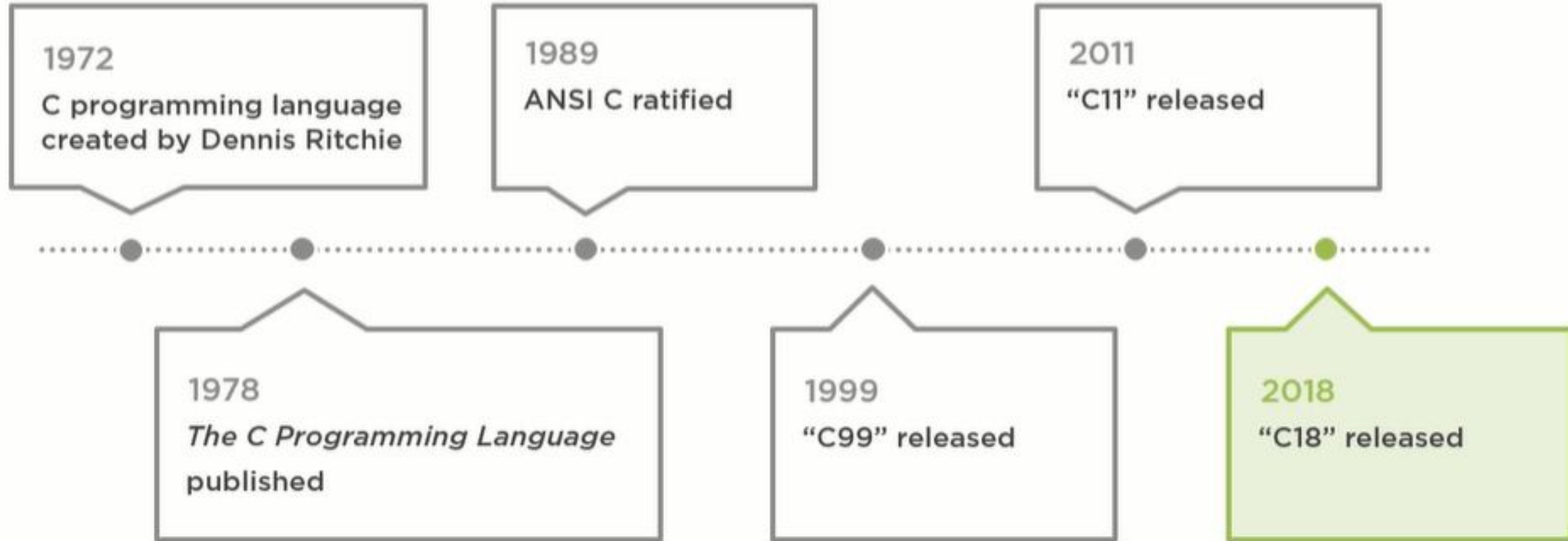
Evolution of C



General Purpose High Level Language

Ex: Java, C#

Current Infrastructure Specific

Not Portable

Supports Programming Constructs
Variable, Loops
Conditionals & more
+
Platform Portability

Source Code

Compiler

Runtime/Virtual Machine

C Source Code

C Compiler

Assembly Code

Machine Code

C Influenced New Languages

By C's SYNTAX or DESIGN or both

What type of Language is C?

Procedural Language

C++ : Object Oriented Language with CLASSes

Objective C : Object Oriented & Message Passing

C# : Object Oriented (Syntax influenced by C)

What

Languages look similar Syntactically? — PHP, Java, Javascript

Languages include standard Libraries? — Python, Ruby

Languages written in C? — Perl, Tcl, Lua

Compare C & C#

hello.c

```c
#include <stdio.h>
main()
{
  printf("hello, world\n");
}
```

HelloWorld.cs

```csharp
using System;
class HelloWorld
{
  static void Main()
  {
    Console.WriteLine("hello, world");
  }
}
```

Compare C & Java

hello.c

```
#include <stdio.h>
main()
{
  printf("hello, world\n");
}
```

HelloWorld.java

```
public class HelloWorld {
  public static void main(String []args) {
    System.out.println("hello, world");
  }
}
```

Advantages Using C

C has Access to MEMORY & is FAST

Modern Operating Systems

C - Designed Linux OS

C - Parts of MAC OS & Windows

Cross Platform & Fast

Development of Applications for Various Platforms (C Compilers)

Developed GIT - Best Source Control System (C Compilers) for many Platforms

Apps Run Faster than most Interpreted Languages

Advantages Using C

Works FAST

Good for Systems with Limited Resources

Programmers have Direct Access to the MEMORY

Widely Available C Compilers

C Code in 1000s of Micro Controllers - One Language for any micro device (IoT - Airplane to Washing M/c, Rasberry Pi, Arduino)

en.wikipedia.org/wiki/TI_MSP430

Article Talk Read Edit View hist

TI MSP430

From Wikipedia, the free encyclopedia

This article has multiple issues. Please help improve it or discuss these issues on the talk page. (
when to remove these template messages)

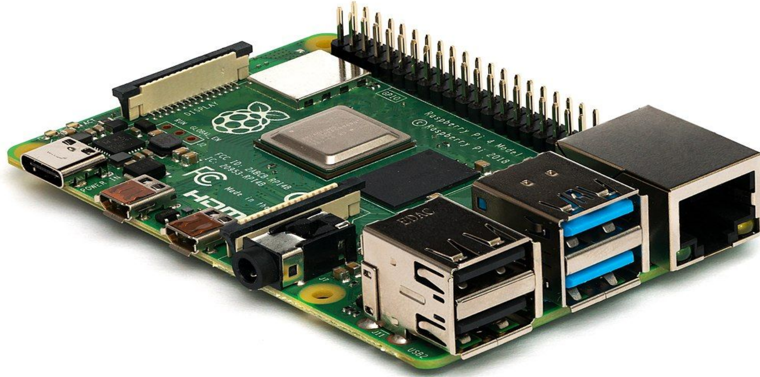
- This article **contains content that is written like an advertisement**. *(January 2017)*
- This article's **use of external links may not follow Wikipedia's policies or guidelines**. *(Decem*

The **MSP430** is a mixed-signal microcontroller family from Texas Instruments, first introduced on 14 February 1992.[1] Built around a 16-bit CPU, the MSP430 is designed for low cost and, specifically, low power consumption[2] embedded applications.

Contents [hide]

1 Applications
1.1 Memory limitations
2 MSP430 generations
2.1 MSP430x1xx series
2.2 MSP430F2xx series
2.3 MSP430G2xx series
2.4 MSP430x3xx series
2.5 MSP430x4xx series
2.6 MSP430x5xx series
2.7 MSP430x6xx series
2.8 RF SoC (CC430) series

