structure of a program :

- A program is a set of functions.
- A function is a closed entity that has :
  - Input data : the parameters.
  - A piece of output data : the return value.
- A function is made of :
  - instructions of our the CPU.
  - local data : the variable.

accolades {} : représente un scope (la portés des choses qui seront a l'intérieur)

~~Avantages~~

Why C :

- multiplateformes.
- Gestion de la mémoire.
- Exécution rapide.
- langage compilé.

A system call is a càll to a function that is not put of the application but inside the Kernel.
the Kernel is a software layer that provides you some basic functionalities to abstract the hardware to you. Roughly the Kernel is something that turns your hardware into software.

Variable :

- Un bit c'est just un petit espace en mémoire qui va valoir soit 0 soit 1
- Un octet c'est un espace avec 8 bits ce qui va nous permettre donc de faire plein de combinaisons de valeur.

le binaire :

La base ce n'est qu'une manière de représenter une valeur mais la valeur elle n'a pas changé. aff-bin(c) //affiche la valeur binaire.

Stack : c'est la partie de la mémoire qui va s'agrandir dans votre programme au fur et a mesure que vous demandez des Variable ou que vous appelez des fonctions.

Recursive : function calling itself

function calling itself is called recursive.
calling a function from the definition of a same function is called Recursion.

stack memory is a memory usage mechanism that allows the system memory to be used as temporary data storage that behaves as first-in-last-out buffer.

A buffer : contains data that is stored for a short amount of time, typically in the computer's memory (RAM). the prupose of a buffer is to hold data right before it is used.

Buffering is used to improve several other areas of computer performance as well. Most hard disks use buffer to enable more efficient access to the data on the disk

memory : an array of bytes within RAM

memory block : a single unit (byte) within memory. Used to hold some value.

memory address : the address of where a memory block is located

---

C is a compiled language. This means we take source code, and and a tool called a compiler translates the code to 1's and 0's that the computer (CPU) understands.

gcc - This our compiler (clang is another popular free compiler) we pass in our source files (.c files) to the compiler. The output (executable binary) from gcc follows the -o flag

Mo Tu We Th Fr Sa Su

Memo No. 1337
Date 31/08/2022

C03 :

1) strcmp, strncmp : Comparaison de deux chaines.

```
int strcmp(const char *s1, const char *s2)
```

* La fonction strcmp() compare les deux chaines s1 et s2. Elle renvoie un entier négatif, null, ou positif, si s1 et s2 respectivement inférieur, égale ou supérieur à s2.

- 0, if the s1 and s2 are equal;
- a negative value if s1 is less than s2;
- a positive value if s1 is greater than s2

2)

* the strncmp() function is similar, except it only compares the first (at most) n bytes of s1 and s2

Mo Tu We Th Fr Sa Su

Memo No. C03
Date / /

III) strcat, strncat : Concatenate two strings.

Char *strcat(char *dest, char *src)

The strcat() function appends the src string to the dest string.

la fonction strcat() ajoute la chaine src à la fin de la chaine dest en écrasant null (\0) à la fin de dest, puis en ajoutant un nouveau caractère null final. Les chaines ne doivent pas se chevaucher, et la chaine dest doit être assez grande pour accueillir le résultat.

Mo Tu We Th Fr Sa Su

Memo No. C03
Date / /

VII) strstr : Rechercher une sous-chaine.

La fonction strstr recherche la premiére occurence d'une sous-chaine (paramétre substring) dans la chaine de caractéres principale (paramétre full string)

```
char *pt_strstr(char *str, char, *to_find)
```

- str : La chaine de caractéres dans laquelle effectuer la recherche.
- to_find : La sous-chaine à rechecher dans la chaine principale.

Ces fonctions renvoient sur le début de la sous-chaine, ou NULL si celle-ci n'est pas trouvée.

Mo Tu We Th Fr Sa Su

Memo No. C arguments
Date 01/09/2022

Command line Arguments in C :

The arguments passed from command line are called command line arguments. These arguments are handled by main() function.

To pass command line arguments, we typically define main() with two arguments : first argument is the number of command line arguments and second is list of command-line arguments.

```
int main (int argc, char *argv[])
```

or

```
int main (int argc, char **argv)
```

- argc (ARGument count) is int and stores number of command-line arguments passed by the user including the name of the program

Mo Tu We Th Fr Sa Su

Memo No.
Date / /

So if we pass a value to program, value of argc would be 2 (one for argument and one from program name)

- the value of argc should be non negative
- argv (ARGument Vector) is array of character pointers listing all the arguments.
- If argc is greater than zero, the array elements from argv[0] to argv[argc-1] will contain pointers to strings.
- Argv[0] is the name of the program, After that till argv[argc-1] every element is command-line arguments.

Mo Tu We Th Fr Sa Su

Memo No. C02:
Date 03/04/2022

C02:

strcpy(): function copies the string pointed by source to destination

```
char *ft_strcpy(char *dest, char *src)
```

strncpy(): function copies upto n characters from the string pointed to src to dest

- in case where the length of src is less than that of n, the reminder of dest will be with null bytes ('\0')

```
char *ft_strncpy(char *dest, char *src, unsigned int n)
```

strlcpy(): function copies upto size-1 characters from src to dest

Memo No. 002
Date 03/09/2022

unsigned int ft_strlcpy(char *dest, char *src, unsigned int size);

The strlcpy() and strlcat() functions return the total length of the string they tried to created. for strlcpy() that means the length of src.
for strlcat() that means the intial length of dest plus the length of src.

Memo No. black-notebook
Date 04/09/2022

Rappel : C

Variable : for every computer application we must store the information at a particular location.

" every memory location is identified by an addresse "

Ces Variable ont des types permettant de définir leur taille (en octets).

Dans une variable, on ne peut que stocker un nombre, dont la limite inférieure et supérieure depend du type.

Datatype : is a representation of data.

" How much memory write is required to allocate and what type of data is allowed to store "

" We can say that data types are used to tell the Variables the type of data it can store "

Mo Tu We Th Fr Sa Su

Memo No.
Date / /

Classification of data :

* primitive : int . char . float . void
* Derived : Array . string - pointer.
* User - define : structure - Union - type def - enum

- Les char, short, int, long et leurs unsigned respectifs ne peuvent contenir que des entiers (2, 6, 42, 244, ...).
- Les float, double et long double peuvent contenir des nombre a virgule flottante (2.0, 6.21, -3,4)
- Le mot-clé void signifie l'absence de Variable, par exemple, si une fonction ne renvoie rien ou si elle ne prend pas de paramètre
- Il existe des types pré-définis qui portent un nom permettant de connaitre leurs rotes par exemple : size_t contient la taille d'un tableau (ssize_t servira à parcourir un tableau)

Memo No. _____
Date / /

Mo Tu We Th Fr Sa Su

Type: char: ✓

Nombre d'octets en mémoire : 1 octet

Nombre de bits en mémoire : 8 bits

limite inférieure : -128.

limite supérieure : 127.

char: → signed → 1 byte → $\frac{2^8}{2}$ → $\frac{256}{2}$ → [-128 to +127]

→ unsigned → 1 byte → 2^8 → 256 → [0 to 255]

"using character system you can represent an entire language (letters - digits - symbols)"

ASCII character System : American standard code for information Interchange.

/A:65 - Z:90 / a:97 - z:122 / 0:48 - 9:57 /

representation of one language.

