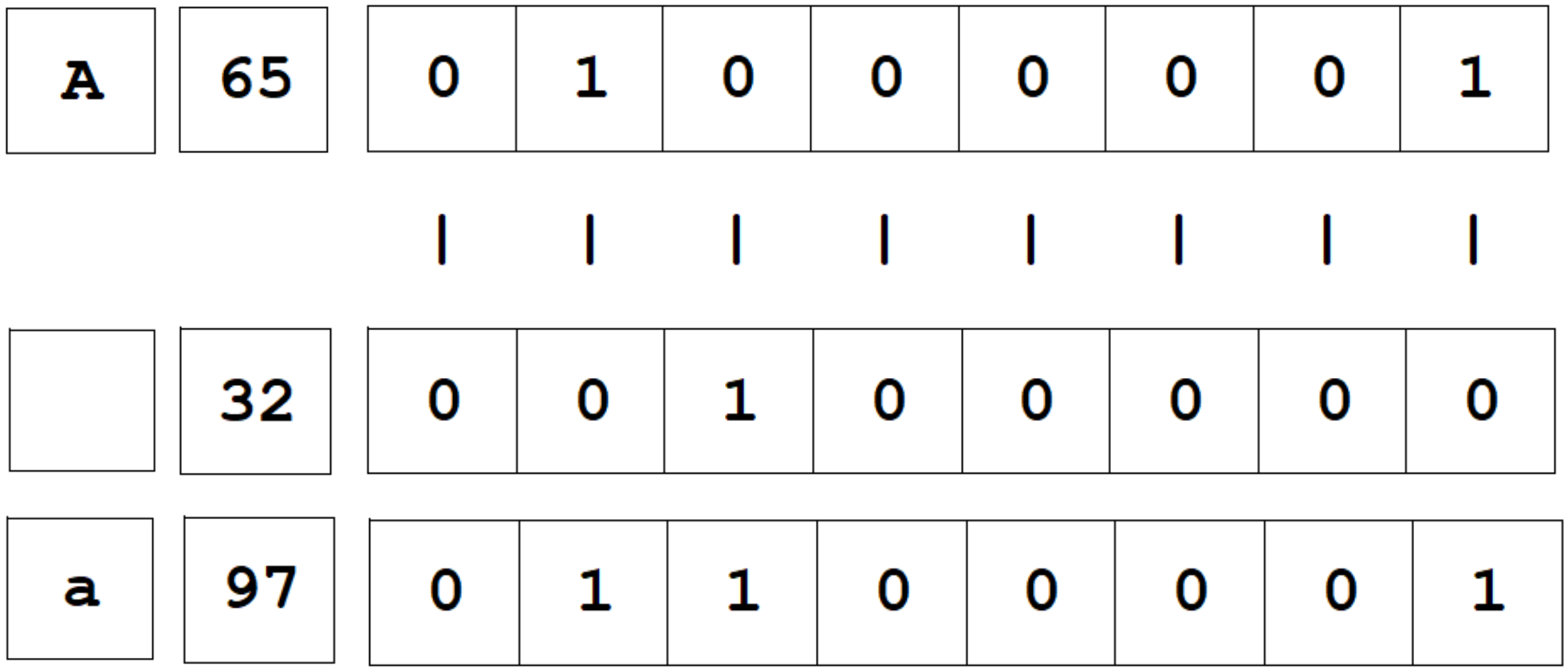
```
int i;
int MyArray[10] = {12,42,63,48,59,62,77,82,91,10};
int Choice;
char MyCharArray[10];

for (i = 0; i < 10; i++)
{
  printf("MyArray[%d] = %d\n", i, MyArray[i]);
}

printf("Which array element do you want to see? ");
scanf("%d", &Choice);
printf("The value of array element %d is %d\n", Choice, MyArray[Choice]);

for (i = 0; i < 10; i++)
{
  MyArray[i] = MyArray[i] >> 1;
  printf("MyArray[%d] = %d\n", i, MyArray[i]);
}

printf("Which array element do you want to see? ");
scanf("%d", &Choice);
printf("The value of array element %d is %d\n", Choice, MyArray[Choice]);
```

```
MyArray[0] = 12
MyArray[1] = 42
MyArray[2] = 63
MyArray[3] = 48
MyArray[4] = 59
MyArray[5] = 62
MyArray[6] = 77
MyArray[7] = 82
MyArray[8] = 91
MyArray[9] = 10
Which array element do you want to see? 7
The value of array element 7 is 82
MyArray[0] = 6
MyArray[1] = 21
MyArray[2] = 31
MyArray[3] = 24
MyArray[4] = 29
MyArray[5] = 31
MyArray[6] = 38
MyArray[7] = 41
MyArray[8] = 45
MyArray[9] = 5
Which array element do you want to see? 0
The value of array element 0 is 6
```