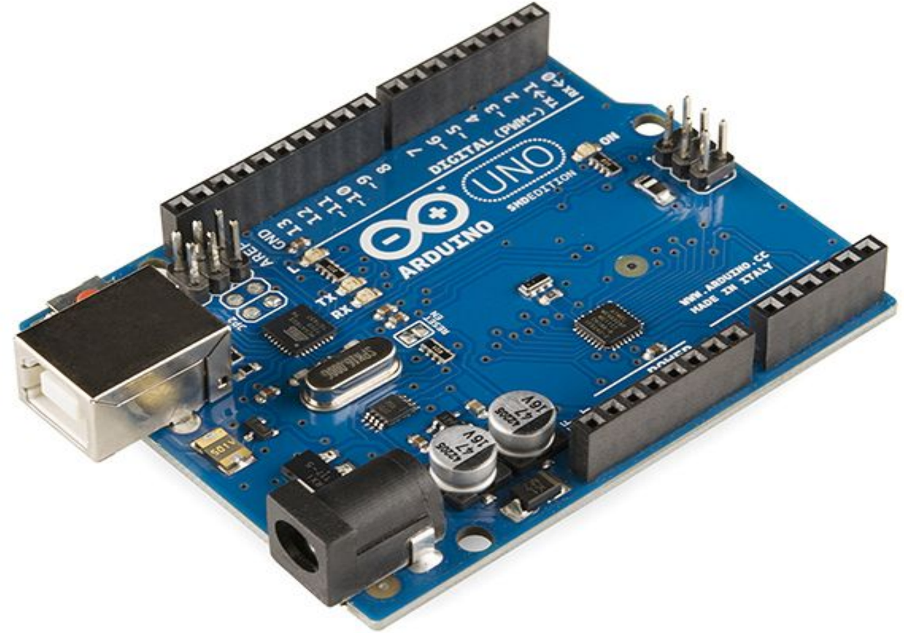
Rasberry Pi



Microprocessor
Clock speed: 1.2 GHz

Code with Python

Arduino



Microcontroller (Not full-computer)
Clock speed: 16 MHz

Interface Sensors, LEDs & Motors

Programming in C

Need 2 things:

Compiler : GCC (GNU Compiler Collection)

Editor: Atom, Visual Studio Code, Turbo C

Data Types in C

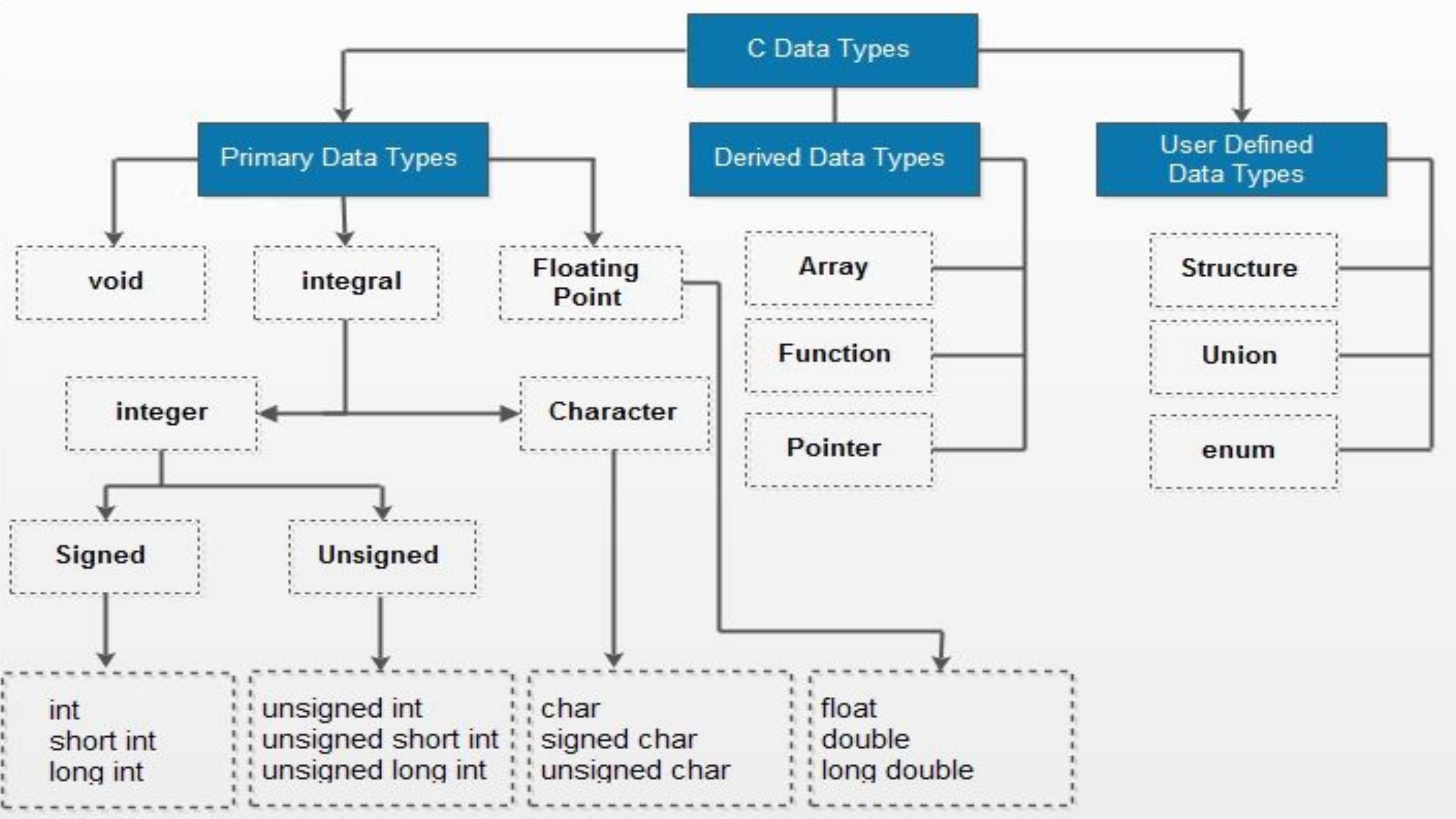
Basic data types

Derived types

Using Type Casting and Type Qualifiers

Using time, date, and localization

C program using Basic and Derived types


C Data Types
Primary Data Types
Derived Data Types
User Defined
Data Types
void
integral
Floating
Point
Array
Function
Pointer
Structure
Union
enum
integer
Character
Signed
Unsigned
int
short int
long int
unsigned int
unsigned short int
unsigned long int
char
signed char
unsigned char
float
double
long double

Data Types in C

Integer
- signed and unsigned
- short and long
- Fixed-width integer types
- #include <stdint.h>
- #include <inttypes.h> (added in C99)

Char
- char, unsigned char, signed char
- A = 65, Z = 90
- Relies on the ASCII table

Boolean
- Added in the C99 standard
- #include <stdbool.h>

Enumeration

Bit
- unsigned int age : 7;

Void

ASCII Table

Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char
0	0	0		32	20	40	[space]	64	40	100	@	96	60	140	`
1	1	1		33	21	41	!	65	41	101	A	97	61	141	a
2	2	2		34	22	42	"	66	42	102	B	98	62	142	b
3	3	3		35	23	43	#	67	43	103	C	99	63	143	c
4	4	4		36	24	44	$	68	44	104	D	100	64	144	d
5	5	5		37	25	45	%	69	45	105	E	101	65	145	e
6	6	6		38	26	46	&	70	46	106	F	102	66	146	f
7	7	7		39	27	47	'	71	47	107	G	103	67	147	g
8	8	10		40	28	50	(	72	48	110	H	104	68	150	h
9	9	11		41	29	51	)	73	49	111	I	105	69	151	i
10	A	12		42	2A	52	*	74	4A	112	J	106	6A	152	j
11	B	13		43	2B	53	+	75	4B	113	K	107	6B	153	k
12	C	14		44	2C	54	,	76	4C	114	L	108	6C	154	l
13	D	15		45	2D	55	-	77	4D	115	M	109	6D	155	m
14	E	16		46	2E	56	.	78	4E	116	N	110	6E	156	n
15	F	17		47	2F	57	/	79	4F	117	O	111	6F	157	o
16	10	20		48	30	60	0	80	50	120	P	112	70	160	p
17	11	21		49	31	61	1	81	51	121	Q	113	71	161	q
18	12	22		50	32	62	2	82	52	122	R	114	72	162	r
19	13	23		51	33	63	3	83	53	123	S	115	73	163	s
20	14	24		52	34	64	4	84	54	124	T	116	74	164	t
21	15	25		53	35	65	5	85	55	125	U	117	75	165	u
22	16	26		54	36	66	6	86	56	126	V	118	76	166	v
23	17	27		55	37	67	7	87	57	127	W	119	77	167	w
24	18	30		56	38	70	8	88	58	130	X	120	78	170	x
25	19	31		57	39	71	9	89	59	131	Y	121	79	171	y
26	1A	32		58	3A	72	:	90	5A	132	Z	122	7A	172	z
27	1B	33		59	3B	73	;	91	5B	133	[	123	7B	173	{
28	1C	34		60	3C	74	<	92	5C	134	\	124	7C	174	|
29	1D	35		61	3D	75	=	93	5D	135	]	125	7D	175	}
30	1E	36		62	3E	76	>	94	5E	136	^	126	7E	176	~
31	1F	37		63	3F	77	?	95	5F	137	_	127	7F	177	