- character data type representation in intgers because we are reprenting a character using character system. : representing all the symbols

Mo Tu We Th Fr Sa Su

Memo No.
Date / /

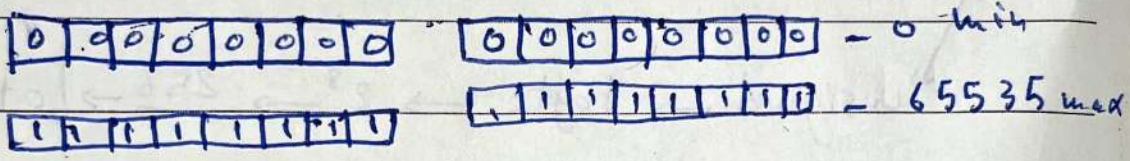
of one programing language, Using constant Intger values.

Singed short → 2 bytes.

Unsigned short → 2 bytes → 1 byte [8 bits]

↓

2^{16} → 65536.



int : 4 octets : -2147483648 + 47

long : 4 octets : -9,223,372,036,854

float : 4 octets : -3.4×10^{-38}, 3.4×10^{38}

The key difference between int and long is that int is 32 bits in width while long is 64 bits in width.

Memo No.
Date / /

la compilation :

On compile avec gcc (alias cc).
Le compilateur génère alors un fichier (executable) a.out. Il est possible de modifier le nom du fichier grace à l'option -o.

gcc -o nom-executable mon-code.c

S'il y a une ou plusieurs erreurs, elles seront affichées et la compilation s'arrêtera.
Cependant, certaines "erreurs" peuvent ne pas empêcher la compilation, on les appelle les Warnings
Il est malgré tout très important de corriger ces petites erreurs car la plupart du temps, elle perturberont le fonctionnement normal du programme.

Mo Tu We Th Fr Sa Su

Memo No.
Date / /

pour afficher plus de warings, il est possible (et conseillé) d'ajouter des options (dites flags de compilation) :

- -W : Désactive tout les warings (déconseillé)
- -Wextra : Affiche encore plus de warings (détaillés)
- -Wall : affiche plus warnings
- -Werror : considère les warings comme des erreurs et cesse la compilation.
- -ansi : affiche warnings en cas de non respect de la norme ISO C90.

Les tableaux : ✓

- Si l'on a besoin de stocker plusieurs variables, on peut utiliser un tableau. Un tableau, c'est plusieurs variables d'un même type les unes à la suite des autres.

```
type nom[nombre_de_cases];
```

Mo Tu We Th Fr Sa Su

Memo No.
Date / /

Les chaines de caractères: ✓

- On appelle une chaine de caractères un tableau de type char qui contient des lettres (leurs valeurs ASCII) et qui se termine par un '\0';
- On peut peut envoyer une chaine de caractère à une fonction en utilisant des double quotes (") "hello".
Les simple quots (') sont utilisées pour les caractère seuls.
- La fonction à laquelle on aura envoyé la chaine de caracters grace à des doubles quotes prendra alors en ~~poem~~ paramètre un pointeur vers un char. C'est la pointeur vers la première case du tableau qui contient une chaine de caractères.

Les pointeurs: ✓

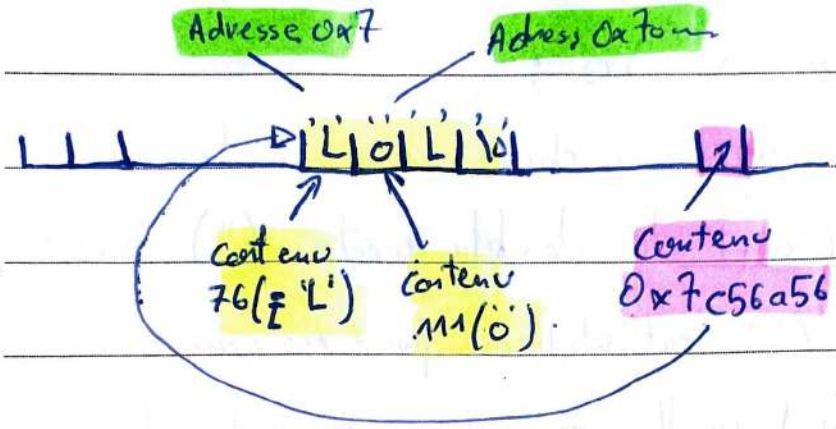
- Chaque variable, pour être utilisée, doit être stockée quelque part dans la mémoire. On dit que cette variable se trouve à une adresse en mémoire.

Mo Tu We Th Fr Sa Su

Memo No.
Date / /

Cette adresse est un numéro que l'on exprime en général en hexadécimal (type 0x7F6a8b).

- Un pointeur est une variable qui contient l'adresse d'une autre variable.



- Sur le schéma, les petites cases représentent les variables stockées dans la mémoire.

En dessous, le contenu des cases du tableau, et au dessus, l'adresse ou elles sont stockées.

A un autre endroit dans la mémoire, une autre variable est stockée. Elle a pour contenu l'adresse de la première case du tableau.

Memo No.
Date / /

"Un pointeur est une variable qui peut contenir l'adresse d'une autre variable si un pointeur P contient l'adresse d'une variable A, on dit que « P point sur A »

Les opérations de base à lors de travail avec des pointeurs nous avons besoin :

& : pour obtenir l'adresse d'une variable.

* : pour accéder au contenu d'une adresse.

```
Ex : int a = 2;
     int *p = &a;
```

- Dans le passage par valeur, une copie des arguments réels est transmise aux arguments formels respectifs.
- Dans le passage par adresse, l'emplacement (adresse) des arguments réels est transmis à des arguments formels, toute modification apportée

Mo Tu We Th Fr Sa Su

Memo No.
Date / /

aux arguments formels se réfèra également dans les argument réels.

Resumé : chaque Variable, pour être utilisée, doit être stockée quelque part dans la mémoire. On dit que cette Variable se trouve à une adresse en mémoire cette adresse et un numero qu'on exprime en général en hexadecimal (type 0x76a8d)

• Un pointeur et une Variable qui contient l'adresse d'une autre Variable.

Mo Tu We Th Fr Sa Su

Memo No.
Date / /

La récursivité :

- Un fonction récursive est une fonction qui s'appelle elle-même.
- La récursivité peut s'utiliser en remplacement d'une boucle.

Exemple :

Ces deux fonctions ont le même comportement, mais l'une utilise une boucle et l'autre est récursive.

```
void    print_c_while (int n)
{   int     i;
    i = 0;
    while (i<n)
    {   print_char('c');
        ++i;
    }

Void print_c_recursive (int n, int i).
{   void if (i<n) {
        Print_char('c');
        Print_c_recursive (n, i++);
    }
}
```

