**C Cheat Sheet For Varsity**

|  |  |
| --- | --- |
| **c** | Case Sensitive (small/large) |
|  |
| **C** has support for **Octal** & **Hexadecimal** |
|  |

|  |
| --- |
| **Programs** |
|  |

|  |  |
| --- | --- |
| **Difference** |  |

|  |  |
| --- | --- |
| **keywords** | **Word** |
| 32 |
|  |
|  |

|  |  |
| --- | --- |
| **Identifier** | **Name** |
| int a=5, b=10; here **a, b identifier** |
| Identifier helps variables to remind its name |
|  |

|  |  |
| --- | --- |
| **Variable** | Variable’s Value can be changed but address cannot be changed. |
|  |
|  |
|  |

|  |  |
| --- | --- |
| **Character Set** | **A - Z** |
| a – z |
| + - / \* $ % 1 2 3 …… etc |
| Total numbers of characters 256 |

|  |  |  |
| --- | --- | --- |
| **Identifier** | **Character** | **String** |
| a | ‘a’ | “a” |

|  |  |  |
| --- | --- | --- |
| **string** | **integer** | **floating point** |
| saifur | 67 | 67.0 |
| A+ | 4 | 4.0 |

|  |  |
| --- | --- |
| **Data Types** |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Library/Built-In Function** | | | | | |
| **Serial** | **Name** | **Description** | **Parameters** | **Return Value** | **Example & output** |
| 1 | **abs()** | **Absolute value**  Returns the absolute value of parameter *n* ( /n/ ).  In C++, this function is also overloaded in header [<cmath>](http://www.cplusplus.com/cmath) for floating-point types (see [cmath abs](http://www.cplusplus.com/cmath:abs)), in header[<complex>](http://www.cplusplus.com/complex) for complex numbers (see [complex abs](http://www.cplusplus.com/complex:abs)), and in header [<valarray>](http://www.cplusplus.com/valarray) for valarrays (see [valarray abs](http://www.cplusplus.com/valarray:abs)).  double abs (double x);  float abs (float x);  long double abs (long double x); | Integral value. | The absolute value of n.   1. Absolute value for **int** input & 2. **int** value for output   Like( **abs(-3)=3** ) | |  |  | | --- | --- | | 1 2 3 4 5 6 7 8 9 10 11 12 13 | /\* abs example \*/  #include <stdio.h> /\* printf \*/  #include <stdlib.h> /\* abs \*/  int main ()  {  int n,m;  n=abs(23); // 23  m=abs(-11); // 11  printf ("n=%d\n",n);  printf ("m=%d\n",m);  return 0;  } | |
| 2 | **fabs()** | **Compute absolute value**  Returns the *absolute value* of *x*: |*x*|.  double fabs (double x);  float fabsf (float x);  long double fabsl (long double x); | Value whose absolute value is returned. | The absolute value of *x*.   1. Absolute value for floating point input but 2. Int value for output   Like( fabs(-3.5)=3.5) | |  |  | | --- | --- | | 1 2 3 4 5 6 7 8 9 10 | /\* fabs example \*/  #include <stdio.h> /\* printf \*/  #include <math.h> /\* fabs \*/  int main ()  {  printf ("The absolute value of 3.1416 is %f\n", fabs (3.1416) );  printf ("The absolute value of -10.6 is %f\n", fabs (-10.6) );  return 0;  }  Result:::::::  The absolute value of 3.1416 is 3.141600  The absolute value of -10.6 is 10.600000 | |
| 3 | **Ceil()** | **Round up value**  Rounds *x* upward, returning the smallest integral value that is not less than *x*.  double ceil (double x);  float ceilf (float x);  long double ceill (long double x); | Value to round up. | The smallest integral value that is not less than x (as a floating-point value).   1. To get Larger int output 2. Value int 3. Type floating   Like( **ceil(2.5)=3.0** ) | |  |  | | --- | --- | | 1 2 3 4 5 6 7 8 9 10 11 12 | /\* ceil example \*/  #include <stdio.h> /\* printf \*/  #include <math.h> /\* ceil \*/  int main ()  {  printf ( "ceil of 2.3 is %.1f\n", ceil(2.3) );  printf ( "ceil of 3.8 is %.1f\n", ceil(3.8) );  printf ( "ceil of -2.3 is %.1f\n", ceil(-2.3) );  printf ( "ceil of -3.8 is %.1f\n", ceil(-3.8) );  return 0;  } |   Output:   |  | | --- | | ceil of 2.3 is 3.0  ceil of 3.8 is 4.0  ceil of -2.3 is -2.0  ceil of -3.8 is -3.0 | |