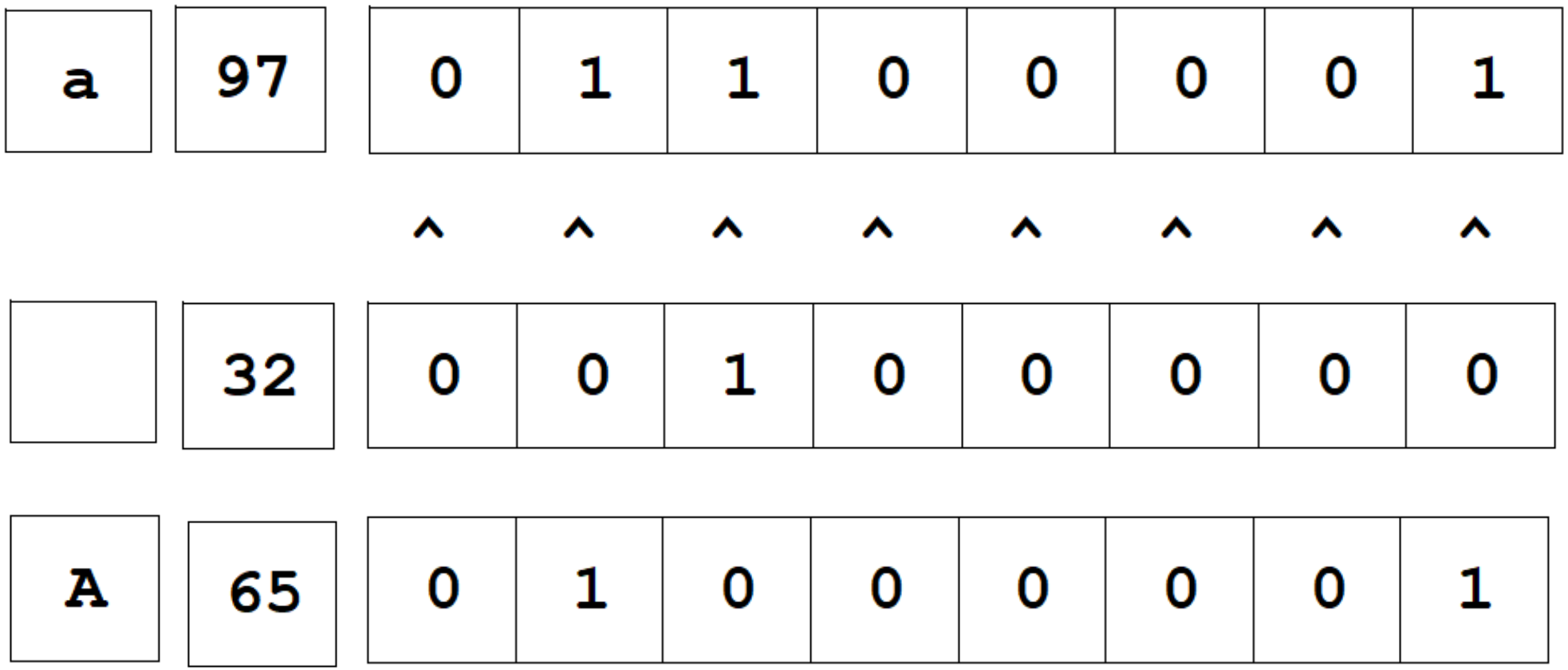
```
int main(void)
{
	int i;
	int Choice;
	char MyCharArray[10] = {'A','B','C','D','E','F','G','H','I','J'};

	for (i = 0; i < 10; i++)
	{
		printf("MyCharArray[%d] = %c\n", i, MyCharArray[i]);
	}

	printf("Which array element do you want to see? ");
	scanf("%d", &Choice);
	printf("The value of array element %d is %c\n",
		 Choice, MyCharArray[Choice]);

	for (i = 0; i < 10; i++)
	{
		MyCharArray[i] = MyCharArray[i] | 32;
		printf("MyCharArray[%d] = %c\n", i, MyCharArray[i]);
	}

	return 0;
}
```

```
MyCharArray[0] = A
MyCharArray[1] = B
MyCharArray[2] = C
MyCharArray[3] = D
MyCharArray[4] = E
MyCharArray[5] = F
MyCharArray[6] = G
MyCharArray[7] = H
MyCharArray[8] = I
MyCharArray[9] = J
Which array element do you want to see? 5
The value of array element 5 is F
MyCharArray[0] = a
MyCharArray[1] = b
MyCharArray[2] = c
MyCharArray[3] = d
MyCharArray[4] = e
MyCharArray[5] = f
MyCharArray[6] = g
MyCharArray[7] = h
MyCharArray[8] = i
MyCharArray[9] = j
```

```
char MyCharArray[10] = {'A','B','C','D','E','F','G','H','I','J'};

MyCharArray[i] = MyCharArray[i] | 32;
```



```
char MyCharArray[10] = {'a','b','c','d','e','f','g','h','i','j'};

MyCharArray[i] = MyCharArray[i] ^ 32;
```



