General

05 November 2021 16:18

Basics
Variables
Data types
Function
Pointers
Enum, Struct and Union
Memory management
Arrays and string
Control statement
I/o
File handling
Operators
Puzzles - relevant to us
[16:13, 05/11/2021] Intisar: finish functions by monday

Basics

05 November 2021 09:08

Regular int is 32 bits (leading bit for signed (0 for positive, 1 for negative). Max is 2147483647
Unsigned int is 32 bits. Max is 4294967295

Using include means content of header file copied into C file.
Using define (macros) means variable will be replaced in all places into C file.
Macros can take functions

Header file can be included multiple times by accident. Leads to problems of redeclaration of some variables/functions. To avoid use DEFINED, IFEEF, IFNDEF

Can remove macros like #undef MACRO_NAME

If macro not defined then pre-processor assigns 0 to it by default

Preprocessing generates a .i file which is passed to compiler

"#pragma once" used in header file to avoid its inclusion more than once.

Can't be spaces between macro name and the (
INC1 correct INC2 not
#define INC1(a) ((a)+1)
#define INC2 (a) ((a)+1) // will be INC2 = "(a) ((a)+1)" instead of evaluating (a)+1

even incorrectly defined macros will let compilation happens

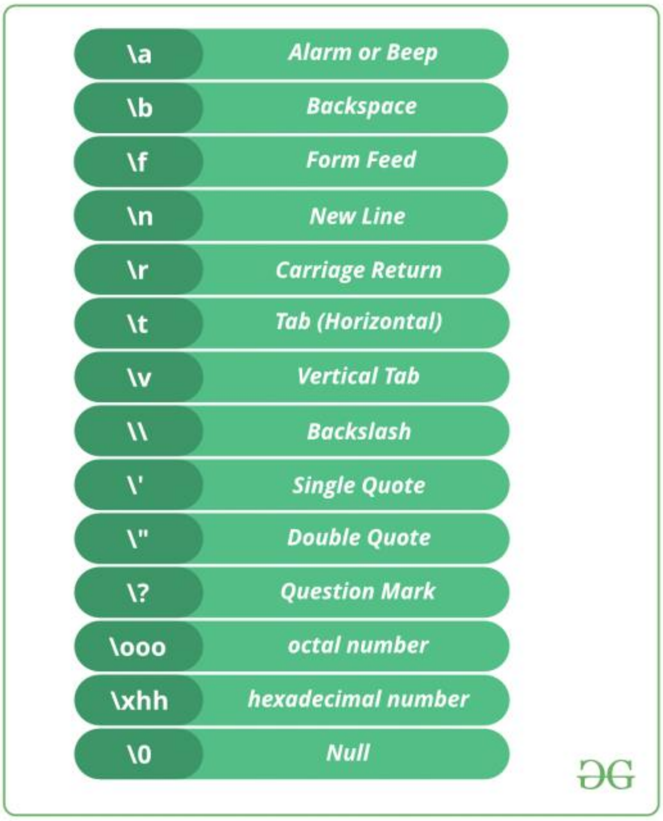
The translator which performs macro calls expansion is called Macro pre - processor.
A Macro processor is a program that copies a stream of text from one place to another, making a systematic set of replacements as it does so.
A Dynamic linker is the part of an operating system that loads and links the shared libraries.
The C preprocessor is a micro processor that is used by compiler to transform your code before compilation. It is called micro pre-processor because it allows us to add macros.

C is a middle-level language. Can handle low level things such as kernels/drivers and supports high -level such as scripting for software applications.
C features

- Access machine level hardware API
- Presence of C compilers
- Deterministic resource use and dynamic memory allocation (good for scripting apps and drivers of embedded systems)
- Case sensitive
- Portable with windows/unix/linux

Signals way for communication between process and OS. When there's an error OS sends signal to process.
Signal generated by kernal and handled by process. Interrupt generated by process and handled by kernel.
Errors include:

