C Programming Language

By: Mohamed Aziz Tousli

About C

- C is a high level language
- Code source = a written program
- IDE = Integrated Development Environment = Compiler + Text editor + Debugger
- Window program (paint) vs Console program (cmd)
- Every code source in C must end with an empty line
- We can’t print accents in C + Windows

Basics

- //This is a short comment
- /* This is a long comment */
- Common libraries: stdio.h, stdlib.h
- Main function: obligatory cause the program starts with it

```
int main(int argc, char* argv[]){
Instructions ;
return 0} //return 0 for orgazional purposes
```

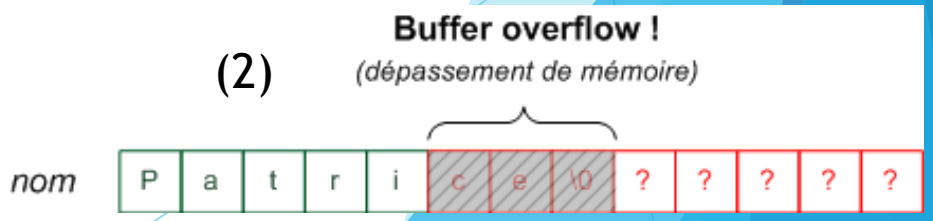
- Operations: + - * / %
- Quotient = 5/2 = 2; Rest = 5%2 = 1; Division = 5.0/2.0 = 2.5
- x=x+1 ⇔ x++ ⇔ x+=1 //Incrementation
- x=x-1⇔ x-- ⇔ x-=1 //Decrementation

Memory

- Memories: (Faster → Slower) (Smallest → Biggest)
  - Registers: Ultra-rapid situated directly on the processor (Volatile)
  - Cache: Link between registers and RAM (Volatile)
  - RAM: The most used memory in practice (Volatile)
  - Hard disk: Where files are saved (Non volatile)
    - Volatile: Maintains data while the device is powered
- Contents of a memory:
  - Address: Index number (0→…)
  - Value: Content of address (number)
    - We have **one** value per address

Variables

- const constType CONST_NAME=Value; //Can't be changed
- variableType variableName1, variableName2=Value; //Ab_1✓; 1✗; é✗
- Initially, a variable takes the value of the content of the address: random
- ➔ We have to do initialization for variables to avoid randomness problems
- variableType = signed char, int, long; float, double //float x=3.5
- variableType = unsigned char, unsigned int, unsigned long
- printf("The variable %d is not %f",variable1,variable2); //To write
- "\n" "\t" special characters for formatting the message
- scanf("%d %f",&variable1,&variable2); //To read
- Print float/long float with "%f", scan float with "%f" and long float with "%lf"
- "%p" for hexadecimal; "%c" for character; "%s" for string
- scanf problems: (1) it stops at a space or a special character
- c=getchar(); ⇔ scanf("%c", &c); //Read first character
- 'C'=toupper('c'); //Upcase a character, we need the library ctype.h



Math library & Random library

- #include<math.h>
- fabs(-5.0)=5.0 //|x|
- abs(x) exists in stdio.h for integers only
- ceil(25.5)=26.0 //Ceiling
- floot(25.5)=25.0 //Floor
- pow(5,2)=5^2=25 //Power
- sqrt(25)=5 //Square root
- sin(x), x is in radian
- #include<time.h>
- srand(time(NULL)); //Must be called only one time
- x=rand() % (MAX-MIN+1) + MIN; //Generate a random number in [MIN,MAX]

Conditioning

```
if (/* condition1 */)
{ /* code */ }
else if (/* condition2 */)
{ /* code */ }
else
{ /* code */ }
```

```
switch(x)
{
    case 1:
        /* code */
        break;
    default:
        /* code */
}
```

- Comparative forms: ==, <, <=, >, >=, != ; && **AND**, || **OR**, ! **NOT**
- If code has 1 instruction, {} can be removed
- if (x) <=> if(x==1) : Boolean variables - 0=False, !0=True
- Boolean variables do not exist in C, so we use int instead
- x = (/* condition */) ? ifChoice : elseChoise; //Conditional teranary

Loops

```
while(/* condition */)
{
	/* code */
}
```

```
for (int i = 0; i < count;
i++)
{
	/* code */
}
```