Array Initialization

```
Breakpoint 1, main () at array3Demo.c:9
9           char MyCharArray[10] = {'A','B','C','D','E','F','G','H','I','J'};
(gdb) p MyCharArray
$1 = "\340\005@\000\000\000\000\000\000"
(gdb) step
11              for (i = 0; i < 10; i++)
(gdb) p MyCharArray
$2 = "ABCDEFGHIJ"
```

```
Breakpoint 1, main () at array4Demo.c:9
9           char MyCharArray[] = {"ABCDEFGHIJ"};
(gdb) p MyCharArray
$3 = "\340\005@\000\000\000\000\000\000\000"
(gdb) step
11              for (i = 0; i < 10; i++)
(gdb) p MyCharArray
$4 = "ABCDEFGHIJ"
```

array3Demo.c
array4Demo.c

Array Initialization

Method 2 - in the executable code

```
for (i = 0; i < 10; i++)
{
    printf("Enter value for MyArray[%d] ", i);
    scanf("%d", &MyArray[i]);
}
```

```c
int main(void)
{
  int i;
  int MyArray[10];
  int Choice;

  for (i = 0; i < 10; i++)
  {
    printf("Enter value for MyArray[%d] ", i);
    scanf("%d", &MyArray[i]);
  }

  for (i = 0; i < 10; i++)
  {
    printf("MyArray[%d] = %d\n", i, MyArray[i]);
  }

  printf("\nEnter array element to display? ");
  scanf("%d", &Choice);
  printf("\nArray element %d is %d\n",
         Choice, MyArray[Choice]);

  return 0;
}
```

```
Enter value for MyArray[0] 4
Enter value for MyArray[1] 5
Enter value for MyArray[2] 22
Enter value for MyArray[3] 77
Enter value for MyArray[4] 11
Enter value for MyArray[5] 33
Enter value for MyArray[6] 98
Enter value for MyArray[7] 3
Enter value for MyArray[8] 56
Enter value for MyArray[9] 23
MyArray[0] = 4
MyArray[1] = 5
MyArray[2] = 22
MyArray[3] = 77
MyArray[4] = 11
MyArray[5] = 33
MyArray[6] = 98
MyArray[7] = 3
MyArray[8] = 56
MyArray[9] = 23

Enter array element to display? 6

Array element 6 is 98
```

array1Demo.c

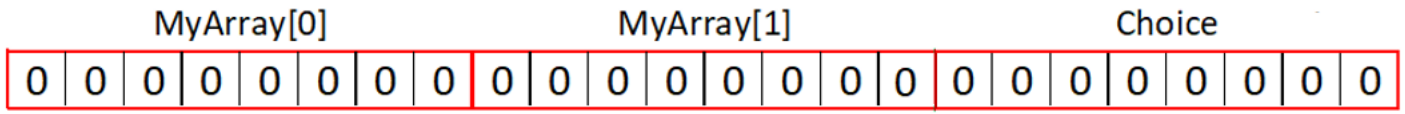
Array Initialization

```
int i;
int Choice = 0;
int MyIntArray[2] = {0,0};

printf("Choice is currently %d at %p\t", Choice, &Choice);

for (i = 0; i <= 2; i++)
{
   MyIntArray[i] = i;
   printf("MyIntArray[%d] = %d\t%p\n", i, MyIntArray[i], &MyIntArray[i]);
   getchar();
   printf("Choice is currently %d at %p\t", Choice, &Choice);
}
```