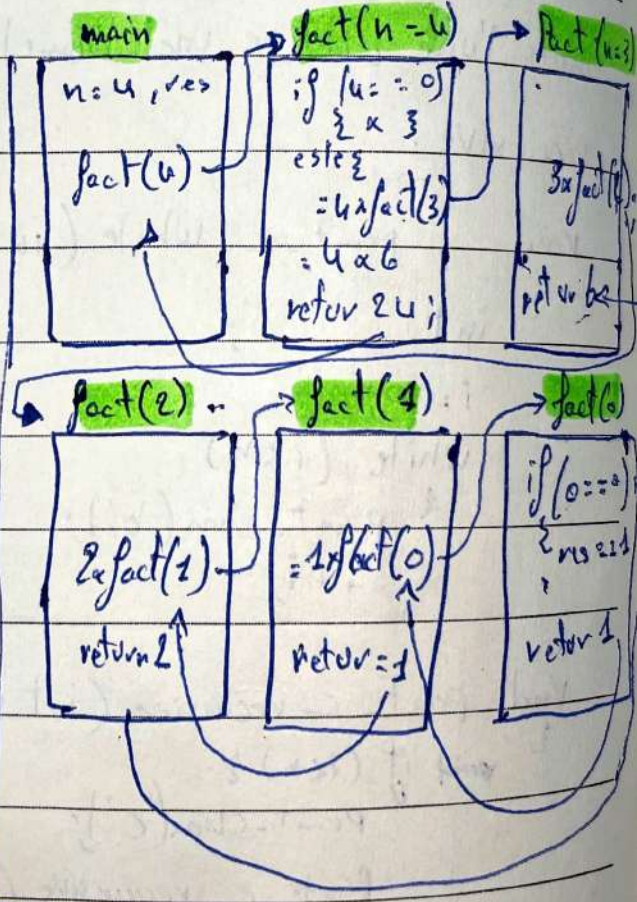
Memo No.
Date

"Recursion and iteration are both different ways to execute a set of instructions repeatedly. The main difference between these two is that in recursion, we use function calls to execute the statements repeatedly inside the function body. While in iteration, we use loops like "for" and "while" to do same."

Example:

```
main()
{
int n, res;
pf("enter number");
sf("%d", &n);
res = fact(n);
Printf("Result: %d");
}

fact(int n)
{
  int res;
  if(n==0)
  { res = 1;
  }
  else
  {
    res = n * fact(n-1);
  }
  return res;
}
```



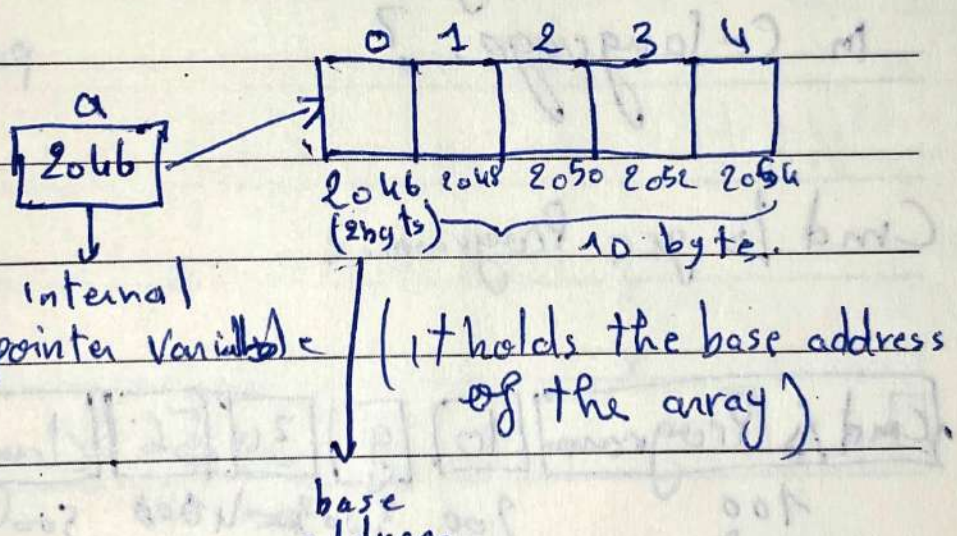
Mo Tu We Th Fr Sa Su

Memo No.
Date / /

Array: array is an aggregate data or we can call it as derived data type.

data.type Identity [size].

Ex: int a[5];



~~int~~ local declarations:

int arr[5]; → | G.V | G.V | G.V | G.V | G.V |

int arr[5] = {10, 20, 30, 40, 50}; → | 10 | 20 | 30 | 40 | 50 |

Int arr[5] = {10, 20}; → | 10 | 20 | 0 | 0 | 0 |

Global Declaration:

Int arr[5]; → | 0 | 0 | 0 | 0 | 0 |;

Mo Tu We Th Fr Sa Su

Memo No.
Date / /

Command line Args :-

Input Value.

CUI [Dos]

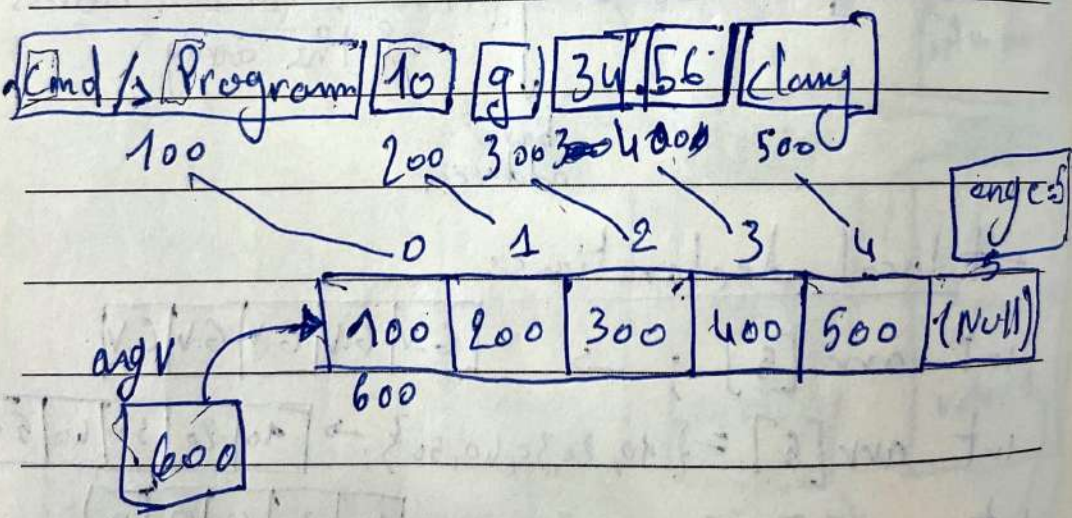
How memory allocates to Command line Arguements in C-language ?

OS

CUI → programs

GUI → End...

Cmd /> gcc Programm.c

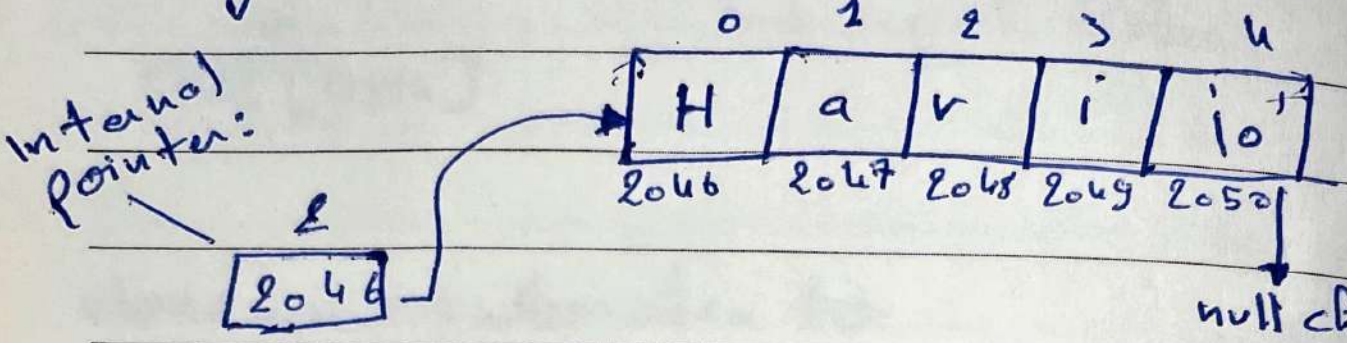


argv[0][0]

Mo Tu We Th Fr Sa Su

Memo No.
Date / /

String: it is one-dimensional character array.



char s2[4] : {'o', 'n', 'e'}; ASCII - 0

"%s" → format specifier.