- SIGFPE; SIGILL; SIGSEGV; SIGBUS



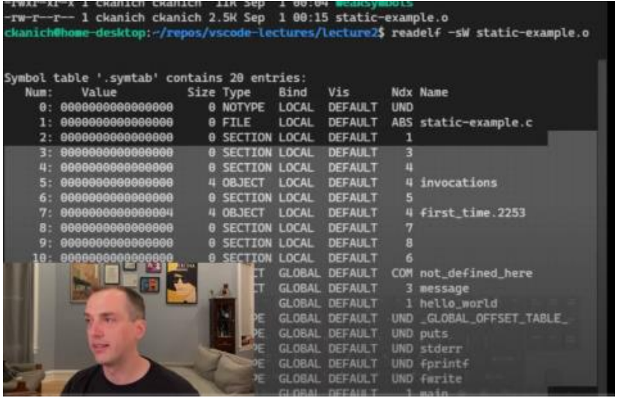
\a	Alarm or Beep
\b	Backspace
\f	Form Feed
\n	New Line
\r	Carriage Return
\t	Tab (Horizontal)
\v	Vertical Tab
\\	Backslash
\'	Single Quote
\"	Double Quote
\?	Question Mark
\ooo	octal number
\xhh	hexadecimal number
\0	Null

Variables

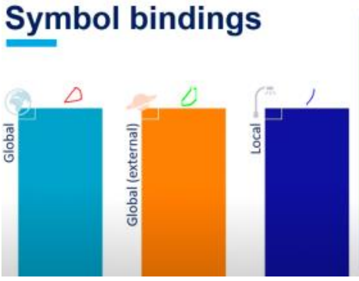
05 November 2021 11:46

Declaration - declares variable or function exists in program. Memory not allocated. Tells program what type it's going to be. Function declaration tells program arguments, datatypes, order, and return.
Definition - in addition to declaration also allocates memory for that variable/function. Superset of declaration. Only happens once.

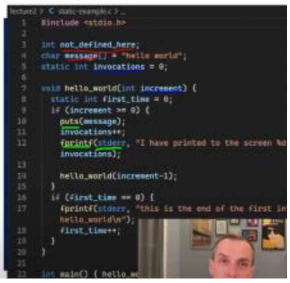
Strong symbol - defined functions; initialized variables (multiple with same name not allowed)
weak symbol - uninitialized variables
https://www.geeksforgeeks.org/how-linkers-resolve-multiply-defined-global-symbols/
After GCC these are the symbols I think. Symbol is a variable that the linker needs to care about.