| 4 | **floor()** | double floor (double x);  float floorf (float x);  long double floorl (long double x);  **Round down value**  Rounds *x* downward, returning the largest integral value that is not greater than *x*.  Header [<tgmath.h>](http://www.cplusplus.com/%3Cctgmath%3E) provides a type-generic macro version of this function. | Value to round down. | The value of x rounded downward (as a floating-point value).   1. To get smaller int output but 2. Value int 3. Type floating point   Like( **floor(2.5)=2.0** ) | |  |  | | --- | --- | | 1 2 3 4 5 6 7 8 9 10 11 12 | /\* floor example \*/  #include <stdio.h> /\* printf \*/  #include <math.h> /\* floor \*/  int main ()  {  printf ( "floor of 2.3 is %.1lf\n", floor (2.3) );  printf ( "floor of 3.8 is %.1lf\n", floor (3.8) );  printf ( "floor of -2.3 is %.1lf\n", floor (-2.3) );  printf ( "floor of -3.8 is %.1lf\n", floor (-3.8) );  return 0;  } |   Output:   |  | | --- | | floor of 2.3 is 2.0  floor of 3.8 is 3.0  floor of -2.3 is -3.0  floor of -3.8 is -4.0 | |
| 5 | **fmod()** | double fmod(double x, double y);  The C library function **double fmod(double x, double y)** returns the remainder of **x** divided by **y**.  The C library function **double fmod(double x, double y)** returns the remainder of **x** divided by **y**. |  | 1. Type floating point 2. Value int / floating point | |  |  | | --- | --- | | 1 2 3 4 5 6 7 8 9 10 | /\* fmod example \*/  #include <stdio.h> /\* printf \*/  #include <math.h> /\* fmod \*/  int main ()  {  printf ("fmod of 2.5 / 2 is %lf\n", fmod(2.5,2));  printf ( "fmod of 5.3 / 2 is %f\n", fmod (5.3,2) );  printf ( "fmod of 18.5 / 4.2 is %f\n", fmod (18.5,4.2) );  printf ( "fmod of 18.5 / 4.2 is %f\n", fmod (5,2) );  return 0;  } | |  |  |   Output:   |  | | --- | | fmod of 2.5 / 2 is 0.500000  fmod of 5.3 / 2 is 1.300000  fmod of 18.5 / 4.2 is 1.700000 | |
| 6 | **sqrt()** | Returns the *square root* of x.  *Additional overloads* are provided in this header ([<cmath>](http://www.cplusplus.com/%3Ccmath%3E)) for the [integral types](http://www.cplusplus.com/is_integral): These overloads effectively cast x to a double before calculations (defined for T being any [*integral type*](http://www.cplusplus.com/is_integral)).  This function is also overloaded in [<complex>](http://www.cplusplus.com/%3Ccomplex%3E) and [<valarray>](http://www.cplusplus.com/%3Cvalarray%3E) (see [complex sqrt](http://www.cplusplus.com/complex:sqrt) and [valarray sqrt](http://www.cplusplus.com/valarray:sqrt)). |  |  | |  |  | | --- | --- | | 1 2 3 4 5 6 7 8 9 10 11 12 | /\* sqrt example \*/  #include <stdio.h> /\* printf \*/  #include <math.h> /\* sqrt \*/  int main ()  {  double param, result;  param = 1024.0;  result = sqrt (param);  printf ("sqrt(%f) = %f\n", param, result );  return 0;  } |   Output:   |  | | --- | | sqrt(1024.000000) = 32.000000 | |
| 7 | **pow()** |  | base  Base value.  exponent  Exponent value. | The result of raising base to the power exponent.  If x [is finite](http://www.cplusplus.com/isfinite) negative and y [is finite](http://www.cplusplus.com/isfinite) but not an integer value, it causes a *domain error*. If both x and y are zero, it may also cause a *domain error*. If x is zero and y is negative, it may cause a *domain error* or a *pole error* (or none, depending on the library implementation). The function may also cause a *range error* if the result is too great or too small to be represented by a value of the return type. | |  |  | | --- | --- | | 1 2 3 4 5 6 7 8 9 10 11 | /\* pow example \*/  #include <stdio.h> /\* printf \*/  #include <math.h> /\* pow \*/  int main ()  {  printf ("7 ^ 3 = %f\n", pow (7.0, 3.0) );  printf ("4.73 ^ 12 = %f\n", pow (4.73, 12.0) );  printf ("32.01 ^ 1.54 = %f\n", pow (32.01, 1.54) );  return 0;  } |   Output:   |  | | --- | | 7 ^ 3 = 343.000000  4.73 ^ 12 = 125410439.217423  32.01 ^ 1.54 = 208.036691 | |