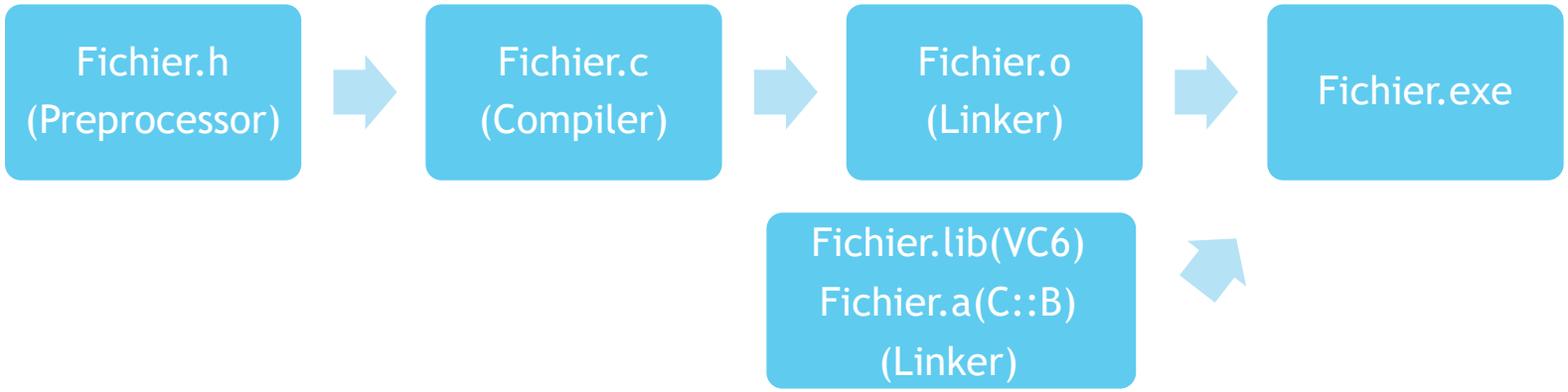
```
do
{
	/* code */
} while(/* condition */);
```

Functions

- typeResult FunctionName(typeA A, typeB B) //parameters or arguments
- {
- /* code */
- return x;
- }
- Type is void if function has no return
- x=FunctionName(A,B) //Call a function
- Modular programming: Program using librairies .h
- type FunctionName(typeA, typeB); //Prototype
- If prototype before main, function can be anywhere; else function before main
- Project = a set of .c & .h files ; .c contains functions & .h contains prototypes

Compilation & Global/Local variables

- Seperate compilation:



- Created .h call .c & standard .h call .a/.lib
- Global variable: x after #include → Visible for all the files in the project
- Local variable: x in a function → Visible for the function only
- Global variable: static x after #include → Visible for the file only
- Local variable: static x in a function → x keeps its value when the function exits
- Global function: by default → Visible in all the files in the project
- Local function: static function → Visible in the file only

Pointers

- For variables:
  - x //Shows its value
  - &x //Shows its address
- For pointers:
  - x //Shows its value
  - *x //Shows the value of the variable that x points on
- T* pointer=&x; T *pointer1=NULL,*pointer2; //Create a pointer on type T
- void FunctionName(Type *Pointer) {Everything *Pointer} //In prototype
- FunctionName(&x); //In main {1}
- Type *Pointer=&x;FunctionName(Pointer);Use(*Pointer or x); //In main {2}

Arrays

- Array = Sequence of variables of the same type, located in contiguous space in memory
- int array[N]; //Create an array of size N; array itself is a pointer on array[0]
- N must not be a variable/constant, N must be a number
- array[i]; //ith+1 value of array because arrays start with index 0
- int Array[N]={Value1, Value2}; //array=[Value1, Value2, 0, .., 0] //It completes with 0 by default
- void FunctionName(Type *Array, int ArraySize) //Use functions to call arrays {1}
- void FunctionName(Type Array[], int ArraySize) //Use functions to call arrays {2}
- PS: matrix[x][y];
- PS: C doesn't know matrices, so if we want to give a matrix as an argument, we have give the second dimension

Strings (1)

- char c='A'; //Create ONE character
- 'A'=65 - ASCII Table
- char str[N]; //Create a string of size N, we can str[i]='X'
- char* str; //Create a string, we can't str[i]='X'
- " " for strings; ' ' for characters
- **A string ends with the character '\0'**
- For arrays of type char (i.e. string), we don't have to pass ArraySize as an argument
- char str[]="StringName"; //string={'S','t', .., '\0'}; Size is automatically calculated
- Last line can't be done in code. It is done only in initialization