→ read and write string Elements.

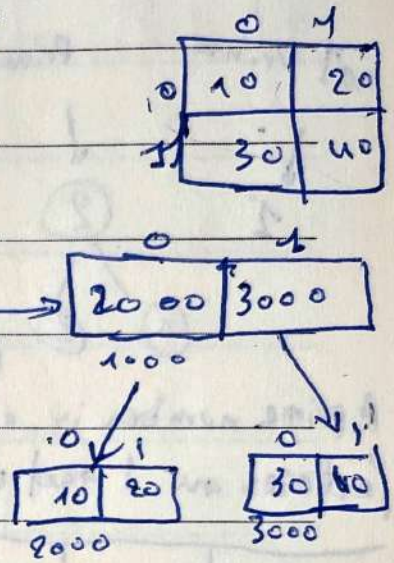
→ No need to use iterator [loop]

Mo Tu We Th Fr Sa Su

Memo No. _____
Date / /

Access Elements of 2 dimensional Array using pointers.

```
int main (){
  int a[2][2] = {{10,20},{30,40}};
  int i, j;
  for (i=0; i<2; i++){
    for (j=0; j<2; j++){
      pf("%d" a[i][j])
      pf("%d" *(*(a+i)+j));
```



Pointers:

Typed: Points to specific type of data.

int * → int data

double * → double data.

struct Emp * → Employee data.

Untyped: Can points to any data (Generic Pointer)

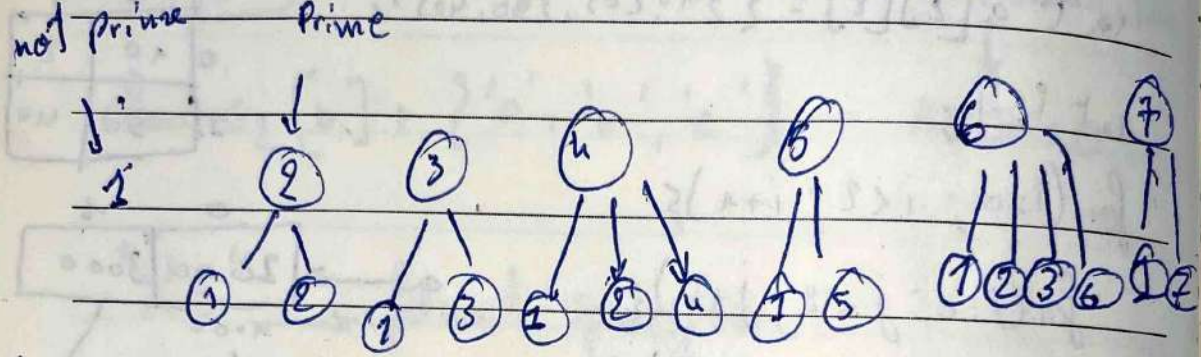
void * → Any data.

Mo Tu We Th Fr Sa Su

Memo No. C05
Date / /

C05:

Prime numbers: → Natural number (1, 2, 3, 4...)
→ Divisible by exactly 2 natural numbers. (itself & 1)



A prime number is a whole number greater than 1 whose only factors are 1 and itself.

- **Rational & irrational numbers:**

any integer is a rational number.

* $1 = \frac{1}{1} = \frac{-2}{-2} = \frac{10000}{10000}$
* $-7 = \frac{7}{1} = \frac{7}{-1} = \frac{-14}{2}$

$3.75 = \frac{375}{100} = \frac{750}{200} = \frac{15}{4} = \frac{-30}{-8}$

$0.3 = \frac{1}{3}$

$0.6 = \frac{2}{3}$

Mo Tu We Th Fr Sa Su

Memo No.
Date / /

Irrational numbers:

$\pi = 3.14...$

$e = 2.7...$

$\sqrt{2} = 1.414$

$\varphi = 1.618$

Sqrt:

number * number

$\sqrt{9} = 3$ / $3 * 3 = 9$

$\sqrt{16} = 4$ / $4 * 4 = 16$

```
int i;
while (i <= nb){
    if (nb % i == 0){
        c++;
    i++;}
if (c == 2)
    return (1);
```

Mo Tu We Th Fr Sa Su

Memo No.
Date / /

Structures : User-defined datatype

using structure we can define a data type which holds More than one elements of different types.

Syntaxe :

```
struct <Identity>
{ data_type elem1;
  data_type elem2;
};
```

Ex : Employee

```
struct Employee
{ int eno;
  char ename[20];
  float esal;
};
```

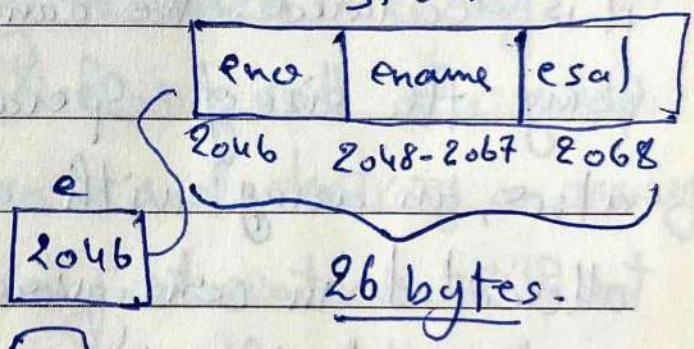
Mo Tu We Th Fr Sa Su

Memo No.
Date / /

Array Vs structure.

int arr[5] = {10, 20, 30, 40, 50}

struct Emp
{
int eno
char ename[20];
float esal
};
main( ) {
int a
struct Emp e;



Access:
e.Eno
e.ename
e.esal

Mo Tu We Th Fr Sa Su

Memo No.
Date / /

DMA: Dynamic memory Allocation

When you declare a variable using a basic data type, the C compiles automatically allocates memory space for the variable in a pool of memory called the Stack.

for example, a float variable takes typically 4 bytes (according to the platform) when it is declared. We can verify this information using the sizeof operator. also, an array with a specified size is allocated in contigvous blocks of memory. each block has the size for one element.

When declaring a basic data type or an array, the memory is automatically managed. However there is a process for allocating memory inc which will permit you to implement a

Mo Tu We Th Fr Sa Su

Memo No.
Date / /

program in which the array size is undecided until you run your program (runtime). this process is called "Dynamic memory allocation".

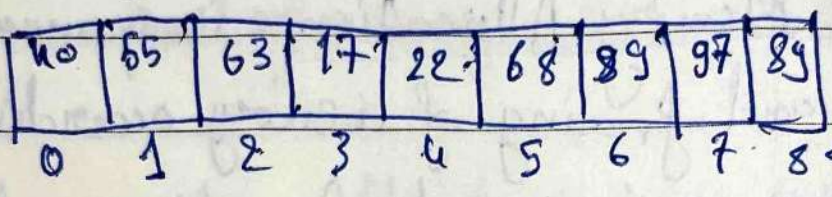
Dynamic Memory Allocation is manual allocation and freeing of memory according to your programming needs. Dynamic memory is managed and served with pointers that point to the newly allocated memory space in an area wich we call the heap. Now you can create and destroy an array of elements dynamically at runtime without any problems. to sum up, the automatic management uses the stack, and the C Dynamic Memory Allocation uses the heap.