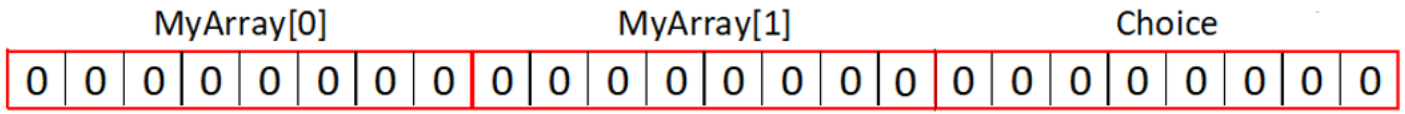
Array Initialization

```
Choice is currently 0 at 0x7fff58751b98 MyIntArray[0] = 0       0x7fff58751b90

Choice is currently 0 at 0x7fff58751b98 MyIntArray[1] = 1       0x7fff58751b94

Choice is currently 0 at 0x7fff58751b98 MyIntArray[2] = 2       0x7fff58751b98

Choice is currently 2 at 0x7fff58751b98
```

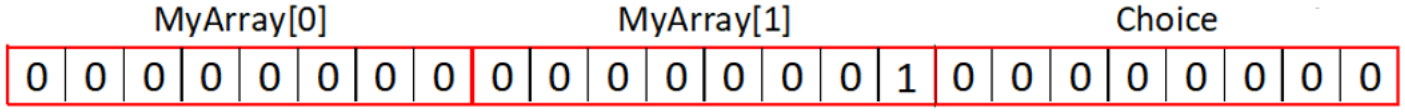
Initialization of Arrays – Out of Bounds



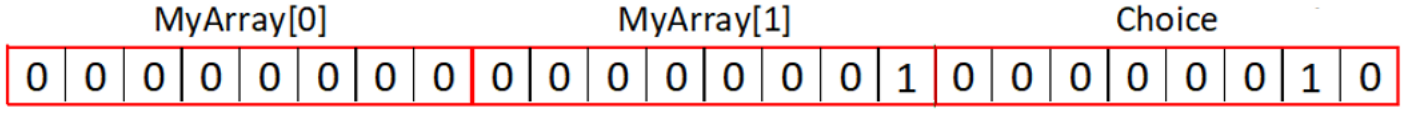
Assign 0 to MyArray[0]



Assign 1 to MyArray[1]



Assign 2 to MyArray[2]



Arrays as Parameters to Functions

Passing an array as a parameter to a function

- array must be declared in the function prototype

```
int MyFunction(int MyIntArray[]);
```

Formal parameter name is `MyIntArray` and it is of type `int`.

There is no indication of the number of elements in the parameter. This will not cause an error because the prototype does not cause any memory to be allocated (not a variable definition). Function only knows the address and element type of the array.

Arrays as Parameters to Functions

Passing an array as a parameter to a function

- array must be declared in the function header
- array may be accessed in the function code

```
void MyFunction(int MyIntArray[])
{
      printf("Element 0 of MyIntArray is %d", MyIntArray[0]);
}
```

Arrays as Parameters to Functions

Passing an array as a parameter to a function

- calling the function
  - when the name of the array is used without brackets, the array name is evaluated as the address of the array
  - passing the address of the array allows the function to access the elements
  - not possible to pass a copy of the array as a parameter

```
MyFunction(MyIntArray);
```

Arrays as Parameters to Functions

Can I return an array from a function via the return statement?

Can I create an array inside a function, put data in it and then use

```
return MyArray;
```

to return the filled up array back to `main()`?

*C:\Users\Donna\Desktop\UTA\Programs\CSE1320\passarray3Demo.c - Notepad++

File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?