© w3resource.com

Data Types in C

Type	32 bit size	64 bit size
char	1 byte	1 byte
short	2 bytes	2 bytes
int	4 bytes	4 bytes
long	4 bytes	8 bytes
Long long	8 bytes	8 bytes
float	4 bytes	4 bytes
double	8 bytes	8 bytes
Long double	16 bytes	16 bytes

Data type	Size (in bytes)	Range	Format Specifier
short int	2	-32,768 to 32,767	%hd
unsigned short int	2	0 to 65,535	%hu
unsigned int	4	0 to 4,294,967,295	%u
int	4	-2,147,483,648 to 2,147,483,647	%d
long int	4	-2,147,483,648 to 2,147,483,647	%ld
unsigned long int	4	0 to 4,294,967,295	%lu
long long int	8	-(2^63) to (2^63)-1	%lld
unisgned long long int	8	0 to 18,446,744,073,709,551,615	%llu
signed char	1	-128 to +127	%c
unsigned char	1	0 to 255	%c
float	4		%f
double	8		%lf
long double	16		%LF

Predefined - Data Types in C

Specifier	Common Equivalent	Signing	Bits	Bytes	Minimum Value	Maximum Value
int8_t	signed char	Signed	8	1	-128	127
uint8_t	unsigned char	Unsigned	8	1	0	255
int16_t	short	Signed	16	2	-32,768	32,767
uint16_t	unsigned short	Unsigned	16	2	0	65,535
int32_t	int	Signed	32	4	-2,147,483,648	2,147,483,647
uint32_t	unsigned int	Unsigned	32	4	0	4,294,967,295
int64_t	long long	Signed	64	8	-9,223,372,036,854,775,808	9,223,372,036,854,775,807
uint64_t	unsigned long long	Unsigned	64	8	0	18,446,744,073,709,551,615

Predefined - Data Types in C

Number of bits	Signed	Unsigned
8	int8_t $-2^7 \le x \le 2^7 - 1$ $-128 \le x \le 127$	uint8_t $0 \le x \le 2^8 - 1$ $0 \le x \le 255$
16	int16_t $-2^{15} \le x \le 2^{15} - 1$ $-32{,}768 \le x \le 32{,}767$	uint16_t $0 \le x \le 2^{16} - 1$ $0 \le x \le 65{,}535$
32	int32_t $-2^{31} \le x \le 2^{31} - 1$ $-2.15 \times 10^9 \le x \le 2.15 \times 10^9$	uint32_t $0 \le x \le 2^{32} - 1$ $0 \le x \le 4.3 \times 10^9$
64	int64_t $-2^{63} \le x \le 2^{63} - 1$ $-9.22 \times 10^{18} \le x \le 9.22 \times 10^{18}$	uint64_t $0 \le x \le 2^{64} - 1$ $0 \le x \le 18.4 \times 10^{18}$

Printing & Reading Integer Types

Format	Use with
%d	decimal (signed int, short, char)
%c	char (in character format)
%u	decimal (unsigned int, unsigned short, unsigned char)
%x	hexadecimal (int, short, char)
%o	octal (int, short, char)
%ld	decimal (signed long)
%lu or %lx or %lo	as above but for longs

```
int num = 10;
printf("%d", num);
scanf("%d", &num);
```

Fixed Width Integer Types

Enables integer types with specific sizes
For program portability & same behavior in different systems
Header files: <inttypes.h> and <stdint.h> header.

Exact-width	Minimum-width	Fastest minimum-width	Greatest-width
int8_t	int_least8_t	int_fast8_t	intmax_t
int16_t	int_least16_t	int_fast16_t	uintmax_t
int32_t	int_least32_t	int_fast32_t	
int64_t	int_least64_t	int_fast64_t	
uint8_t	uint_least8_t	uint_fast8_t	
uint16_t	uint_least16_t	uint_fast16_t	
uint32_t	uint_least32_t	uint_fast32_t	
uint64_t	uint_least64_t	uint_fast64_t	

Floating Types

Float	Double	Long Double
Declaration • float x = 3.0;	**Declaration** • double x = 3.0;	**Declaration** • long double x = 3.0;
Format • printf(“%f”, x); • scanf(“%f”, &x);	**Format** • printf(“%f”, x); • scanf(“%f”, &x);	**Format** • printf(“%lf”, x); • scanf(“%lf”, &x);

Complex Types

$x + yi \qquad (i^2 = -1)$

$3.0 + 4.0i$ = 3.0 + 4.0 * _Complex_I

header: #include<complex.h>

Standard type	Microsoft type
float complex or float _Complex	_Fcomplex
double complex or double _Complex	_Dcomplex
long double complex or long double _Complex	_Lcomplex

Derived Data Types - Arrays

Collection of same type values
Represented by single name
Index to select individual members

Array of Dimension 100, Index 0-99
Multi-dimensional array with N Dimensions

int arr[5][4]
int arr[100][5][3]

Derived Data Types - Pointers

Points to address of another variable;
Declare a pointer of corresponding data type before assigning an address.

```c
#include <stdio.h>

int main () {

    float var1 = 5;
    float *p = &var1;

    printf("%f \n", p);

    printf("%f \n", *p);

}
```

Address of pointer p

Value of pointer p

Derived Data Types - Function

```c
#include <stdio.h>
#include <stdint.h>

int calculateSum(int x, int y);

int main(){
    int calc = calculateSum(1,2);
    printf("%d \n", calc);
}


int calculateSum(int x, int y){
    return x+y;
}
```

User Defined Data Types - Structure

```
struct address
{
   char name[100];
   char street[100];
   char city[50];
   char state[20];
   int pin;
};
```

Collection of members (variables) of possibly different types into a single user defined type

```
int main(){

      struct address address1 = {“My House”, “Street 1”, “Lisbon”, “Lisbon”, 0}

      addr1.name = “My House 2”;

      printf(“%s \n”, addr1.name);
      printf(“%s \n”, addr1.street);
}
```