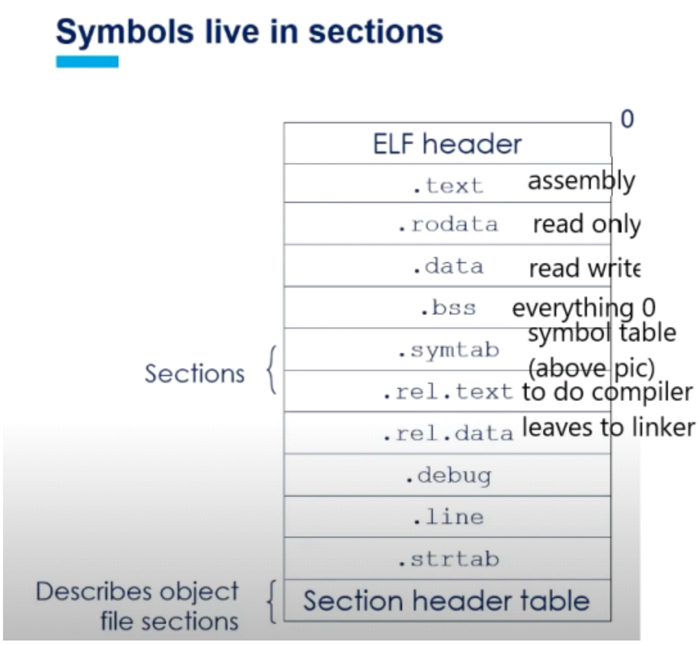
```
-rwxr-xr-x 1 ckanich ckanich  11K Sep  1 00:04 weaksymbols
-rw-r--r-- 1 ckanich ckanich 2.5K Sep  1 00:15 static-example.o
ckanich@home-desktop:~/repos/vscode-lectures/lecture2$ readelf -sW static-example.o


Symbol table '.symtab' contains 20 entries:
   Num:    Value          Size Type    Bind   Vis      Ndx Name
     0: 0000000000000000     0 NOTYPE  LOCAL  DEFAULT  UND
     1: 0000000000000000     0 FILE    LOCAL  DEFAULT  ABS static-example.c
     2: 0000000000000000     0 SECTION LOCAL  DEFAULT    1
     3: 0000000000000000     0 SECTION LOCAL  DEFAULT    3
     4: 0000000000000000     0 SECTION LOCAL  DEFAULT    4
     5: 0000000000000000     4 OBJECT  LOCAL  DEFAULT    4 invocations
     6: 0000000000000000     0 SECTION LOCAL  DEFAULT    5
     7: 0000000000000004     4 OBJECT  LOCAL  DEFAULT    4 first_time.2253
     8: 0000000000000000     0 SECTION LOCAL  DEFAULT    7
     9: 0000000000000000     0 SECTION LOCAL  DEFAULT    8
    10: 0000000000000000     0 SECTION LOCAL  DEFAULT    6
                                    CT GLOBAL DEFAULT  COM not_defined_here
                                    CT GLOBAL DEFAULT    3 message
                                       GLOBAL DEFAULT    1 hello_world
                                    PE GLOBAL DEFAULT  UND _GLOBAL_OFFSET_TABLE_
                                    PE GLOBAL DEFAULT  UND puts
                                    PE GLOBAL DEFAULT  UND stderr
                                    PE GLOBAL DEFAULT  UND fprintf
                                    PE GLOBAL DEFAULT  UND fwrite
                                       GLOBAL DEFAULT    1 main
```



Symbol bindings

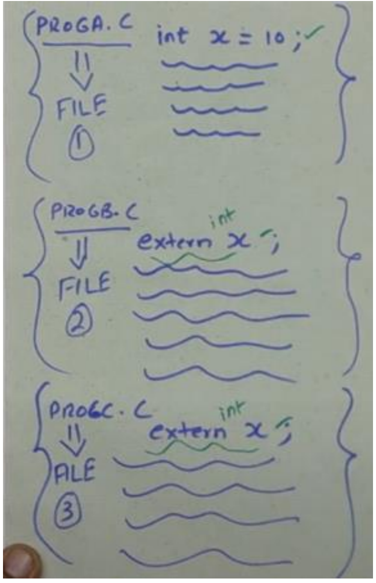


```
lecture2 > C static-example.c > ...
#include <stdio.h>

int not_defined_here;
char message[] = "hello world";
static int invocations = 0;

void hello_world(int increment) {
  static int first_time = 0;
  if (increment >= 0) {
    puts(message);
    invocations++;
    fprintf(stderr, "I have printed to the screen %d
    invocations);

    hello_world(increment-1);
  }
  if (first_time == 0) {
    fprintf(stderr, "this is the end of the first in
    hello_world\n");
    first_time++;
  }
}

int main() { hello_wo
```



Symbols live in sections

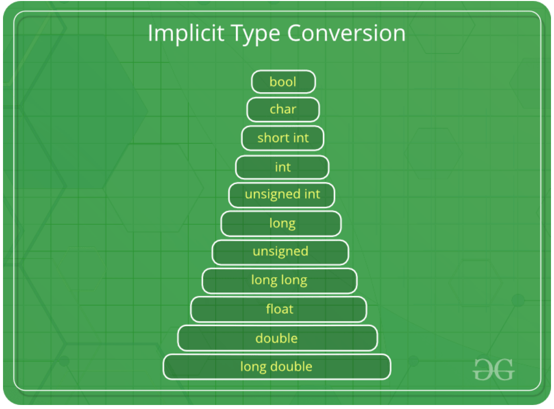
Extern
Applies to variables and functions. Extends visibility of them. Doesn't allocate memory.



File 2 and 3 simply declare X that it exists.