The <stdlib.h> library has functions responsible for Dynamic Memory Management

Mo Tu We Th Fr Sa Su

Memo No.
Date / /

Since C is structured language. It has some fixed rules for programming. One of them includes changing the size of an array. An array is a collection of items stored at contiguous memory locations.



As it can be seen that the length (size) of the array above made is 9, but what if there is a requirement to change this length (size)?

Eg:

- if there is a situation where only 5 elements are needed to be entered in this array. In this case, the remaining 4 indices are just wasting memory in this array. So there is requirement to lessen the length (size) of the array from 9 to 5.

Memo No. _____
Date / /

Mo Tu We Th Fr Sa Su

• Take another situation. In this, there is an array of 9 elements with all 9 indices filled. but there is a need to enter 3 more elements in this array in this case 3 indices more are required. so the length (size) of the array needs to be changed from 9 to 12.

this ~~prod~~ procedure is referred to as Dynamic Memory Allocation in C.

2e) Therefore, C Dynamic Memory Allocation can be defined as a procedure in which the size of a data structure (like Array) is changed during the runtime.

C provides some functions to achive these tasks. there are 4 library functions provided by c defined under <stdlib.h> header file to facilitate dynamic memory allocation in C programming.

Mo Tu We Th Fr Sa Su

Memo No.
Date / /

They are:

1. malloc ()
2. calloc ()
3. free ().
4. realloc()

Function	Purpose.
malloc()	Allocates the memory of requested size and returns the pointer to the first byte of allocated space.
calloc()	Allocates the space for elements of an array. Initializes the elements to zero and returns pointer to the memory.
realloc().	it is used to modify the size of previously allocated memory space.

Memo No. _____
Date / /

free()	Frees or empties the previously allocated memory space

malloc() function in C:

The C malloc() function stands for memory allocation. It is a function which is used to allocate a block of memory dynamically. It reseves memory space of specified size and returns the null pointer pointing to the memory location of the pointer returned is usually of type void. It means that we can assign C malloc() function to any pointer.

```
ptr = (cast-type*) malloc (byte-size);
```

Mo Tu We Th Fr Sa Su

Memo No.
Date / /

- ptr is a pointer of cast_type.
- the C malloc() function returns a pointer to the allocated memory of byte_size.

Ex:

```
ptr = (int *) malloc (50)
```

When this statement is successfully executed, a memory space of 50 bytes is reserved. the address of the first byte of reserved space is assigned to the pointer ptr of type int.

Memo No.
Date / /

Static Memory vs Dynamic Memory

Compile time → Fixed Memory

Runtime → Initial memory → insert / delete → grow / shrink.

Multiple Ways to store Program Data:

- Static global data:
  - Fixed size at compile-time.
  - Entire lifetime of the program (load from executable)
  - Portion is read-only. (eg: String literals)

Mo Tu We Th Fr Sa Su

Memo No.
Date / /

- Stack-allocated data.
  - local/temporary variables.
  - Can be dynamically sized (in some versions of C)
  - Known lifetime (deallocated on return)

- Dynamic (heap) data.
  - Size known only at runtime (based on user-input)
  - Lifetime known only at runtime (long-lived data structures)

Memo No.
Date / /

RAM :

We have four major memory sections in RAM.

1. Text or code section.
2. global / static section.
3. heap section.
4. stack section.

1: In text section we have the read-only executable programming instructions which cannot be modified.

2: In global section we will have all the global and static variables.

3: Dynamic memory allocation happens in heap memory section. that is, if we allocate memory using malloc or calloc at run time.

Mo Tu We Th Fr Sa Su

Memo No.
Date / /

4. all the local variables and function calls are created in the stack section.

What is a stack:

A stack is a special area of computer's memory which stores temporary variables created by a function. In stack, variables are declared, stored and initialized during runtime.

It is a temporary storage memory. When the computing task is complete, the memory of the variable will be automatically erased. the stack section mostly contains methods, local variable, and reference variables.

What is Heap ?:

The heap is a memory used by programming languages to store global variables. By default, at global variable are stored in heap memory space. It supports Dynamic memory allocation.

Mo Tu We Th Fr Sa Su

Memo No.
Date / /

The heap is not managed automatically for you and is not as tightly managed by the CPU. It is more like a free-floating region of memory.

Advantages of using stack:

- helps you to manage the data in a Last in First out (LIFO) method which is not possible with Linked list and array.
- When a function is called the local variables are stored in a stack in a stack, and it is automatically destroyed once returned.
- A stack is used when a variable is not used outside that function.
- stack automatically cleans up the object.
- Not easily corrupted.
- Variables cannot be resized.

Mo Tu We Th Fr Sa Su

Memo No.
Date / /

Advantages of Using Heap:

- Heap helps you to find the greatest and minimum number.
- Garbage collection runs on the heap memory to free the memory used by the object.
- Heap method also used in the Priority Queue.
- It allows you to access variables globally.
- Heap doesn't have any limit on memory size.

Disadvantages of using stack

- stack memory is very limited.
- Creating too many objects on the stack can increas the risk of stack overflow.
- Random access is not possible.

Memo No.
Date / /

- Variable storage will be over written, which sometimes leads to undefined behavior of the function or program.
- The stack will fall outside of the memory area. which might lead to an abnormal termination.

Disadvantages of using Heap.

- It can provide the maximum memory an OS can provide.
- It takes more time to compute.
- Memory management is more complicated in heap memory as it is used globally.
- It takes too much time in execution compared to the stack.

Memo No.
Date / /

When to use the Heap or stack?

You should use heap when you require to allocate a large block of memory. For example, you want to create a large size array or big structure to keep that variable around a long time then you should allocate it on the heap.

However, if you are working with relatively small variables that are only required until the function using them is alive. then you need to use the stack, which is faster and easier.

Memo No. _____
Date / /

malloc: is a library function that allows C to allocate memory dynamically from the heap. The heap is an area of memory where something is stored.

How to use **Malloc**:

malloc() allocates memory of a requested size and returns a pointer to the beginning of the allocated block. To hold this returned pointer, we must create a variable.

malloc() is a part of stdlib.h and to be able to use it you need to use #include <stdlib.h>

```
int *arrayPtr;
```

Mo Tu We Th Fr Sa Su

Memo No.
Date / /

Unlike other languages, C does not known the data type it is allocating memory for; it needs to be told. Luckily, C has a function called sizeof() that we can use.

```
arrayPtr = (int *)malloc(10*sizeof(i
```

This statement used malloc to set aside memory for an array of 10 integers. As sizes can change between computers, it is important to use the sizeof() function to calculate the size on the current computer.

Any memory allocated during the program's execution will need to be freed before the program closes. To free memory, we can use the free() function.

Memo No.
Date / /

Free (arrayPtr);

This statement will deallocate the memory previously allocated. C does not come with a garbage collector like some other languages, such as Java. As a result, memory not properly freed will continue to be allocated after the program is closed.

A Review:

- Malloc is used for dynamic memory allocation and is useful when you don't know the amount of memory needed during compile time.
- Allocating memory allows objects to exist beyond the scope of the current block.

Mo Tu We Th Fr Sa Su

Memo No.
Date / /

- C passes by value instead of reference. Using malloc to assign memory, and then pass the pointer to another function, is more efficient than having the function recreate the structure.