Strings (2)

- #include<string.h>
- sizeStr = strlen(str); //Size of a string
- strcopy(str2,str1); //str2=str1
- strcat(str2,str1); //str2=str2+str1: Concatenation
- We have to put N too big to assure that it has no limits problem
- strcmp(str2,str1); //Compare str2 to str1; 0 if equal, !0 if not
- strRest = strchr(str, character); //Look for ch in str, return the rest of str after ch; NULL if not
- strRest = strpbrk(str, characters); //Look for one of the chs and return the rest of str after it
- strRest = strstr(str1, str2); //Look for str2 in str1 and return the rest of str1 after str
- sprintf(str,"message %d",x); //str="message xValue"
- A string must be initialized with “” to avoid memory problems

Preprocessor directives (1)

- Preprocessor: Replace #'s with other values before compiling
- #include<library.h> //Standard library (Replace contents of .h in .c file)
- #include”library.h” //Created library
- #define N 5 //Replace N with 5 in the whole file
- const takes place in memory, define doesn't (because it is done in the preprocessing)
- Predefined constants by the preprocessor:
  - __LINE__ //number of current line
  - __FILE__ //name of current file
  - __DATE__ //date of compilation
  - __TIME__ //hour of compilation
- Macro without parameters:
  - #define HELLO() printf("Message1"); \
  - printf("Message2"); //Replace HELLO() with printf("Message")
- Macro with parameters:
  - #define Function(x,y) if(x||y) {};
  - Function(5,6) //Replace Function with if and (x,y) with (5,6)

Preprocessor directives (2)

- Conditional compilations:
- #if condition1
- /* code source to compile if condition1 is true */
- #elif condition2
- /* code source to compile if condition2 is true */
- #endif
- Utility of #define constant without value:
- #define WINDOWS
- #ifdef WINDOWS
- /* code source for WINDOWS */
- #endif
- To avoid infinite inclusions:
- #ifndef DEF_hFILE //If DEF_hFILE was not defined, i.e., DEF_hFILE was never included
- #define DEF_hFILE //We define DEF_hFILE so that it won't be called next time
- /* contents of DEF_FILE.h */
- #endif

Types

- typedef struct structureName structureName;
- struct structureName {
- Type1 variable1;
- Type2 variable2;
- }; //Do not forget about ';'
- → typedef create an equivalent of the structure. 'struct structName' is the name of the structure that we want to copy. 'structureName" is the name of the equivalent ➔ Writing 'structureName' is the same as 'struct structureName' since it is annoying to create a structure without a typedef
- structureName variableName={,}; //Create a variable with type structureName
- variableName.variable1 //Value1 of VariableName
- (*variableName).Variable1 ⇔ variableName->Variable1 //Pointers on structures
- '.' is for variables and '->' is for pointers
- We put new types usually in .h files
- typedef enum typeName typeName;
- enum typeName {VALUE1=x, VALUE2=y, VALUE3=z};
- If no (x,y,z), compiler does an automatic association to values 0, 1 and 2

Files

- FILE* f=fopen("C:\\FileDirectory.type",OpeningMode); //Open a file; "r,w,a,r+,w+,a+" (read, write, append)
- if f=NULL, it is impossible to open the file
- "w+" is dangerous because it deletes the content of the file
- "w" if the file exits, it is replaced, else it is created
- a=fclose(f); //Close the file, a=0 if done right, else a='EOF' (End Of File)
- fputc(character,f); //Write a character in the file
- fputs(string,f); //Write a string in the file
- fprintf(f,string); //Write a string in the file (It can have variables within it, like printf)
- → These functions return the integer 'EOF' in case of error
- fgetc(f); //Read a character, return 'EOF' if it doesn't exist
- fgets(str,NumberOfChars,f); //Read a line, return 'NULL' if it doesn't exist
- There is a 'cursor' that goes through the character/lines every time the function is called
- fscanf(f,"%d",&Variable); //Read like scanf()
- position=ftell(f); //Give the position of the 'cursor'
- fseek(f,displacement,origin); //displacement can be positive or negative, origin=SEEK_SET,SEEK_CUR,SEEK_END
- rewind(f); <=> fseek(f,0,SEEK_SET);
- rename(oldName,newName); //Rename a file, return 0 if done right
- remove(f); //Delete a file from the hard disk!