Output: My House 2

Output: Street 1

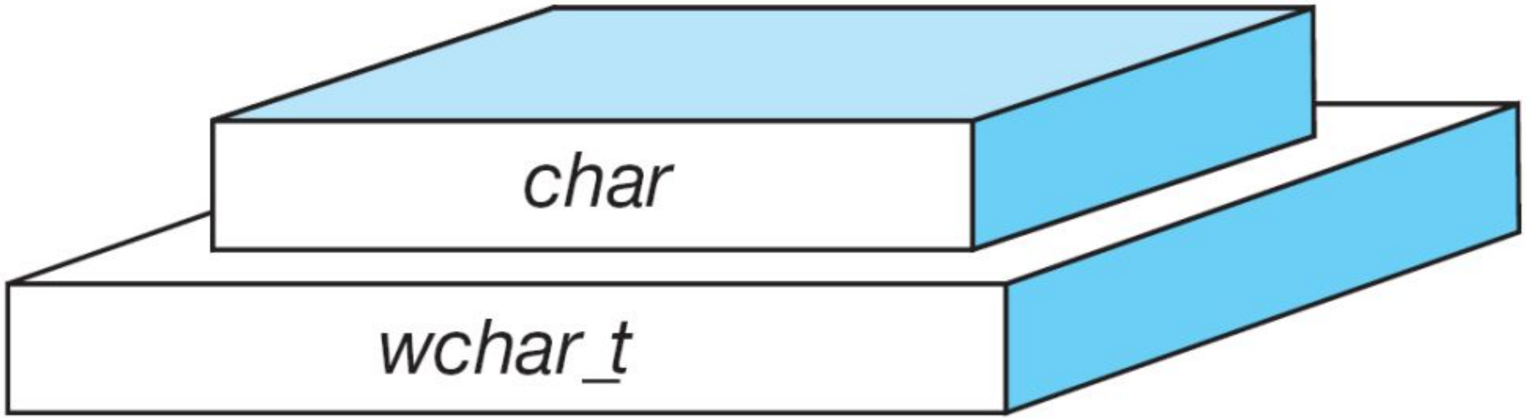
User Defined Data Types - Union

Collection of members (variables) of possibly different types into a single user defined type; **Stored in same memory location**.

```
union address
{
   char name[50];
   char street[100];
   char city[50];
   char state[20];
   int pin;
};

int main(){
   union address address1 = {"My House"};

   printf("%s \n", addr1.name);            <----   Output: My House

   strcpy(addr1.street, "My Street");
   printf("%s \n", addr1.street);          <----   Output: My Street
   printf("%s \n", addr1.name);            <----   Output: My Street

}
```

```
enum Type { Number, Text, Real};

struct question{
      char questionText[500];

      Type answerType;

      union answer
      {
         int answer_number;
         char answer_text[250];
         float answer_real;
      };

};
```

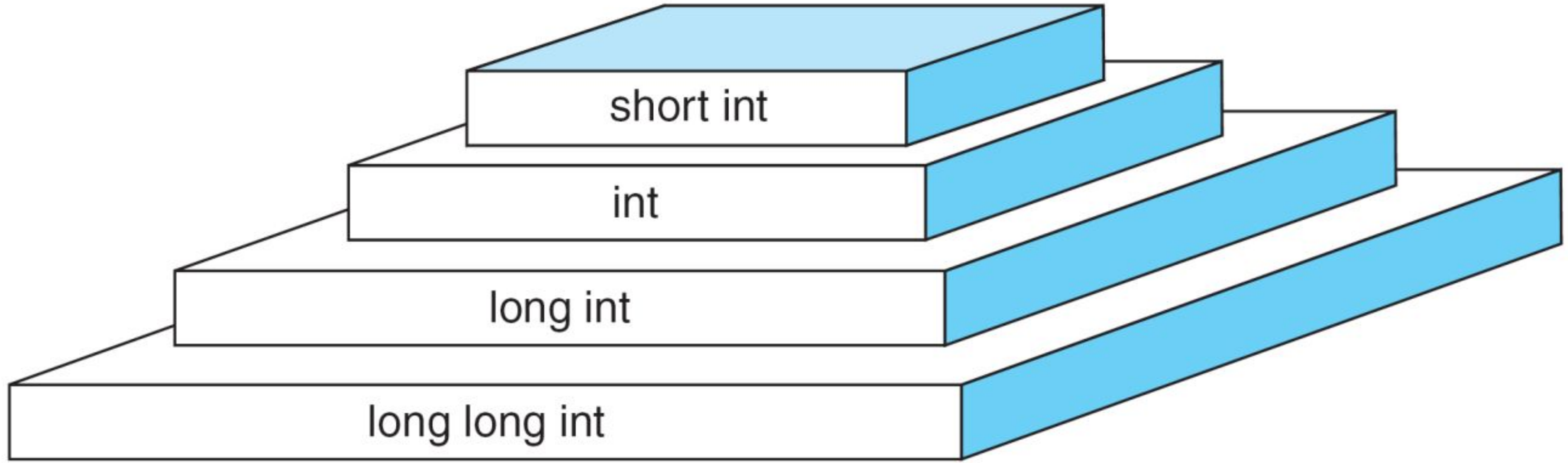
Compare Data Types - char



wchar_t is a wide character: The increased datatype size allows for the use of larger coded character sets.

Width is compiler specific (not portable).

Compare Data Types - int

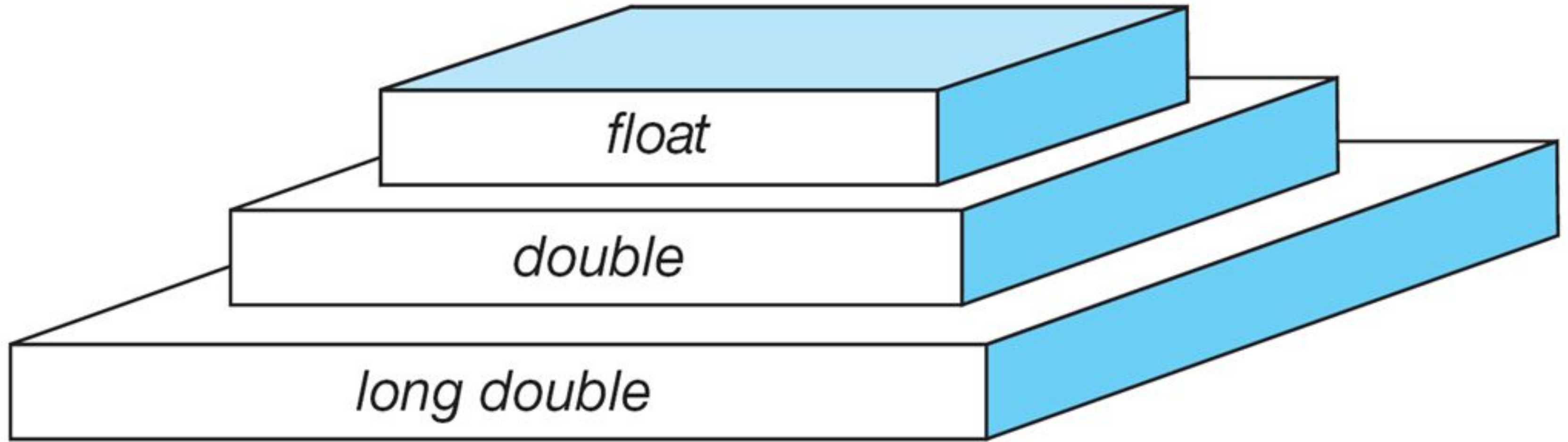


sizeof (short) ≤ sizeof (int) ≤ sizeof (long) ≤ sizeof (long long)

Compare Data Types - signed integers

Type	Byte Size	Minimum Value	Maximum Value
short int	2	–32,768	32,767
int	4	–2,147,483,648	2,147,483,647
long int	4	–2,147,483,648	2,147,483,647
long long int	8	–9,223,372,036,854,775,807	9,223,372,036,854,775,806

Compare Data Types - floating point



sizeof (float) ≤ sizeof (double) ≤ sizeof (long double)

