

Variables:

```
[signed, unsigned] int x;
[signed, unsigned] char x = 'C';
[signed, unsigned] short x;
[signed, unsigned] long;
[signed, unsigned] long long
float x, y, z;
double x;
long double y;
const int x = 88;
```

Structures:**Defining:**

```
struct structName{
    type1;
    type2;
};
```

Declaring:

```
struct structName varName;
struct structName* ptrName;
```

Accessing:

```
varName.x
ptrName->x
```

Pointers:**Declaration:**

```
type *x;
void *v;
struct type *y;
type z[];
```

Accessing:

```
x          - A memory address
*x         - Value stored in address (dereference)
y->a       - Value stored in struct ptr y
&varName  - Memory address of normal var
*(type*)v - Dereference void pointer into type
```

Conditional:

```
if( conditional ) { ... }
else if( conditional ) { ... }
else { ... }
switch( conditional ) {
    case x: ...
        break;
    default: ...
        break;
}
while ( conditional ) { ... }
do { ... } while ( conditional );
for ( i = _ ; i _ _ ; i _ ) { ... }
continue          skip iteration of loop
break             skips rest of loop
```

Arrays:**Declaration:**

```
type name[int];          array length int
type name[int] = {x, y, z}; array length & initialize
type name[int] = {x};    set all elements to x
type name[] = {x, y, z}; compiler sets length
```

Dimensions:

```
name[int]                one-dimensional
name[int][int]           two-dimensional
```

Accessing:

```
name[int]                value at index 'int'
*(name + int)            same as name[int]
&name[int]               memory address at 'int'
name + int                same as &name[int]
```

Measuring:

```
sizeof(array)/sizeof(array[0]) returns length of array
```

Strings:

```
'A'                      char - single quotes
"AB"                     string - double quotes
\\0                       null terminator
char name[4] = "Ash";    strings are char arrays
char name[4] = {'A', 's', 'h', '\\0'}; both are equivalent
```

Functions:

```
type/void funcName([args...]) { [return var;] }
```

By Value

```
void f(type x);          pass variable
f(y);
```

By Reference

```
void f(type *x);         passing pointer
f(&y);                   pointer variable
f(array);                pointer array
f(structure);            pointer struct
```

Return Value

```
return x;                return variable
```

Return Reference

```
return &x;               return variable by pointer
static type x[]; return &x; static type necessary or
&x will not exist outside of function
```

Heap Space:

```
malloc();                returns mem location
type *x; x = malloc(sizeof(type)); allocates for variable
x = malloc(sizeof(type) * length); allocates for array of var
x = malloc(sizeof(struct type));  allocates for struct
free(ptrName);           free memory for pointer
realloc(ptrName, size);   attempt to resize memory
```

Placeholder Types: (printf/scanf)

Type	Example	Description
%d or %i	-42	Signed decimal integer.
%u	42	Unsigned decimal integer.
%o	52	Unsigned octal integer.
%x or %X	2a or 2A	Unsigned hexadecimal integer.
%f or %F	1.21	Signed decimal float.
%e or %E	1.21e+9	Signed w/ scientific notation.
%g or %G	1.21e+9	Shortest representation of
%a or %A	0x1.207c8ap+30	Signed hexadecimal float.
%c	a	A character.
%s	A String.	A character string.
%p		A pointer.
%%	%	A percent character.

Preprocessor Directives:

#include <h >	include standard header
#include "h"	include custom header
#define NAME value	replace NAME with value

Standard Library:

#include <stdlib.h>	loads library
rand()	returns random number
RAND_MAX	maximum value of rand()
srand(unsigned_int)	seeds randomiser
(unsigned)time(NULL)	returns tick-tock value
qsort(array, length, sizeof(type), compFunc (returns int));	sort with quicksort array to sort length of array byte size of each element comparison function

Character Type Library:

#include <ctype.h>	
tolower(char)	
toupper(char)	
isalpha(char)	true if is alphabetical
islower(char)	true if lowercase
isupper(char)	true if uppercase
isnumber(char)	true if numeric
isblank	true if whitespace

String Library:

#include <string.h>	
strlen(a)	returns # of chars
strcpy(a, b)	copies b over a
strcat(a, b)	concatenates strings
strcmp(a,b)	compares strings
strstr(a,b)	searches for b in a
strncpy(a,b,n)	copies b over a up to n
strncat(a,b,n)	concatenates up to n
strncmp(a,b,n)	compares first n chars