Dynamic allocation

- sizeof(Type) //Functionality in C to know the size of a type (not a function)
- → char occupies 1 byte, int occupies 4 bytes..
- So far, the program has been demanding the OS to free space for the variables: Automatic allocation
- But now, we are going to demand the OS manually to free space for the variables: Dynamic allocation
- z=malloc(numberOfNecessaryBytes); //Memory Allocation, returns a pointer on void (on any type) i.e. @
- z=malloc(sizeof(Type)); //'Indicate' for the pointer the type of its variable
- if z==NULL, exit(0); else do the work
- Be careful: In dynamic allocation, we use pointers instead of standard variables
- free(z); //Free the memory allocation
- Dynamic allocation is good for arrays, since we don't know its size and we can't put a variable for it
- arrayType* array=(arrayType*)malloc(arraySize*sizeof(arrayType)); //Create a table dynamically

Secured read

- str=gets(); //Like fgets but doesn't avoid buffer overflow
- str=fgets(str,size(-'\0'),stdin); //Avoid buffer overflow compared to scanf, stdin else FILE*f
- stdin: what is written by the keyboard, pointer on buffer
- Buffer = Memory zone that receives the stdin
- fgets stop when it finds '\n' (and keep it) or when it reaches the size limit, the rest will stay in the buffer and will be extracted from another fgets
- getchar is equivalent to fgets but for characters
- PS: 2 '\0's count as 1, since the program stop at the first one
- long=strtol(str,NULL,base=10); //Convert str to long, 0 if wrong (" 43.5abc" → 43)
- Double=strtod(str,NULL); //Convert str to double, 0 if wrong (" 43.5abc" → 43.5)

SDL Typical Code

```
while (continuer)
{
    SDL_WaitEvent(&event);
    switch(event.type)
    {
        case SDL_TRUC: /* Gestion des événements de type TRUC */
        case SDL_BIDULE: /* Gestion des événements de type BIDULE */
    }

    /* On efface l'écran (ici fond blanc) : */
    SDL_FillRect(ecran, NULL, SDL_MapRGB(ecran->format, 255, 255, 255));

    /* On fait tous les SDL_BlitSurface nécessaires pour coller les surfaces à l'écran */

    /* On met à jour l'affichage : */
    SDL_Flip(ecran);
}
```

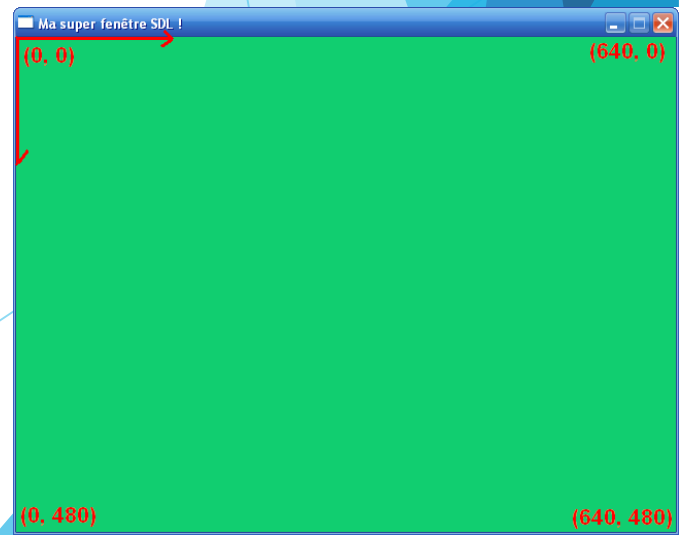
Linker Settings: -lmingw32 -lSDLmain -lSDL -lSDL_image -lSDL_ttf

SDL library (1)