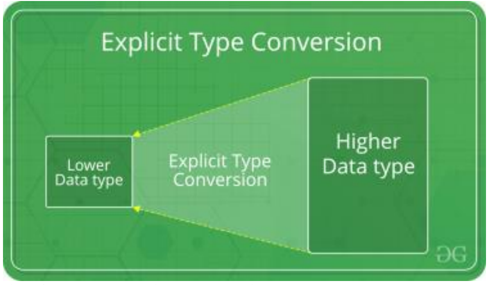
- If no datatype given to variable it is automatically an integer
- Signed is default modifier for char and int
- Can't use modifier (signed/short) on floats
- Can only use long modifier on double
- Doubles have double the precision of float.
- Chars are in range of -128 to +127 or 0 to 255

Implicit type conversion



Automatic type conversion is done by compiler. Happens when in an expression more than one data type is present. Promotion takes place to avoid loss of data.
Data can still be lost on implicit conversion for example signed to unsigned, long long to float.

Explicit type conversion



If you're converting a float to an int for example.

Pointers

05 November 2021 15:43

https://www.geeksforgeeks.org/complicated-declarations-in-c/
come back after finishing

Data Types

05 November 2021 16:25

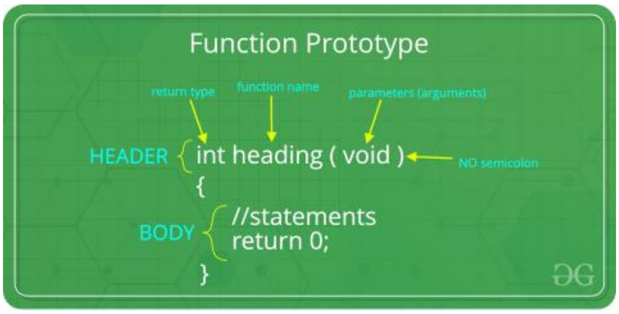
Functions

06 November 2021 19:19

Prototype

1) It tells the return type of the data that the function will return.
2) It tells the number of arguments passed to the function.
3) It tells the data types of each of the passed arguments.
4) Also it tells the order in which the arguments are passed to the function.

Good to have prototype to avoid compiler assuming function is an int or not.



Function Prototype

```
int heading ( void )
{
    //statements
    return 0;
}
```

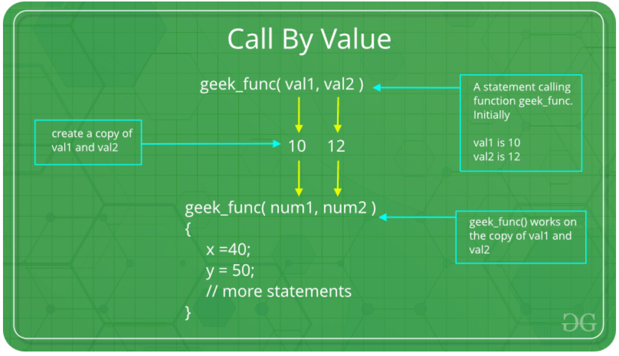
- Pass by value - (value of parameter copied and passed to functions formal parameters. Changes to variable inside function doesn't affect actual parameter of caller.
- Pass by reference - both actual and formal parameters refer to same location.
- In C parameter always by value but can use pointers to pass by reference like so:
  - `void fun(int *ptr)`
  - `{`
  - `    *ptr = 30;`
  - `}`
- C can return any type but function or array (unless we use pointer)
- Empty parameter list means function can be called with any parameters
- Can have main function with or without parameters.
  - With params like so: main(int argc, char * const argv[])
  - Saves every group of character in an array called argv.
- No guaranteed function evaluation order. Can vary compiler to compiler.
- Function overloading is where functions of same name can have different signatures/arguments. Not possible in C but there are some hacks

  `int foo(void * arg1, int arg2);`
  - Arg1 a pointer to any type of variable.
  - Arg2 representative of which data type (0 for struct1, 2 for struct2
  - Then if statement saying if 0 then do this etc.
- Can return multiple args by returning pointer to a struct/array etc
- Functions global by default.
  - Can also be static meaning access to function restricted to file it's in.
- Nested functions not supported by C.
  - Inner function can't access local variable of surrounding block, can only access global variables.
  - C only has 2 nested scope: local and global
  - Can be done with extension of GNU compiler.