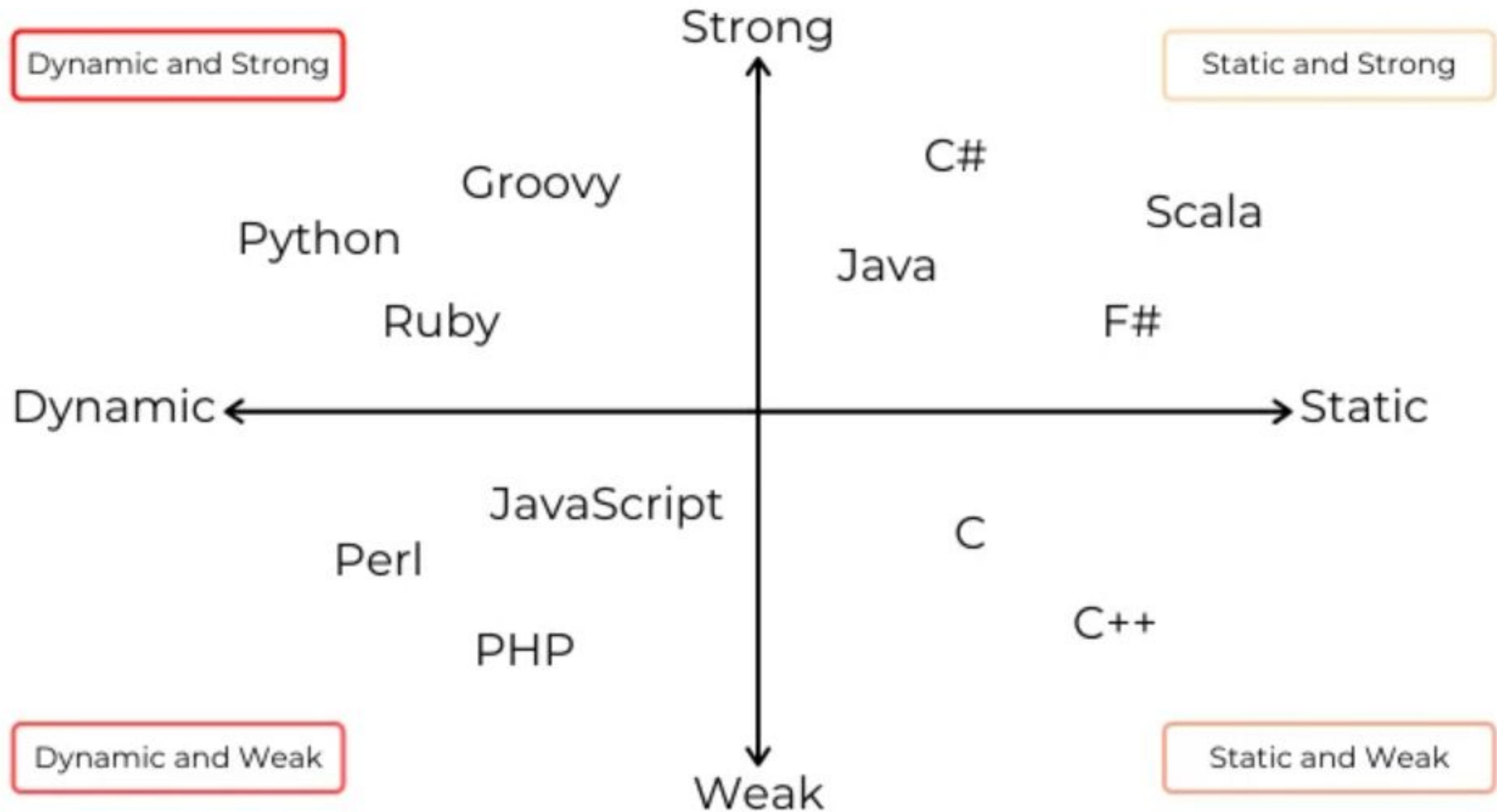
Summary Data Types

Category	Type	C Implementation
Void	Void	*void*
Integral	Boolean	*bool*
	Character	*char, wchar_t*
	Integer	*short int, int, long int, long long int*
Floating-Point	Real	*float, double, long double*
	Imaginary	*float imaginary, double imaginary, long double imaginary*
	Complex	*float complex, double complex, long double complex*

STRONG vs WEAK

STATIC vs DYNAMIC

Languages: Static vs Dynamic, Weak vs Strong



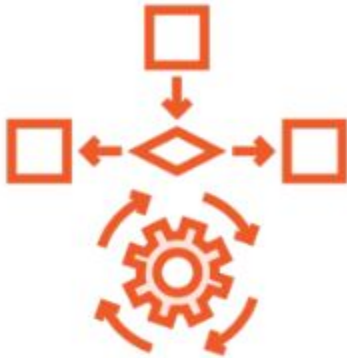
Static vs. Dynamic Declaration of data types.
Static typed languages require explicit definition of variable, parameter, return value.
Dynamic languages can infer, or at least try to guess, the type that we're using.

Strong vs. Weak defines - Operations between data types.
Strongly typed languages will not allow you to add a float to an integer, without you converting it first, even though they are both numbers.
Weak language will try its best to accomodate what the programmer is asking and perform these operations.

No best option in choosing static vs. dynamic or strong vs. weak

Type Casting / Conversion

Process of converting one data type into another



Implicit Type Conversion

Type conversion is performed automatically, by the compiler, without the intervention of the programmer.

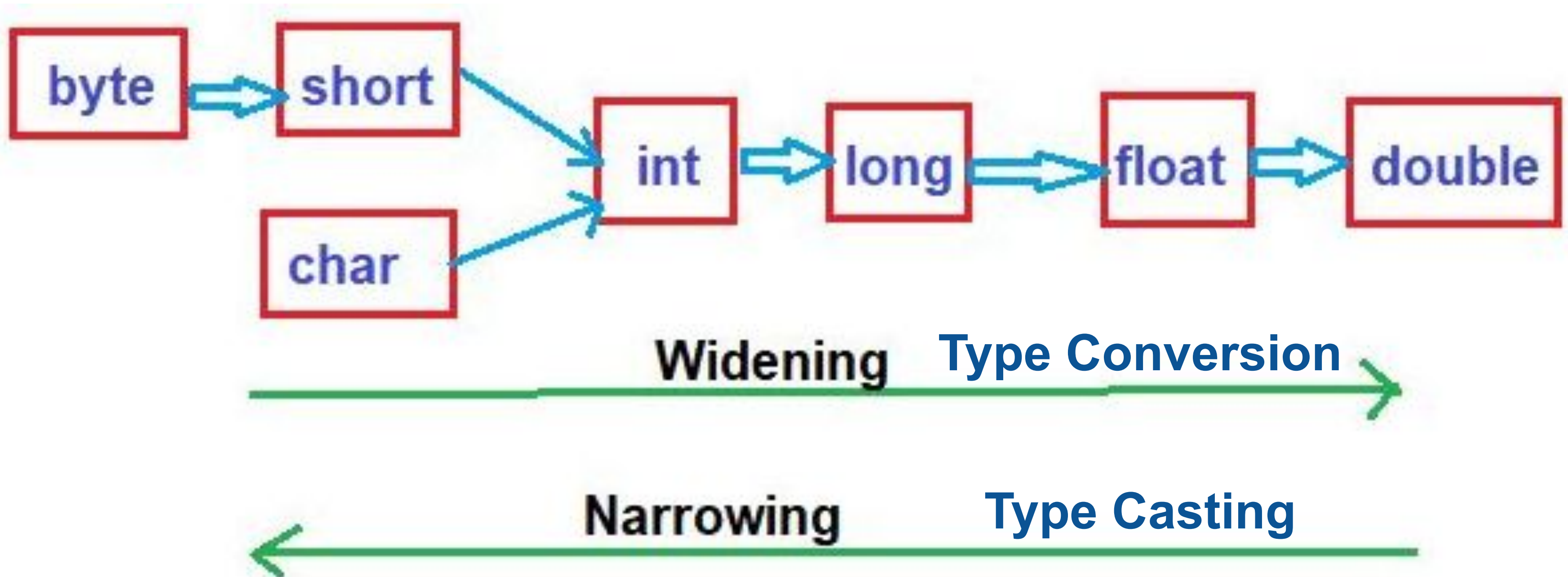
Explicit Type Conversion

Type conversion is manually performed, by the programmer, to convert a from one type to another explicit type, using the cast operator.