Memo No.
Date / /

Mo Tu We Th Fr Sa Su

CO7:

strdup, strndup, strdupa, strndupa
- duplicate a string.

The strdup() function returns a pointer to a new string which is a duplicate of the string is obtained with malloc(3), and can be freed with free()

The strndup() function is similar, but copies at most n bytes.

Mo Tu We Th Fr Sa Su

Memo No.
Date / /

```
char *strdup (const char *src) {
    char *dst = malloc (strlen (src) + 1);
    if (dst == NULL) return NULL; // No m
    strcpy (dst, src); // copy the char
    return dst; // Return the new string.
}
```

1. It tries to allocate enough memory to hold string (plus a '\0' character to mark the end of the string)

2. if the allocation faild, its sets `errno` to `ENOMEM` and returns `NULL` immediately. Setting of `errno` to `ENOMEM` is something `malloc` does in POSIX so we don't need to explicitly do it in our `strdup`. If you're not `POSIX` compilant, ISO C doesn't actually mandate the existence of `ENOMEM` so I haven't included that here.

Mo Tu We Th Fr Sa Su

Memo No.
Date / /

// sacpe for length plus nul
emory
acters.

2. Otherwise the allocation worked so we copy the hold string to the new string and return the new address (which the caller is responsible for freeing at some point).

Posix (Portable Operating System Interface) is a set of standard Operating System interfaces based on the Unix Operating system.

Mo Tu We Th Fr Sa Su

Memo No.
Date / /

Les fichiers headers, les includes, les defines et les macros.

- Il existe des fichiers à l'extension .h que l'on appelle les fichiers headers qui contiennent diverses informations pour utiliser des fonctions en C.
- On a vu par exemple que l'on ne pouvait pas utiliser la fonction write sans ajouter cette ligne en début de fichier

```
#include <unistd.h>
```

- la présence des < > indique que le fichier se trouve dans le dossier des headers standard. En général, c'est /usr/include. On peut donc voir le contenu du fichier pour mieux comprendre.

Mo Tu We Th Fr Sa Su

Memo No.
Date / /

- Si on met des " ", cela signifie que le fichier header est dans le dossier courant (ou celui spécifié à la compilation grace a -I).

- En général, on aura un fichier .h par fichier .c.
- Cela permet de compiler avec plusieurs fichiers séparés.

- Le mieux est d'avoir un fichier .c par fonction.

- Les fichiers headers contiennent des prototypes de fonction.

type_de_variable_retournee nom_fon (type_arg1, typ

Mo Tu We Th Fr Sa Su

Memo No.
Date / /

- Les fichiers headers contiennent aussi des defines.

```
#define NOM Valeur
```

Ce sont des ~~Variable~~ Valeurs qui sont remplacées dans le code avant la Compilation.
On met leurs noms en majuscule de façon à la différencier des noms de variables.
Très utile lorsque que l'on utilise plusieurs fois une Valeur : Si on a besoin de la changer, on ne le fera qu'une seule fois.
C'est toujours bien d'en utiliser, même si la Valeur n'est utilisée qu'une fois, dans le cas où elle peut être modifiée.
Ca marche aussi avec des chaines de caractères !

```
ang2);
```

Mo Tu We Th Fr Sa Su

Memo No.
Date / /

Exemple :

Le fichier test_read.c

```
# include "test_read.h"

int test_read (int fd)
{ char buffer [BUFF_SIZE];

  if (read (fd, buffer, BUFF_SIZE) == -1)
    {
      print_string_fd(STDERR_FILENO, MSG_RE
      return (false);
    }

  Retur (TRUE); //TRUE.
}

int main (void) {
return (test_read (STDIN_FILENO) ? EXIT_S
}
```

Mo Tu We Th Fr Sa Su

Memo No.
Date / /

Le fichier test_read.h

```
#ifndef TEST_READ_H_
#define TEST_READ_H_

#include <stdlib.h>
#include <unistd.h>

#define FALSE 0
#define TRUE !FALSE

#define BUFF_SIZE 1024
#define MSG_READ_ERR "Error : read\n"

int test_read(int);
#endif /* !TEST_READ_H_ */
```

-ERR);

UCCESS, EXIT_FAILURE);

Memo No.
Date / /

- Vous vous en serez douté : STDOUT_FILE
Exit_SUCCESS, etc sont des defines!
Vous pouvez aller voir les déclarations de STDOUT_FILENO aussi ce dans /user/include/unistd.h.
- Le fichier .c inclut son fichier .h associé, et rien d'autre.
Les defines, includes, macros, prototypes se trouvent dans le fichier .h
- On peut faire des if et des else dans les .h !
  - #if EXPRESSION = si EXPRESSION est vrai (les expressions sont à peu près équivalente au c)
  - #else if EXPRESSION = equivalent au else if en c.
  - #else = equivalent au else en c.
  - #ifdef NOM_DEFINE = si NOM_DEFINE a été déclaré.

Mo Tu We Th Fr Sa Su

Memo No. C08
Date / /

10. #ifndef NOM_DEFINE = si NOM_DEFINE n'a pas été déclaré.

#endif = à la fin des vérifications.

Dans l'exemple ci-dessus, on a fait ce qui s'appelle une protection contre la multi-inclusion. En fait, on définit toutes nos déclarations dans une définition appelée _TEST_READ_H_.

Mais avant de le faire, on vérifie si cette grosse définition n'a pas été faite auparavant, si, par ~~exemple~~ exemple, j'avais inclus 2 fois le fichier test_read.h.

Mo Tu We Th Fr Sa Su

Memo No.
Date / /

COS: Ex 03:

The C language contains the typedef keyword to allow users to provide alternative names for the primitive (eg int) and user defined (e.g struct) data type.

This keyword adds a new name for some existing data type but does not create a new type.

typedef <existing_type> <alias>

Using typedef struct results in a cleaner, more readable code, and saves the programmer keystrokes. However, it also leads to a more cluttered global namespace which can be problematic for ~~toy~~ large programs.

Mo Tu We Th Fr Sa Su

Memo No.
Date / /

Eg 9:

Make file is a program building tool which runs on Unix, linux, and their flavors. It aids in simplifying building program executables that may need various modules. To determine how the modules need to be compiled or recompiled together, make takes the help of user-defined makefiles.

Makefile guides the make utility while compiling and linking program modules. Anyone who wants to compile their programs using the make utility ~~wants to gain knowledge on makefile.~~

Compiling the source code files can be tiring, especially when you have to include several source files and

Mo Tu We Th Fr Sa Su

Memo No.
Date / /

type the compiling command every time you need to compile. Makefiles are the solution to simplify this ask.

makefiles are special format files that help build and manage the projects automatically.

for example:

main.c
hello.c
factorial.c
functions.h

gcc main.c hello.c factorial.c -o heller

this command generates hello binary. In this example we have only four files and we know the sequence of the function calls. Hence, it is feasible to type the above

Mo Tu We Th Fr Sa Su

Memo No.
Date / /

Command and prepare a final binary.

However, for a large project where we have thousands of source code files. It becomes difficult to maintain the binary builds

the make command allows you to manage large programs or groups of programs. As you begin to write large program ~~(such as a single)~~, you notice that re-compiling large programs takes longer time than re-compiling short programs. Moreover, you notice that usually only work on a small of the program (such as a single function) and much of the remaining program is unchanged.