MyFirstCProgram.c | win801ca.c | ICQ2.c | new 1 | array7Demo.c | passarray2Demo.c | passarray3Demo.c

```c
// Passing Array3 Demo
#include <stdio.h>


int main(void)
{

    return 0;
}
```

*C:\Users\Donna\Desktop\UTA\Programs\CSE1320\passarray3Demo.c - Notepad++

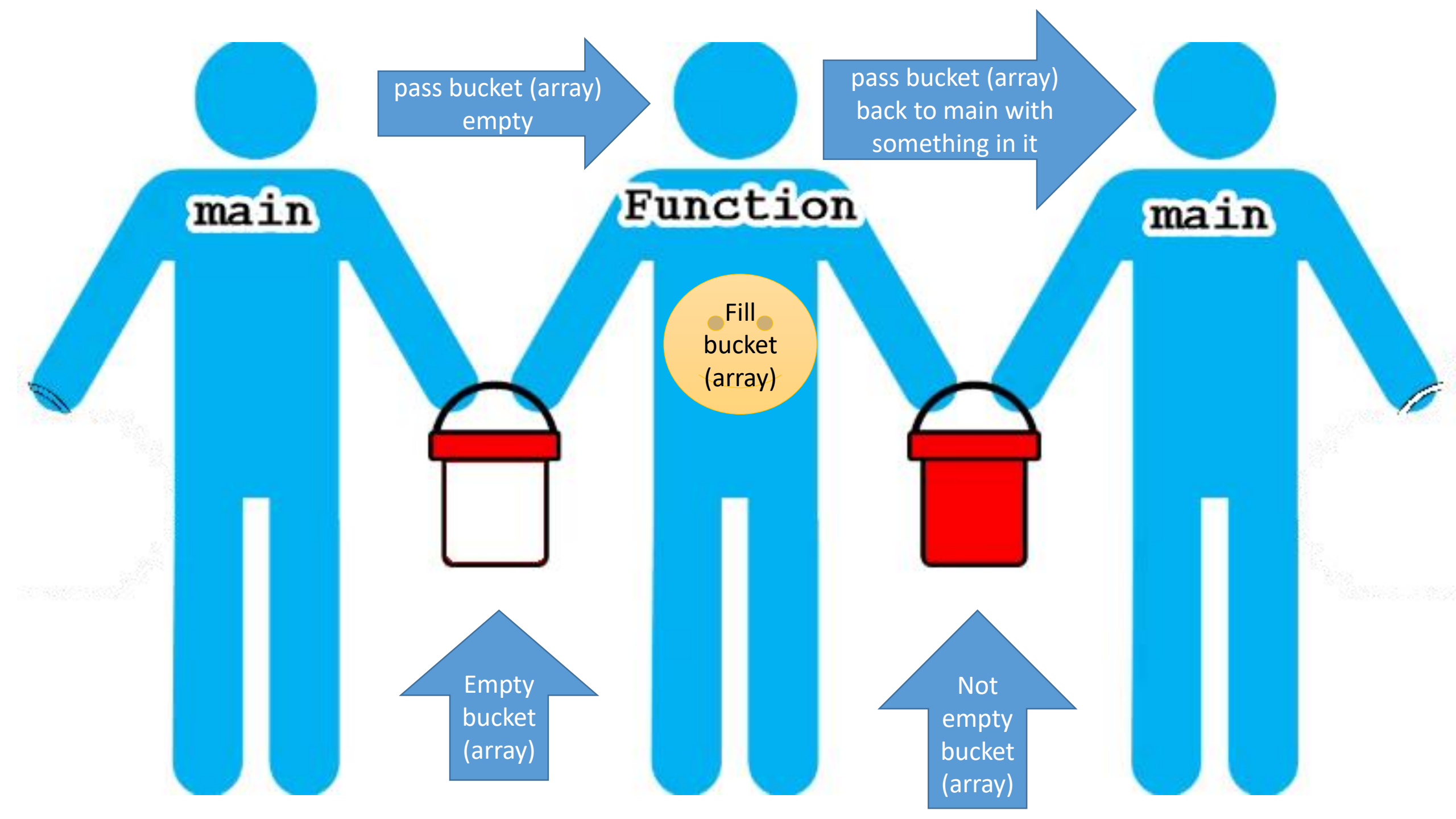
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?