| 8 | **log()** | double log (double x);  float log (float x);  long double log (long double x);  double log (T x); // additional overloads for integral types  Returns the *natural logarithm* of x.  The natural logarithm is the base-e logarithm: the inverse of the natural exponential function ([exp](http://www.cplusplus.com/exp)). For common (base-10) logarithms, see [log10](http://www.cplusplus.com/log10). | Value whose logarithm is calculated. If the argument is negative, a *domain error* occurs. |  | |  |  | | --- | --- | | 1 2 3 4 5 6 7 8 9 10 11 12 | /\* log example \*/  #include <stdio.h> /\* printf \*/  #include <math.h> /\* log \*/  int main ()  {  double param, result;  param = 5.5;  result = log (param);  printf ("log(%f) = %f\n", param, result );  return 0;  } |   Output:   |  | | --- | | log(5.500000) = 1.704748 | |
| 9 | **exp()** | double exp (double x);  float exp (float x);  long double exp (long double x);  double exp (T x); // additional overloads for integral types  **Compute exponential function**  Returns the *base-e* exponential function of *x*, which is *e* raised to the power *x*: ex.  *Additional overloads* are provided in this header ([<cmath>](http://www.cplusplus.com/%3Ccmath%3E)) for the [integral types](http://www.cplusplus.com/is_integral): These overloads effectively cast x to a double before calculations.  This function is also overloaded in [<complex>](http://www.cplusplus.com/%3Ccomplex%3E) and [<valarray>](http://www.cplusplus.com/%3Cvalarray%3E) (see [complex exp](http://www.cplusplus.com/complex:exp) and [valarray exp](http://www.cplusplus.com/valarray:exp)). | Value of the exponent. | Exponential value of x. If the magnitude of the result is too large to be represented by a value of the return type, the function returns[HUGE\_VAL](http://www.cplusplus.com/HUGE_VAL) (or [HUGE\_VALF](http://www.cplusplus.com/HUGE_VALF) or [HUGE\_VALL](http://www.cplusplus.com/HUGE_VALL)) with the proper sign, and an overflow *range error* occurs: | |  |  | | --- | --- | | 1 2 3 4 5 6 7 8 9 10 11 12 | /\* exp example \*/  #include <stdio.h> /\* printf \*/  #include <math.h> /\* exp \*/  int main ()  {  double param, result;  param = 5.0;  result = exp (param);  printf ("The exponential value of %f is %f.\n", param, result );  return 0;  } |   Output:   |  | | --- | | The exponential value of 5.000000 is 148.413159. | |
| 10 | **rand()** | int rand (void);  **Generate random number**  Returns a pseudo-random integral number in the range between 0 and [RAND\_MAX](http://www.cplusplus.com/RAND_MAX).  This number is generated by an algorithm that returns a sequence of apparently non-related numbers each time it is called. This algorithm uses a seed to generate the series, which should be initialized to some distinctive value using function [srand](http://www.cplusplus.com/srand).  [RAND\_MAX](http://www.cplusplus.com/RAND_MAX) is a constant defined in [<cstdlib>](http://www.cplusplus.com/cstdlib).  A typical way to generate trivial pseudo-random numbers in a determined range using rand is to use the modulo of the returned value by the range span and add the initial value of the range:   |  |  | | --- | --- | | 1 2 3 | v1 = rand() % 100; // v1 in the range 0 to 99  v2 = rand() % 100 + 1; // v2 in the range 1 to 100  v3 = rand() % 30 + 1985; // v3 in the range 1985-2014 |   Notice though that this modulo operation does not generate uniformly distributed random numbers in the span (since in most cases this operation makes lower numbers slightly more likely).  C++ supports a wide range of powerful tools to generate random and pseudo-random numbers (see [<random>](http://www.cplusplus.com/random)for more info). |  | An integer value between 0 and [RAND\_MAX](http://www.cplusplus.com/RAND_MAX). | |  |  | | --- | --- | | 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 | /\* rand example: guess the number \*/  #include <stdio.h> /\* printf, scanf, puts, NULL \*/  #include <stdlib.h> /\* srand, rand \*/  #include <time.h> /\* time \*/  int main ()  {  int iSecret, iGuess;  /\* initialize random seed: \*/  srand (time(NULL));  /\* generate secret number between 1 and 10: \*/  iSecret = rand() % 10 + 1;  do {  printf ("Guess the number (1 to 10): ");  scanf ("%d",&iGuess);  if (iSecret<iGuess) puts ("The secret number is lower");  else if (iSecret>iGuess) puts ("The secret number is higher");  } while (iSecret!