Exiting

- Exit()
  - Flushes unwritten buffered data
  - Closes all open files
  - Removes temporary files
  - Returns integer exit status to OS
  - Process parent sent SIGCHILD
  - _EXIT() in C++ doesn't do any flushing
- Abort()
  - May not close files or buffer streams
  - Doesn't call atexit()
  - Uses SIGABRT signal.
- Assert()
  - Not sure

Parameter Passing



Call By Value

```
geek_func( val1, val2 )
10  12
geek_func( num1, num2 )
{
    x =40;
    y = 50;
    // more statements
}
```

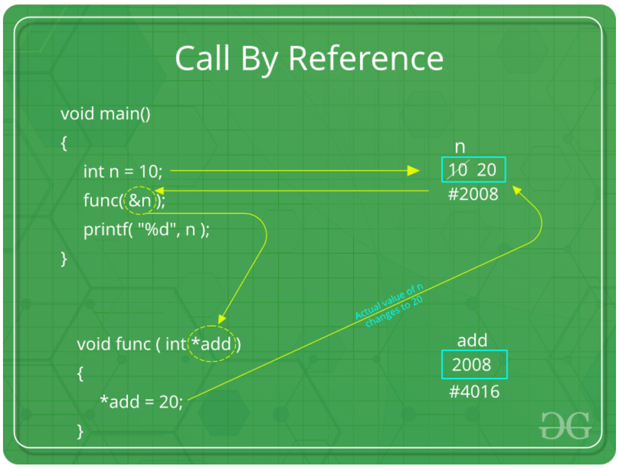
A statement calling function geek_func. Initially

val1 is 10
val2 is 12

create a copy of val1 and val2

geek_func() works on the copy of val1 and val2

Uses in mode semantics. Changes made to formal parameter not transmitted back. Effects only separate storage location.



Call By Reference

```
void main()
{
    int n = 10;
    func( &n );
    printf( "%d", n );
}

void func ( int *add )
{
    *add = 20;
}
```

Uses in/out mode semantics. Changes made to formal parameter not transmitted back through parameter passing but reflected in actual parameter as it receives a reference/pointer to it.

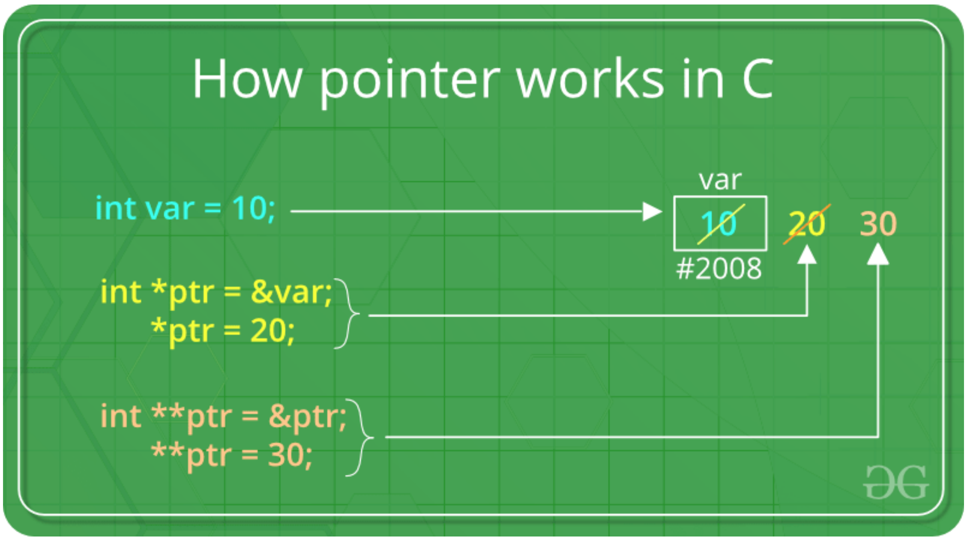
Pass by result - out/mode. Before control transferred back to caller, value of formal parameter is transmitted back to actual parameter. Implemented by copy.