F

Type Casting / Conversion

Process of converting one data type into another



Type Casting / Conversion

Implicit: Automatic conversion by compiler

Explicit: Manually by type casting operator ( )

Implicit Type Casting / Conversion

```c
// Implicit Type Conversion
#include<stdio.h>
int main() {
// create a double variable
double value = 7520.17;
printf("Double Value: %.2lf\n", value);

// convert double value to integer
int number = value;
printf("Integer Value: %d", number);

return 0;
}
```

```c
// Implicit Type Conversion
#include<stdio.h>
int main() {
// character variable
char letter = 'A';
printf("Character Value: %c\n", letter);

//Assign char value to int variable
int number = letter;
printf("Integer Value: %d", number);

return 0;
}
```

Implicit Type Casting / Conversion

```c
// Implicit Type Conversion of numeric types
#include <stdbool.h>
#include <stdio.h>
int main(void)
{// Variables Declarations & Initializations
 bool b = true;
 char c = 'A';
 float f = 100.5;
 int i = 100;
 short s = 77;
 //Statements with implicit conversion
 printf("Implicit conversion\n");
 printf("bool + char is char:   %c\n", b + c);
 printf("int * short is int:    %d\n", i * s);
 printf("float * char is float:%f\n", f * c);
 // bool promoted to char
 c = c + b;
 // char promoted to float
 f = f + c;

 b = false;
 // float demoted to bool
 b = -f;
 printf("After promotion / demotion: \n");
 printf("char + true:  %c\n", c);
 printf("float + char: %f\n", f);
 printf("bool = -float:%d\n", b);
 return 0;
          }
```

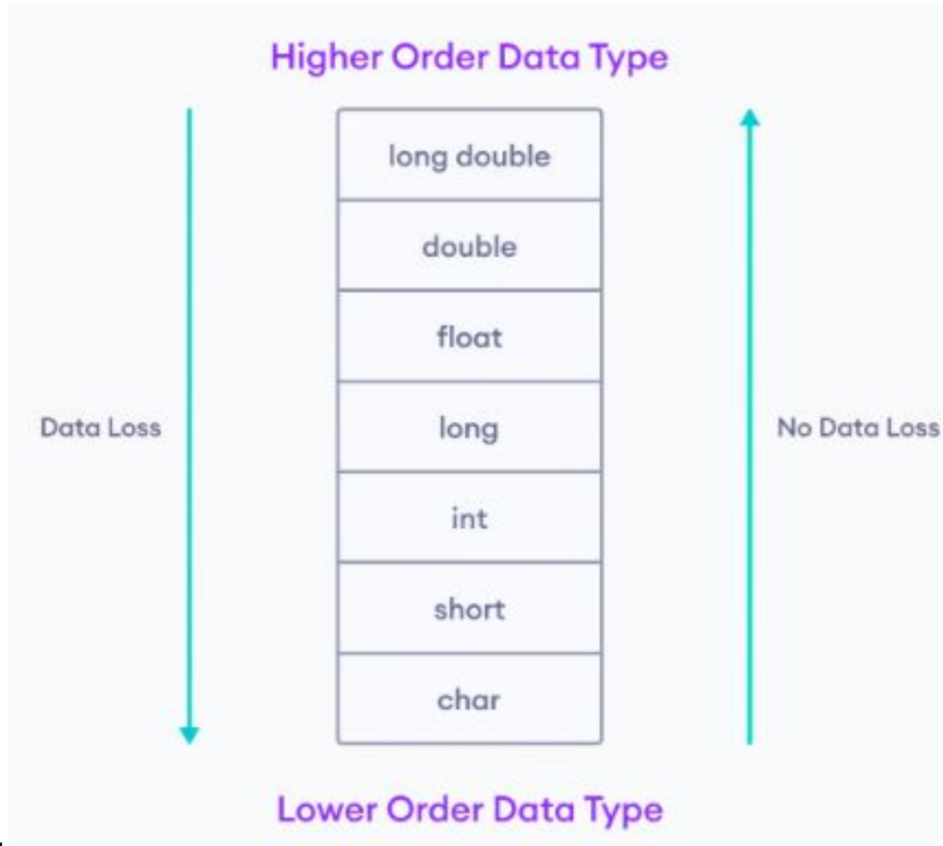
Explicit Type Casting / Conversion

```c
//Explicit type conversion
#include<stdio.h>
int main() {
// create an integer variable
int numb = 97;
printf("Integer Value: %d\n", numb);

// (char) converts number to char
char alpha = (char) numb;
printf("Character Value: %c", alpha);
return 0;
}
```

```c
//Explicit type conversion
#include<stdio.h>
int main() {
// create an integer variable
int numb = 35;
printf("Integer Value: %d\n", numb);

// explicit type conversion
double value = (double) numb;
printf("Double Value: %.2lf", value);
return 0;
}
```

Type Casting / Conversion

Possible data loss during type casting (demoting)



Data loss

long double to double type

No data loss

char is converted to int

LATER!

Pointer type conversion

Storage Classes in C

Storage Class	Keyword	Lifetime	Visibility	Initial Value
Automatic	auto	Function Block	Local	Garbage
External	extern	Whole Program	Global	Zero
Static	static	Whole Program	Local	Zero
Register	register	Function Block	Local	Garbage

Storage Classes: auto, static

```c
//Storage classes auto & static
#include<stdio.h>
void display_count()
{    auto int k = 0;       //prints 1 1 1
     //static int k = 0; //prints 1 2 3
     k = k + 1;
     printf("\n %d",k);
}
void main() {
int j;
for(int j=1;j<=3;j++)
  display_count();
getch();
}
```

```c
//Storage classes auto
#include<stdio.h>
#include<conio.h>
void main() {
auto int a=50;
{ //local block
  auto int a = 10;
  printf(" %d",a); //prints 10
}
printf(" %d",a); //prints 50
getch();
}
```

Storage Classes: register

```
//Storage class register
#include<stdio.h>
#include<conio.h>
void main() {
//memory in CPU register
//faster execution but loads CPU
register int x=50, y=10, result;
result = x/y;
printf(" %d",result);
getch();
}
```

Storage Classes: extern

```c
//Storage class extern parent;
//a header file
#include<stdio.h>
int pass_marks = 65;

void verify(int m)
{
  if (m>=pass_marks)
    printf("Passed! Marks %d > Pass %d",m,pass_marks);
  else
    printf("Failed! Marks %d < Pass %d",m,pass_marks);
}
```

Header File

```c
//Storage class extern child
#include<stdio.h>
#include "E:\Courses\C\C-PS\c…parent.c"
extern void verify(int m);
int main(){
    extern int pass_marks;
    int marks = 70;
    verify(marks);
return 0;
}
```

Program

Volatile type in C

Declaration

```
volatile int flag;

int volatile flag;
```

```
int main (){
   int flag=0;
   flag++;
}
```

CPU Optimization
Mem to Registers

```
int main (){
   volatile int flag=0;
   flag++;
}
```