=iGuess);  puts ("Congratulations!");  return 0;  } |   In this example, the random seed is initialized to a value representing the current time (calling [time](http://www.cplusplus.com/time)) to generate a different value every time the program is run.  Possible output:   |  | | --- | | Guess the number (1 to 10): 5  The secret number is higher  Guess the number (1 to 10): 8  The secret number is lower  Guess the number (1 to 10): 7  Congratulations! | |
| 11 | **srand()** | void srand (unsigned int seed);  **Initialize random number generator**  The pseudo-random number generator is initialized using the argument passed as seed.  For every different seed value used in a call to srand, the pseudo-random number generator can be expected to generate a different succession of results in the subsequent calls to [rand](http://www.cplusplus.com/rand).  Two different initializations with the same seed will generate the same succession of results in subsequent calls to[rand](http://www.cplusplus.com/rand).  If seed is set to 1, the generator is reinitialized to its initial value and produces the same values as before any call to [rand](http://www.cplusplus.com/rand) or srand.  In order to generate random-like numbers, srand is usually initialized to some distinctive runtime value, like the value returned by function [time](http://www.cplusplus.com/time) (declared in header [<ctime>](http://www.cplusplus.com/ctime)). This is distinctive enough for most trivial randomization needs. | An integer value to be used as seed by the pseudo-random number generator algorithm. |  | |  |  | | --- | --- | | 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 | /\* srand example \*/  #include <stdio.h> /\* printf, NULL \*/  #include <stdlib.h> /\* srand, rand \*/  #include <time.h> /\* time \*/  int main ()  {  printf ("First number: %d\n", rand()%100);  srand (time(NULL));  printf ("Random number: %d\n", rand()%100);  srand (1);  printf ("Again the first number: %d\n", rand()%100);  return 0;  } |   Possible output:   |  | | --- | | First number: 41  Random number: 13  Again the first number: 41 | |
| 12 | **tolower()** | |  | | --- | | [**short**](http://tigcc.ticalc.org/doc/keywords.html#short) \_tolower ([**short**](http://tigcc.ticalc.org/doc/keywords.html#short) c); |   **Translates uppercase characters to lowercase.**  \_tolower does the same conversion as [tolower](http://tigcc.ticalc.org/doc/ctype.html" \l "tolower) or [toextlower](http://tigcc.ticalc.org/doc/ctype.html" \l "toextlower), except that it should be used only when *c* is known to be uppercase (either ordinary or foreign). It is faster, and generates much shorter code than [tolower](http://tigcc.ticalc.org/doc/ctype.html" \l "tolower). \_tolower returns the converted value of *c* if it is uppercase; otherwise, the result is undefined. \_tolower is a simple macro.   |  | | --- | | [**short**](http://tigcc.ticalc.org/doc/keywords.html#short) tolower ([**short**](http://tigcc.ticalc.org/doc/keywords.html#short) c); |   **Translates characters to lowercase.**  tolower is a function that converts an integer *c* to its lowercase value ('a' to 'z') if it was uppercase ('A' to 'Z'). All others are left unchanged. Returns the converted value of *c*. tolower is a relatively small inline function which is implemented using GNU C smart macros. |  |  |  |