Pass by value-result - in/out mode. Combination of pass-by-value and -result. Before control back to caller, value of formal parameter is transferred back to actual parameter.

Pass by name - Symbolic name of variable passed allowing it to be accessed.

Pointers

08 November 2021 10:20



How pointer works in C

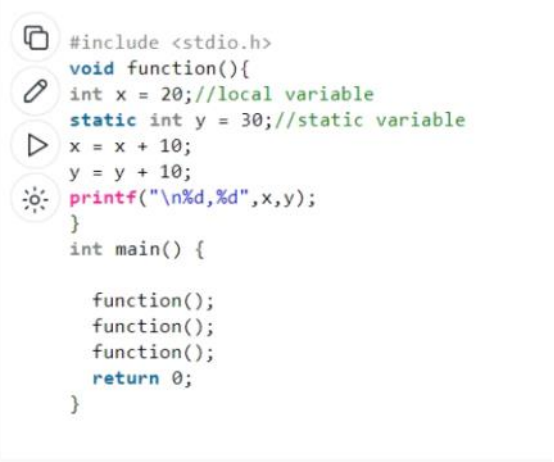
```
int var = 10;
int *ptr = &var;
    *ptr = 20;
int **ptr = &ptr;
    **ptr = 30;
```

Pointers store address/memory location of a variable.

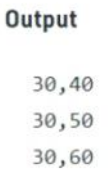
Questions

06 November 2021 22:32

Basics
Variables
Data types
Function
Pointers
Enum, Struct and Union
Memory management
Arrays and string
Control statement
I/o
File handling
Operators
Puzzles - relevant to us

- 

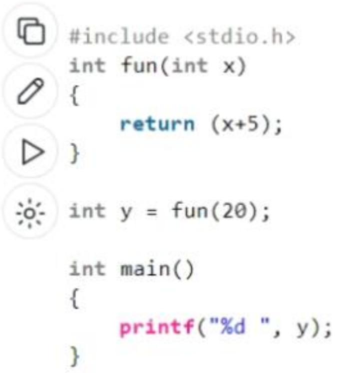
```
#include <stdio.h>
void function(){
int x = 20;//local variable
static int y = 30;//static variable
x = x + 10;
y = y + 10;
printf("\n%d,%d",x,y);
}
int main() {

  function();
  function();
  function();
  return 0;
}
```



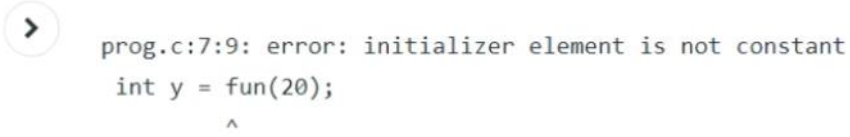
Output

```
30,40
30,50
30,60
```

- Why called static if it changes?

- 

```
#include <stdio.h>
int fun(int x)
{
    return (x+5);
}

int y = fun(20);

int main()
{
    printf("%d ", y);
}
```

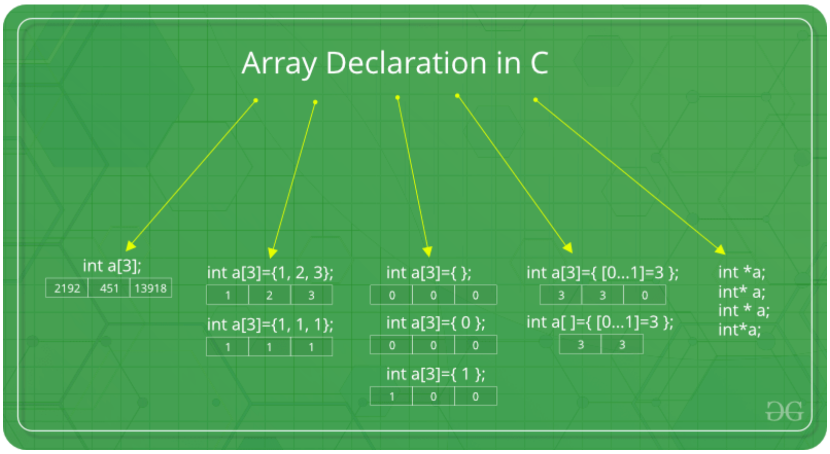


```
prog.c:7:9: error: initializer element is not constant
  int y = fun(20);
          ^
```