NO CPU Optimization
NO Mem to Registers

Applications:

Global Variables

Multi Threaded Apps

Interrupt routines

Preprocessor Directives in C

Directive	Function
#define	Defines a **Macro** Substitution
#undef	Undefines a **Macro**
#include	Includes a File in the Source Program
#ifdef	Tests for a **Macro** Definition
#endif	Specifies the end of #if
#ifndef	Checks whether a **Macro** is defined or not
#if	Checks a Compile Time Condition
#else	Specifies alternatives when #if Test Fails

Preprocessor Directives in C

Directive	Function
#include	Includes a header file in the source program
#define	Defines Macro substitution
#undef	Undefines Macro
#ifdef	Tests for a Macro definition
#ifndef	Checks whether a Macro is defined or not
#if	Checks a compile time condition
#elif	Checks another compile time condition
#else	Specifies alternative when #if condition fails
#endif	Specifies end of #if

Demo Programs - Preprocessor Directives in C

Operators in C

<table>
<tr><th></th><th>Operators</th><th>Type</th></tr>
<tr><td>Unary operator</td><td>++ , --</td><td>Unary operator</td></tr>
<tr><td rowspan="5">Binary operator</td><td>+, -, *, /, %</td><td>Arithmetic operator</td></tr>
<tr><td><, <=, >, >=, ==, !=</td><td>Relational operator</td></tr>
<tr><td>&&, | |, !</td><td>Logical operator</td></tr>
<tr><td>&, |, <<, >>, -, ^</td><td>Bitwise operator</td></tr>
<tr><td>=, +=, -=, *=, %=</td><td>Assignment operator</td></tr>
<tr><td>Ternary operator</td><td>?:</td><td>Ternary or conditional operator</td></tr>
</table>

Other Operators in C

`.` individual members of struct & union

`->` pointer to an object in C++

`*` pointer to a variable

`&` returns address of a variable

`()` cast operator - converts one data type to another

`sizeof` finds size in bytes

Operators in C

Demo `c_operator1.c`

Q&As on Operators in C

What is the difference between prefix and postfix operators in C?

Prefix – first adds or subtracts 1 and then assigns the resultant value to the variable. ++a and - -a.

Postfix – first assigns the value to the variable, then adds or subtracts 1, and then assigns the resultant value.

Q&As on Operators in C

```
#include<stdio.h>
Int main()  {
     int x;
     x = 2;
          printf( “%d\n”, x );
          printf( “%d\n”, x++ );
          printf( “%d\n\n”, x );
     x = 2;
          printf( “%d\n”, x );
          printf( “%d\n”, ++x );
          printf( “%d\n”, x );
return 0;
}
```

A. 2 2 3 2 3 3

B. 2 3 3 2 2 3

C. 3 2 3 3 3 2

D. 3 3 2 3 2 2

Q&As on Operators in C

```
#include<stdio.h>
Int main()  {
    int x;
    x = 2;
        printf( “%d\n”, x );
        printf( “%d\n”, x++ );
        printf( “%d\n\n”, x );
    x = 2;
        printf( “%d\n”, x );
        printf( “%d\n”, ++x );
        printf( “%d\n”, x );
return 0;
}
```

A. 2 2 3 2 3 3

B. 2 3 3 2 2 3

C. 3 2 3 3 3 2

D. 3 3 2 3 2 2

Q&As on Operators in C

```
int a = 10;
int b = a++%5;
```

What will be the value of a and b after we execute the code?

A. a is 10, and b is 1.

B. a is 10, and b is 0.

C. a is 11, and b is 0.

D. a is 11, and b is 1.

Q&As on Operators in C

int a = 10;

int b = a++%5;

What will be the value of a and b after we execute the code?

A. a is 10, and b is 1.

B. a is 10, and b is 0.

C. a is 11, and b is 0.

D. a is 11, and b is 1.

Q&As on Operators in C

```
int a = 10;
int b = ++a%5;
```

What will be the value of a and b after we execute the code?

A. a is 10, and b is 1.

B. a is 10, and b is 0.

C. a is 11, and b is 0.

D. a is 11, and b is 1.

Q&As on Operators in C

```
int a = 10;
int b = ++a%5;
```

What will be the value of a and b after we execute the code?

A. a is 10, and b is 1.

B. a is 10, and b is 0.

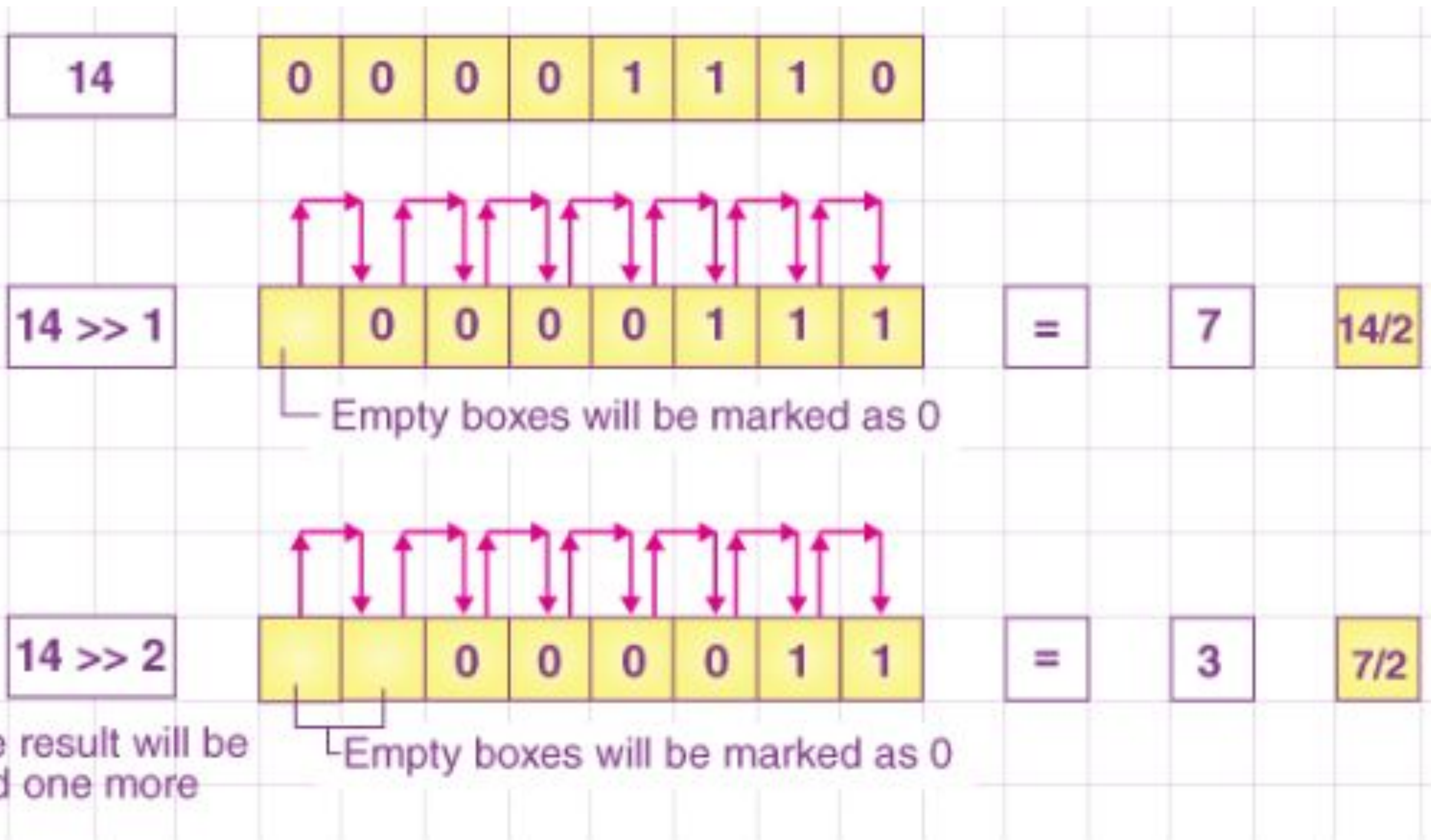
C. a is 11, and b is 0.