| 13 | **toupper()** | |  | | --- | | [**short**](http://tigcc.ticalc.org/doc/keywords.html#short) \_toupper ([**short**](http://tigcc.ticalc.org/doc/keywords.html#short) c); |   **Translates uppercase characters to lowercase.**  \_toupper does the same conversion as [toupper](http://tigcc.ticalc.org/doc/ctype.html" \l "toupper) or [toextupper](http://tigcc.ticalc.org/doc/ctype.html" \l "toextupper), except that it should be used only when *c* is known to be lowercase (either ordinary or foreign). It is faster, and generates much shorter code than [toupper](http://tigcc.ticalc.org/doc/ctype.html" \l "toupper). \_toupper returns the converted value of *c* if it is lowercase; otherwise, the result is undefined. \_toupper is a simple macro.   |  | | --- | | [**short**](http://tigcc.ticalc.org/doc/keywords.html#short) toupper ([**short**](http://tigcc.ticalc.org/doc/keywords.html#short) c); |   **Translates characters to uppercase.**  toupper is a function that converts an integer *c* to its uppercase value ('A' to 'Z') if it was lowercase ('a' to 'z'). All others are left unchanged. Returns the converted value of *c*. toupper is a relatively small inline function which is implemented using GNU C smart macros. |  |  |  |
| 14 | **isalnum()** | |  | | --- | | [**short**](http://tigcc.ticalc.org/doc/keywords.html#short) isalnum ([**short**](http://tigcc.ticalc.org/doc/keywords.html#short) c); |   **Checks whether a character is an alphanumeric.**  isalnum returns nonzero if *c* is a letter ('A' to 'Z' or 'a' to 'z') or a digit ('0' to '9'), otherwise it returns zero. It is an inline function which is implemented using GNU C smart macros, which expands to a medium-sized code. |  |  |  |
| 15 | **isalpha()** | |  | | --- | | [**short**](http://tigcc.ticalc.org/doc/keywords.html#short) isalpha ([**short**](http://tigcc.ticalc.org/doc/keywords.html#short) c); |   **Checks whether a character is a letter.**  isalpha returns nonzero if *c* is a letter ('A' to 'Z' or 'a' to 'z'), otherwise it returns zero. It is an inline function which is implemented using GNU C smart macros, which expands to a relatively small code. |  |  |  |
| 16 | **isascii()** | |  | | --- | | [**short**](http://tigcc.ticalc.org/doc/keywords.html#short) isascii ([**short**](http://tigcc.ticalc.org/doc/keywords.html#short) c); |   **Checks whether a character is an ASCII character.**  isascii returns nonzero if *c* is in the range 0 to 127 (0x00-0x7F), otherwise it returns zero. It is a simple macro. |  |  |  |
| 17 | **islower()** | |  | | --- | | [**short**](http://tigcc.ticalc.org/doc/keywords.html#short) islower ([**short**](http://tigcc.ticalc.org/doc/keywords.html#short) c); |   **Checks whether a character is a lowercase.**  islower returns nonzero if *c* is a lowercase letter ('a' to 'z'), otherwise it returns zero. It is a small inline function which is implemented using GNU C smart macros. |  |  |  |
| 18 | **isupper()** | |  | | --- | | [**short**](http://tigcc.ticalc.org/doc/keywords.html#short) isupper ([**short**](http://tigcc.ticalc.org/doc/keywords.html#short) c); |   **Checks whether a character is an uppercase.**  isupper returns nonzero if *c* is an uppercase letter ('A' to 'Z'), otherwise it returns zero. It is a small inline function which is implemented using GNU C smart macros. |  |  |  |
| 19 | **toascii()** | |  | | --- | | [**short**](http://tigcc.ticalc.org/doc/keywords.html#short) toascii ([**short**](http://tigcc.ticalc.org/doc/keywords.html#short) c); |   **Translates characters to ASCII format.**  toascii converts the integer *c* to ASCII by clearing all but the lower 7 bits. This gives a value in the range 0 to 127. Returns the converted value of *c*. toascii is a simple macro. |  |  |  |
| 20 |  |  |  |  |  |
| 21 |  |  |  |  |  |
| 22 |  |  |  |  |  |
| 23 |  |  |  |  |  |
| 24 |  |  |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Function** | **Input: Type** | **Input: Value** | **Output: Type** | **Output: Value** |
| abs() | int | int | int | int |
| fabs() | float | float | float | float |
| ceil() | float | int / float | float | int |
| floor() | float | int / float | float | int |
| fmod() | int / float | int / float | float | int / float |
| Sqrt() | float | int | float | Int/float |
| Pow() | Int/float | int / float | float | int / float |
| Log() | Int/float | Int/float | float | Int / float |
| Exp() |  |  | float | float |