MyFirstCProgram.c | win801ca.c | ICQ2.c | new 1 | array7Demo.c | passarray2Demo.c

```c
// Passing Array3 Demo
#include <stdio.h>

int CreateMonsterFunctionFunction(void);

int main(void)
{
    int Monster[5];

    Monster = CreateMonsterFunction();

    return 0;
}


int CreateMonsterFunction(void)
{
    int Monster[5] = {0};
    int i = 0;

    for (i = 0; i < 5; i++)
    {
        Monster[i] = i;
    }

    return Monster;
}
```

frenchdm@omega:~

```
[frenchdm@omega ~]$ gcc passarray3Demo.c
passarray3Demo.c: In function 'main':
passarray3Demo.c:10: error: incompatible types in assignment
passarray3Demo.c: In function 'CreateMonsterFunction':
passarray3Demo.c:25: warning: return makes integer from pointer without a cast
passarray3Demo.c:25: warning: function returns address of local variable
[frenchdm@omega ~]$
```


pass bucket (array) empty
pass bucket (array) back to main with something in it
main
Function
main
Fill bucket (array)
Empty bucket (array)
Not empty bucket (array)

```
int i;
int ElementsToEnter;
char MyCharArray[20] = {};

printf("How many characters do you want to enter? ");
scanf("%d", &ElementsToEnter);

getchar();

for (i = 0; i < ElementsToEnter; i++)
{
   printf("Enter character %d ", i);
   MyCharArray[i] = getchar();
   getchar();
}

printf("\n\nThe ASCII sum of the entered "
       "characters is %d\n\n",
       PrintArray(MyCharArray, i));
```

```
int PrintArray(char MyCharArray[], int ElementCount)
{
   int i, ASCIIsum = 0;

   for (i = 0; i < ElementCount; i++)
   {
      printf("MyCharArray[%d] = %c which is ASCII %d\n",
              i, MyCharArray[i], MyCharArray[i]);

      ASCIIsum += MyCharArray[i];
   }

   return ASCIIsum;
}
```

```
printf("\n\nThe ASCII sum of the entered "
       "characters is %d\n\n",
       PrintArray(MyCharArray, i));
```

```
How many characters do you want to enter? 4
Enter character 0 H
Enter character 1 E
Enter character 2 L
Enter character 3 P
MyCharArray[0] = H which is ASCII 72
MyCharArray[1] = E which is ASCII 69
MyCharArray[2] = L which is ASCII 76
MyCharArray[3] = P which is ASCII 80


The ASCII sum of the entered characters is 297
```

Passing Arrays

```
int main(void)
{
    int Lion[5];
    char Tiger[5];

    PassArrayFunction(Lion, Tiger);
    PrintArrayFunction(Tiger, Lion);

    return 0;
}
```

passarray2Demo.c

```
PassArrayFunction(Lion, Tiger);
```

```
void PassArrayFunction(int Grizzly[], char Polar[])
{
      int Bear;

      Grizzly[0] = UCHAR_MAX;
      Polar[0] = 'A';

      for (Bear = 1; Bear < 5; Bear++)
      {
            Grizzly[Bear] = Grizzly[Bear-1] >> 1;

            Polar[Bear] = Polar[Bear-1]+1;
      }

      return;
}
```

passarray2Demo.c

```
PrintArrayFunction(Tiger, Lion);
```

```
void PrintArrayFunction(char African[], int Asian[])
{
  int Elephant;

  for (Elephant = 0; Elephant < 5; Elephant++)
  {
    printf("African[%d] = %c\tAsian[%d] = %d\t\t",
           Elephant, African[Elephant],
           Elephant, Asian[Elephant]);

    printf("%d\n", (Asian[Elephant] & 16) ? 1 : 0);
  }
}
```

```
African[0] = A  Asian[0] = 255          1
African[1] = B  Asian[1] = 127          1
African[2] = C  Asian[2] = 63           1
African[3] = D  Asian[3] = 31           1
African[4] = E  Asian[4] = 15           0
```

passarray2Demo.c

? :

Known as

- Conditional operator
- inline if (iif)
- **ternary if**

```
if (condition)
{
   variable = expr1;
}
else
{
   variable = expr2;
}
```

```
variable = (condition) ? expr1 : expr2;
```

```
if (bit1 % 2)
{
   bit1 = 1;
}
else
{
   bit1 = 0;
}
```

```
bit1  = (bit1 % 2) ? 1 : 0;
```

? :

```
printf("Enter a number ");
scanf("%d", &MyNumber);

if (myNumber & 1)
{
   x = 1;
}
else
{
   y = 1;
}
```

Turn this into a ternary if

```
variable = (condition) ? expr1 : expr2;
```

```
printf("Enter a number ");
scanf("%d", &MyNumber);

if (myNumber & 1)
{
   x = 1;
}
else
{
   x = 0;
}
```

Turn this into a ternary if

```
x = (myNumber & 1) ? 1 : 0;
```