Mo Tu We Th Fr Sa Su

Memo No.
Date / /

what alternatives are there to make?

popular C/C++ alternative build systems are Scons, CMake, Bazel, and Ninja. Some code editors like Microsoft Visual Studio have their own built in build tools. for java, there's Ant, Maven and gradle other languages like Go and Rust have their own build tools.

The versions and types of Make:
There are a variety of implementations of a Make, but most of this guide will work on whatever version you're using. However, it's specifically written for GNU Make, which is the standard implementation on Linux and Mac Os.

Memo No.
Date / /

Makefile Syntax:

Targets : prerequisites
    command
    command
    command.

- The targets are file names, separated by spaces. Typically, there is only one per rule.
- The commands are a series of steps typically used to make the target(s) these need to start with a tab character, not spaces.
- The prerequisites are also file names, separated by spaces. theses files need to exist before the commands for the target are run. these are also called dependencies.

Mo Tu We Th Fr Sa Su

Memo No.
Date / /

Cos : ex 0 :

Creating libraries :

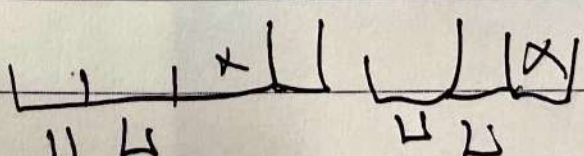
- If you have a bunch of files that contain just functions, you can turn these source files into libraries that can be used statically or dynamically by programs. This is good for program modularity. and code re-use. Write One, Use many.

Memo No.
Date / /

Char * strchr Vs char strchr

1. Arguments in C are passed by Value which means that the argument Variable (s in your func) is a copy of the Value from your variable in the main function. modifing this copy (like assigning to it) will only change the local Variables Value and not the original Variables Value.

To solve the problem easily by making the function return a pointer instead.



Mo Tu We Th Fr Sa Su

Memo No.
Date / /

1: Pre-processing:

is the first step in the compilation process.

C performed using the pre-processor tool. (A pre-written program invoked by the system during the compilation)

All the statements starting with the # symbol in a C program are processed by the pre-processor.

and it converts our program file into an intermediate file with no # statements

Under following pre-processing tasks are performed:

Mo Tu We Th Fr Sa Su

Memo No.
Date / /

1- Comments Removal :

2- Macros

are some constant values or expressions.

3- file inclusion.

is addition of another file containing some pre-written code.

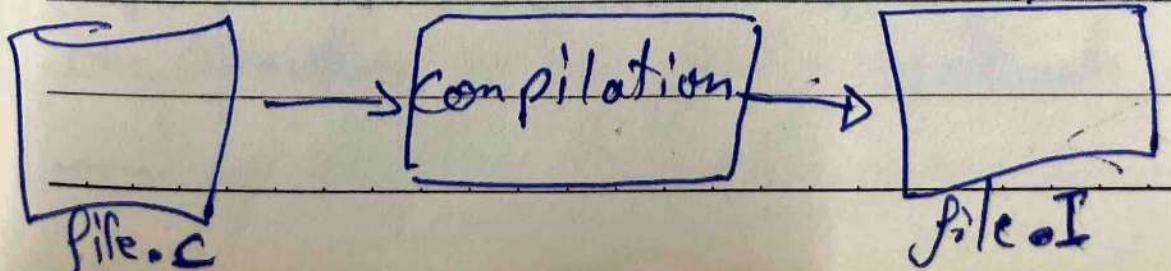
its is done using the #include.

4- Conditional Compilation :

conditional compilation is burning or avoiding a block of code after checking if a macro is defined or not.

#if def

#end gf.



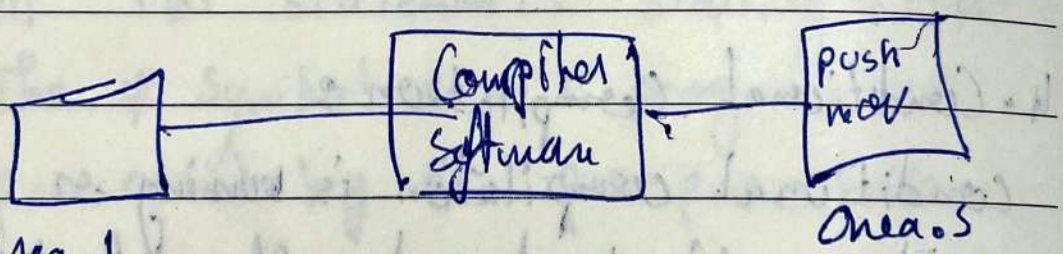
Mo Tu We Th Fr Sa Su

Memo No.
Date / /

(2) Compiling :

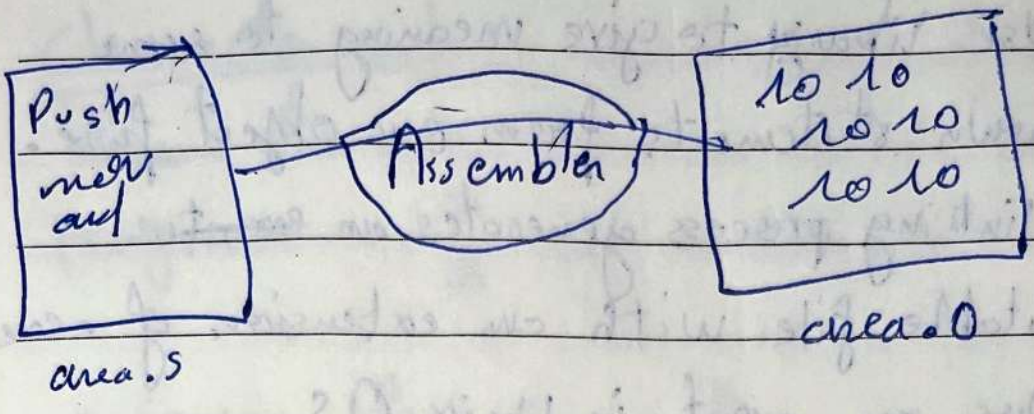
Compiling phase in C uses an inbuilt compiler software to convert the intermediate (.i) file into an Assembly file (.s)

Assembly code : is a simple English-type language used to write low-level instruction (in micro-controller programs, we used assembly language)



(3) Assembling :

Assembly level code (.s file) is converted into a machine-understandable code



(ii) Linking

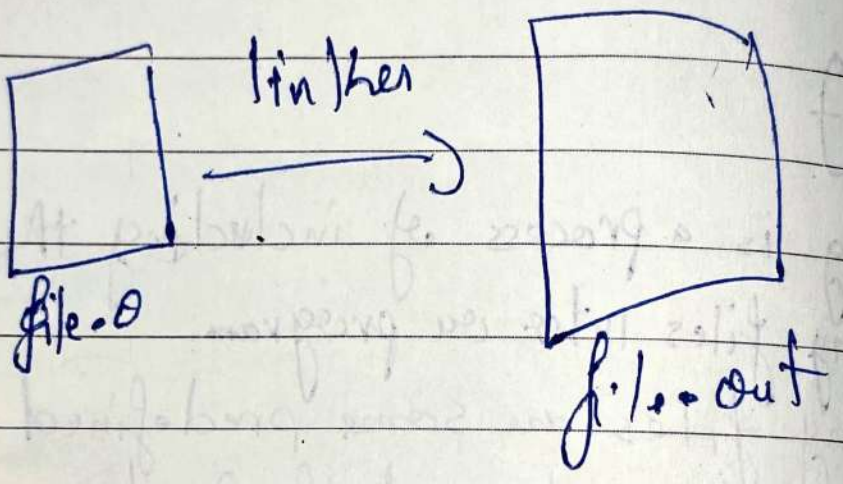
Linking is a process of including the library files into our program.
library files are some predefined files that contain of the functions in the machine language and these files have an extension of .lib.