|  |
| --- |
| **Escape Sequence** |
| |  |  | | --- | --- | | **Escape Sequence** | **Represents** | | **\a** | Bell / (alert) / beep / | | **\b** | Backspace / non-erase | | **\f** | Formfeed / clear screen / new page / | | **\n** | New line | | **\r** | Carriage return | | **\t** | Horizontal tab | | **\v** | Vertical tab | | **\'** | Single quotation mark | | **\** " | Double quotation mark | | **\\** | Backslash | | **\?** | Literal question mark | | **\0** | null | | **\** *ooo* | ASCII character in octal notation | | **\x** *hh* | ASCII character in hexadecimal notation | | **\x** *hhhh* | Unicode character in hexadecimal notation if this escape sequence is used in a wide-character constant or a Unicode string literal.  For example, WCHAR f = L'\x4e00' or WCHAR b[] = L"The Chinese character for one is \x4e00". | |

|  |  |  |  |
| --- | --- | --- | --- |
| **printf** | | **scanf** | |
| %d | integer | %d | integer |
| %f | float or double | %f | float |
| %s | string (char array) | %lf | double (first character is l, not one!) |
| %c | char (single character) | %s | string (char array) |
|  |  | %c | char (single character) |

**Measurements**

|  |  |
| --- | --- |
| 1 Byte/B | 8 Bit |
| 1 KB | 1024 Byte |
| 1 MB | 1024 KB |
| 1 GB | 1024 MB |
| 1 TB | 1024 GB |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Integer** | | | | |
| **Type** | **Turbo C++ (16 Bit)** | **Visual C++ (32 Bit)** | **gcc / g++** | **64 Bit** |
| short | 2 Byte (16 Bit) | 2 | 2 | - |
| int | 2 | 4 | 4 | 8 |
| long | 4 | 4 | 4 | - |
| long long | - | 8 | 8 | 8 |

|  |
| --- |
| **32 Bit / 4 Byte { 1Bit (contains = 0,1)}** |
| |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 65536 | 32768 | 16384 | 8192 | 4096 | 2048 | 1024 | 512 | 256 | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 | | 32 | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 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 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | | 2^31 | 2^30 | 2^29 | 2^28 | 2^27 | 2^26 | 2^25 | 2^24 | 2^23 | 2^22 | 2^21 | 2^20 | 2^19 | 2^18 | 2^17 | 2^16 | 2^15 | 2^14 | 2^13 | 2^12 | 2^11 | 2^10 | 2^9 | 2^8 | 2^7 | 2^6 | 2^5 | 2^4 | 2^3 | 2^2 | 2^1 | 2^0 | |

|  |  |
| --- | --- |
| **Range** | |
| **Signed short/int/long/long long** | **Unsigned short/int/long/long long** |
|  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **Int** | | **Float /Fractional** | |
| **Character** | 1 Byte |  |  |
| **short** | 2 |  |  |
| **int** | 2/4 |  |  |
| **long** | 4 | **float** | 4 |
| **long long** | 8 | **double** | 8 |
|  |  | **long double** | 10 |

|  |  |  |
| --- | --- | --- |
| **Usage** | **Decimal** | **Hexa-Decimal** |
| int + float | int |

|  |  |  |  |
| --- | --- | --- | --- |
| small + small = large  small + large = large  large + large = extra large  **minimum low result:**  int & float (~~char~~ & ~~float~~ & ~~short~~) | char + char =int | float + float = double | int + float = double |
| char + int =int | float +double = double | long + double = double |
| int + int =long (not sure) | double +long double=long double |  |
| int + char = int |  |  |
| Int + long=long |  |  |
| int + long long = long long |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **ASCII Character set/ American Standard Code for Information Interchange** | | | |
|  | | | |
| ‘A’ = 65 | ‘a’ = 97 | ‘0’ = 48 | Total character = 256 ( 0 -> 255) |
| ‘Space’ = 32 |  |  |  |

|  |
| --- |
| **2.1** |
|  |