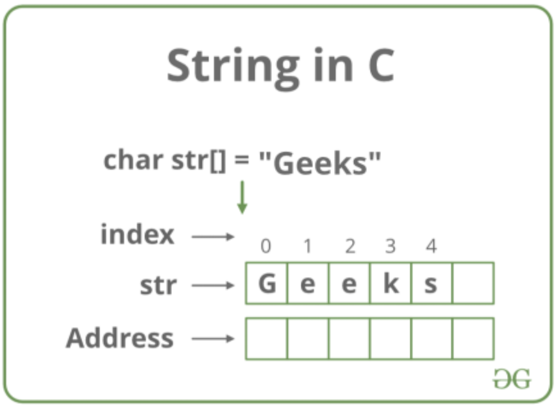
- CPU calls data from RAM by giving the address of the location to MAR (Memory Address Register). The location is found and the data is transferred to MDR (Memory Data Register). This data is recorded in one of the Registers in the Processor for further processing. That’s why size of Data Bus determines the size of Registers in Processor.
- size_t void fun(int arr[], size_t arr_size)   (inbuilt to array?)
- https://www.geeksforgeeks.org/importance-of-function-prototype-in-c/
  - Function prototype tells program what a functions paramater datatypes are and what its return type is too.
  - <returnType> <functoinName>(<params>);
- https://www.geeksforgeeks.org/how-to-count-variable-numbers-of-arguments-in-c/
  - Example of this please. What does va_start do
- Signals like SIGABRT. Whats the data flow like for that?
- Asser()
- Explain syntax in callback function

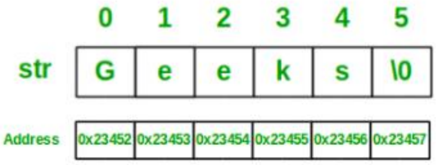
Arrays and Strings

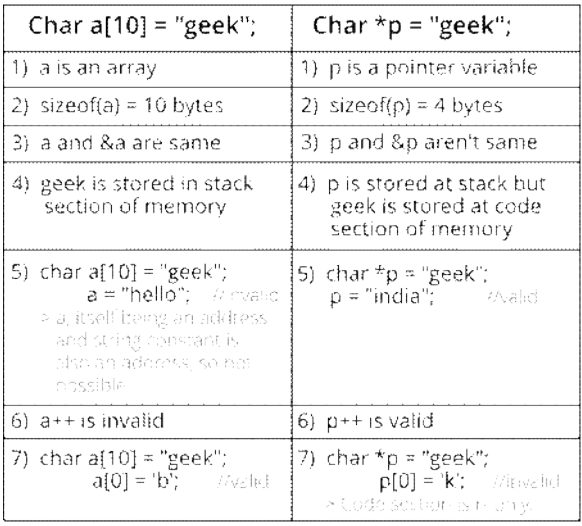
08 November 2021 11:50



- Cannot initialize variable sized array   int arr[M][M] = {0}; // Trying to initialize all values as 0







Char a[10] = "geek";	Char *p = "geek";
1) a is an array	1) p is a pointer variable
2) sizeof(a) = 10 bytes	2) sizeof(p) = 4 bytes
3) a and &a are same	3) p and &p aren't same
4) geek is stored in stack section of memory	4) p is stored at stack but geek is stored at code section of memory

5) char a[10] = "geek";
a = "hello"; // invalid
> a, itself being an address and string constant is also an address, so not possible
	5) char *p = "geek";
	p = "india"; //valid

6) a++ is invalid	6) p++ is valid

7) char a[10] = "geek";
a[0] = 'b'; //valid
	7) char *p = "geek";
	p[0] = 'k'; //invalid
	> Code section is r only