Some unknown statements are written in the object in the object (.o/.obj) file that our operating system can't understand.

Mo Tu We Th Fr Sa Su

Memo No.
Date / /

We use library to give meaning to some unknown statements from our object file. The linking process generates an executable file with an extension of .exe in Dos an .out in Unix OS.



Difference between compiler and interpreter

Mo Tu We Th Fr Sa Su

Memo No. _____
Date / /

Character set: includes set of valid ~~char~~ characters we can use in our programs in different environments:

- Source character set

Mo Tu We Th Fr Sa Su

Memo No.
Date / /

Memory layout in C :

When we execute a C program, the executable code of the file loads into RAM in an organized manner

computers do not access program instructions directly from secondary storage because the access time of secondary storage is longer when compared to that of RAM.

RAM is faster than secondary but has a limited storage capacity, so it is necessary for

Mo Tu We Th Fr Sa Su

Memo No.
Date / /

A C program memory layout in C mainly comprises six components these are heap, stack, code segment, command-line arguments, uninitialized and initialized data segments. Each of these segments has its own read, write permissions.

A segmentation fault occurs when a program tries to access any of the segments in a way that is not allowed, which is also a common reason for the program to crash.

① Text segment:

- after we compiled the program, a binary file generate, which is used to execute our program by loading

Memo No.
Date / /

it into RAM. This binary file contains instructions and, these instructions get stored in the text segment of the memory.

- Text segment has read-only permission that prevents the programs from accidental modificatio
- text segment in RAM is shareable so that a single copy is required in the memory for frequent applications like text-editor, shells etc.

② Initialized data segments: or data segment is a part of the computers virtual memory space of a program that contains

Memo No.
Date / /

values of all external, global, static and constant variables whose values are initialized at the time of variable declaration in the program. ~~Dec~~ Because the values of variables can change during program execution.

this memory segment has read-write permission. we can further classify the data segment into the read-write, and read-only areas.
const variable comes under the read-only area. the remaining types of variables come in the read-write area.

~~Keyword~~ const

Mo Tu We Th Fr Sa Su

Memo No.
Date

(d) Uninitialized data segment:
An uniditialized data segment is also known as bss (block started by symbol). The program loaded allocates memory of this segment when it loads. Every data in bss is initialized to 0 and pointers to null pointer.

Stack:
The stack segment follows the LIFO (last in first out) structure and grows down to the lower address but it depends on computer architecture.

Stack grows in the direction opposite to heap.

Stack segment stores the value of local variables and values of parameters

Mo Tu We Th Fr Sa Su

Memo No.
Date / /

(4) Uninitialized data segment.

An uninitialized data segment is also known as bss (block started by symbol). The program loaded allocates memory of this segment when it loads. Every data in bss is initialized to 0 and pointers to null pointer.

Stack:

The stack segment follows the LIFO (last in first out) structure and grows down to the lower address but it depends on computer architecture.

Stack grows in the direction opposite to heap.

Stack segment stores the value of local variables and values of parameters

Memo No.
Date / /

passed to a function along with some additional information like the instruction's return address, which is to be executed after a function call.

Stack pointer register keeps track of the top of the stack and its value changes when push/pop actions are performed on the segment.

The values are passed to stack when a function is called stack frame. Stack frame stores the value of function temporary variables and some automatic variables that store extra information like the return address and details of the caller's environment (memory regist

Memo No.
Date / /

Mo Tu We Th Fr Sa Su

Each time function calls itself recursively, a new stack frame is created, which allows a set of variables of one stack frame to not ~~interface~~ interfere with other variables of a different instance of the function. This is how recursive functions work.

Why

~~1) any computer system~~
~~It needs a storage~~

1) any computer system to be useful, it needs a storage - storage for data.

- a computer need : primary storage for any data. that it needs access quickly.

- this includes the start-up instructions. the operating system, Program

* There are two main types of primary storage : RAM-ROM

RAM:

Ram is a part of the main memory in a computer system.

When a program is loaded, it is copied from secondary storage, such as a hard-disk
* Any data is associated with the program will also be stored in RAM so that the CPU can access both the data and instructions.

that are running and any associated data.

The units of a data storage.

Computer uses electronic circuits etched onto computer chips to store data and instructions. These circuits electronic switches made from tiny transistors. Each switch can be in one of two states: On or off.

Memo No.
Date / /

Mo Tu We Th Fr Sa Su

The two states are represented by the numbers 1 or 0. A computer uses combinations of these 1s and 0s to represent data and instructions.

There is a number system that only uses the two values 1 and 0. it is called the binary number system and it is used to describe the On/off status of all the switches in a computer.

One binary digit is called a bit, Computers often group 8 bits together as one unit of data.

These 8 bits togather are called a byte.

Memo No. _____
Date / /

Mo Tu We Th Fr Sa Su

Data Storage:

Numbers:

In the denary (or decimal) system we are used to using 10 symbols or values: 0, 1 ... 9, we use these symbols to write numbers.

100s	10s	1s
5	2	7

← heading value

In binary we just have two symbols or values, 0 and 1. this means that in binary number system each column heading is twice as big as the previous one. as we move from right to left

128	64	32	16	8	4	2	1
		1	0	0	1	1	1

How convert 32 + 4 + 2 + 1

= 39

~~how to convert 39 to~~

100111 in binary is 39.

Convert the denary number 142 into binary:

Is 128 smaller than 142? yes it we record 1 in the first column from left-hand side.

	128	64	32	16	8	4	2	1
	1							

142 - 128 = 14

We subtract 128 from 142, to leave a remainder of 14.

We now check 14 against the next column value,

	128	64	32	16	8	4	2	1
	1	0	0	0	1			

8 is smaller than 14.

14 - 8 = 6

	1	0	0	0	1	1	0

Memo No. _____

Date / /

Mo Tu We Th Fr Sa Su

characters:

Text is stored on a computer by first converting each character to an integer and then storing the integer.

Example:

to store the letters 'A', we will actually store the number 65; 'B' is 66. 'C' is 67...

A letter is usually stored using a single byte (8 bits) Each letter is assigned an intger number and that number is stored. A = 65

The conversion of letters to numbers is called an encoding. The encoding used in the examples above is called ASCII

Memo No.
Date / /

Mo Tu We Th Fr Sa Su

4a The programming language variables are used for?:

When you want to create a program in your computer to do some instructions for example the sum of 2 numbers

~~Firstly you must to store this~~ ~~in your data store (RAM)~~

for every computer application we must store the information at a particular location.

Variables are used in a computer program to store a single piece of data. The content of a variable may be changed during the running of a program. when a variable is first defined, the programming language allocate a

small area of memory to store this data.

A variable has an identifier, or name this is the label given to the area of memory. ~~Variable~~ Variable identifiers can be almost anything, but they must not contain a

Constants are used in computer programs to store a single piece of data that has a fixed value and cannot change during ~~the~~ of the program. ~~A~~

Assignment means to give a value to a variable a constant. this normally done using the = Sign

~~data~~ data type name = value ;

Mo Tu We Th Fr Sa Su

Memo No. _____

Date / /

Data types:

the type of data to be stored determines how much memory needs to be allocated to that variable or constant