D. a is 11, and b is 1.

Truth Table - Bitwise Operators

<table>
<tr><th>a</th><th>b</th><th>a & b</th><th>a | b</th><th>a ^ b</th><th>~a</th></tr>
<tr><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>1</td><td>1</td><td>1</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td></tr>
</table>

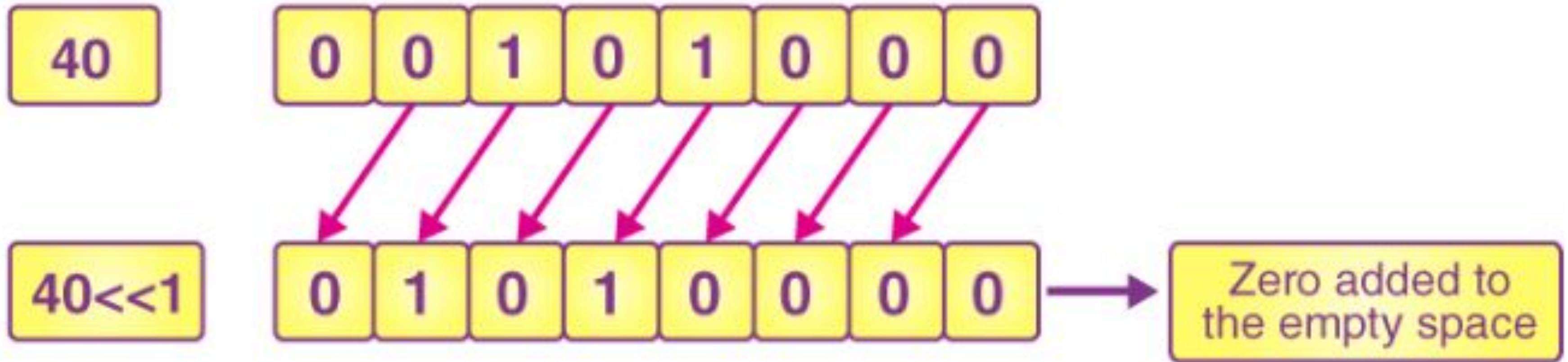
>> Right Shift - Bitwise Operators in C



```c
/*Output:  The right shift will be 0: 10
           The right shift will be 1: 5   */
#include <stdio.h>
int main() {
int a = 10; // 1010 binary equivalent
int b = 0;
for(b;b<2;b++)
 printf("Right shift will be %d: %d\n", b, a>>b);
return 0;
}
```

Formula: $a / 2^b$

>> Left Shift - Bitwise Operators in C



>> Left Shift - Bitwise Operators in C

```
/* Left Shift - Bitwise Operator */
#include<stdio.h>
int main() {
int a = 28; // 11100
int b = 0;
for(b;b<=3;++b)
 printf("Left shift by %d: %d\n", b, a<<b);
 //x<<y, Formula: x*(2 pow y)
return 0;
}
```

Left shift by 0: 28

Left shift by 1: 56

Left shift by 2: 112

Left shift by 3: 224

Q&A - Bitwise Operators in C

1. What would be the result obtained by using a right shift operator on 23 >> 2?

A. 6

B. 1

C. Undefined

D. 13

Q&A - Bitwise Operators in C

1. What would be the result obtained by using a right shift operator on 23 >> 2?

A. 6

B. 1

C. Undefined

D. 13

$23 / 2^2 = 23 / 4 = 6$ (closest integer).

Q&A - Bitwise Operators in C

What would be the result obtained by using a right shift operator on 128 >> 5?

A. 120

B. 4

C. 11

D. Undefined

Q&A - Bitwise Operators in C

What would be the result obtained by using a right shift operator on 128 >> 5?

A. 120

B. 4

C. 11

D. Undefined

128 >> 5 = 4, i.e,
$128 / 2^5 = 128 / 32$
= 4

Q&A - Bitwise Operators in C

What would be the result obtained by using a right shift operator on 64 >> 3?

A. 60

B. 12

C. Undefined

D. 8

Q&A - Bitwise Operators in C

What would be the result obtained by using a right shift operator on 64 >> 3?

A. 60

B. 12

C. Undefined

D. 8

64 >> 3 = 4, i.e,

$64 / 2^3 = 64 / 8 = 8$

Q&A - Bitwise Operators in C

Can we use the right shift operator with the negative numbers?

No. the right shift operator only works with the positive integers.

We **must not use a negative number**.

When either of the operands is negative, the result obtained will be **undefined**.

Operators in C

Type of Operator	Associativity	Category
() [] -> . ++ − −	Left to right	Postfix
− + ! ~ − − ++ (type)* & sizeof	Right to left	The Unary Operator
/ * %	Left to right	The Multiplicative Operator
− +	Left to right	The Additive Operator
>> <<	Left to right	The Shift Operator
< > >= <=	Left to right	The Relational Operator
!= ==	Left to right	The Equality Operator
&	Left to right	Bitwise AND
^	Left to right	Bitwise XOR
|	Left to right	Bitwise OR
&&	Left to right	Logical AND
||	Left to right	Logical OR
?:	Right to left	Conditional
= -= += /= *= %=>>= &= <<= |= ^=	Right to left	Assignment
,	Left to right	Comma

Precedence (Order of Execution)

BEDMAS

(Bracket, Exponent, Div, Mult, Add, Sub)

PEMDAS

(Parenthesis, Exponent, Mult, Div, Add, Sub)

Variables in C

Rules for Identifiers - Variables in C

- Case sensitive
- First character: Alphabetic or Underscore
- NO Space, NO Hyphen, No Special character
- First 63 characters are significant
- Cannot duplicate a Keyword

Valid Names		Invalid Name	
a	// Valid but poor style	$sum	// $ is illegal
student_name		2names	// First char digit
_aSystemName		sum-salary	// Contains hyphen
_Bool	// Boolean System id	stdnt Nmbr	// Contains spaces
INT_MIN	// System Defined Value	int	// Keyword

```
bool    fact;
short   maxItems;               // Word separator: Capital
long    long national_debt;     // Word separator: underscore
float   payRate;                // Word separator: Capital
double  tax;
float   complex voltage;
char    code, kind;             // Poor style—see text
int     a, b;                   // Poor style—see text
```

When a variable is defined, it is not initialized.
We must initialize any variable requiring prescribed data when the function starts.

Control characters

ASCII Character	Symbolic Name
null character	'\0'
alert (bell)	'\a'
backspace	'\b'
horizontal tab	'\t'
newline	'\n'
vertical tab	'\v'
form feed	'\f'
carriage return	'\r'
single quote	'\''
double quote	'\"'
backslash	'\\'

Fo

EscapeControl characters

Special Character	Description
\b	backspace BS
\n	new line NL (like pressing return)
\f	form feed FF (also clear screen)
\t	horizontal tab HT
\v	vertical tab (not all versions)
\r	carriage return CR (cursor to start of line)
\a	audible bell
\”	double quotes (not all versions)
\’	single quote character '
\\	backslash character \

Activate Windows
Go to Settings to activate Windows