- → Create 2D games
- Standard library (by default) vs Third party library (has to be installed)
- GPL License (General Public License) vs LGPL License (Lesser GPL); both are open source
- #include<SDL/SDL.h>
- x=SDL_Init(SDL_INIT_VIDEO | SDL_INIT_AUDIO | SDL_INIT_CDROM | SDL_INIT_JOYSTICK | SDL_INIT_EVERYTHING); ⇔ Malloc //Load SDL; x=0 if good, -1 if error; SDL_INIT_X: flag
- SDL_Quit(); ⇔ Free //Stop SDL, (and free screen from memory)
- fprintf(stderr,"%s",SDL_GetError()); //To write the error in stderr.txt, and get the latest SDL error
- exit(EXIT_FAILURE); /*main*/ return EXIT_SUCCESS; //Variables of exit that go with any OS
- SDL_Surface* surface = NULL; //Create a pointer on screen (basically, a surface)
- SDL_Surface* screen=SDL_SetVideoMode(width,height,number of colors{32 bits/px},SDL_HWSURFACE{video memory, faster & less space} | SDL_SWSURFACE{RAM, slow & more space} | SDL_NOFRAME | SDL_FULLSCREEN | SDL_RESIZABLE | SDL_DOUBLEBUF{for fluid motion}); //Open a window, NULL if error
- → Without a pause function, it closes automatically
- SDL_WM_SetCaption(newWindowName,NULL); //Change the name of the window

SDL library (2)

- SDL_FillRect(screen,NULL,color); //Color the screen; Type of 'color' is 'Uint32'=int in SDL
- Uint32 color=SDL_MapRGB(screen->format, Red, Green, Blue); //Return a color
- SDL_Flip(screen); //Update the screen (after modifications)
- SDL_Surface* surface=SDL_CreateRGBSurface(SDL_HWSURFACE | SDL_SWSURFACE,W,H,32b/p,0,0,0,0); //Create a new surface inside the main surface, i.e., the screen
- SDL_FreeSurface(surface); //Free surface from memory
- SDL_Rect position; position.x = 0; position.y = 0; //Create a position
- SDL_BlitSurface(surface, NULL, screen, &position); //Blitter surface on screen
- SDL_LockSurface(screen); //Block surface to modify it manually
- SDL_UnlockSurface(screen); //Unblock surface
- PS: Sprites: Images that compose games
- PS: **SDL can't be loaded if main doesn't have a typical main form**

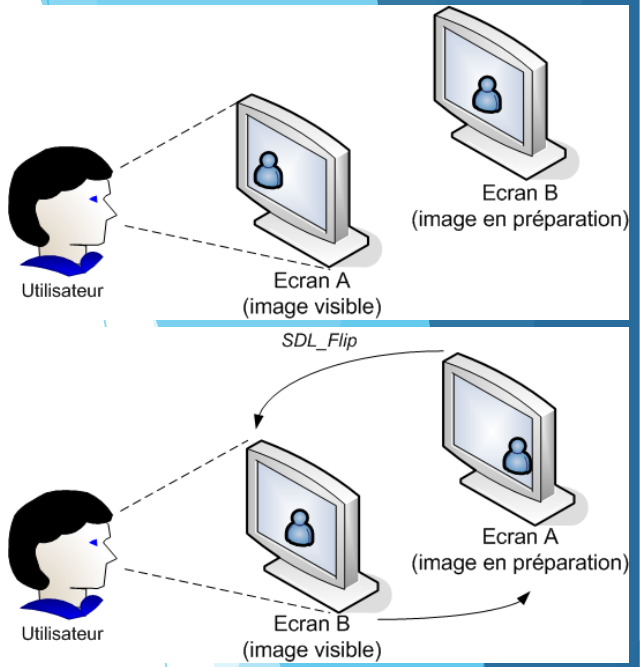


Images on SDL

- BMP – Bitmap (Not compressed image type, better quality but bigger size)
- mySurface = SDL_LoadBMP("imagePath.bmp"); //Load BMP image; SDL works only with BMP type
  - This function replaces SDL_CreateRGBSurface (size of image) & SDL_FillRect (pixels of image)
- SDL_WM_SetIcon(SDL_LoadBMP("iconPath.bmp"), NULL{Transparency}); //Must be 32x32 on Windows
  - PS: SDL_WM_SetIcon must be called before SDL_SetVideoMode
- SDL_SetColorKey(imageToTransparent, SDL_SRCCOLORKEY{Activate transparency}, SDL_MapRGB(imageToTransparent->format, R, G, B)); //Transform color to transparent
  - PS: SDL_SetColorKey must be called before SDL_BlitSurface
- SDL_SetAlpha(imageToTransparent, SDL_SRCALPHA{Activate transparency}, α); //Transform image to transparen
  - PS: α = 0 → Invisible image ; α = 255 → Opaque image ; α = 128 is an optimized value
- #include <SDL_image.h> //Library to load other image types and generate transparency automatically
- myImageSurface = IMG_Load("imagePath.*"); //Load image

Events on SDL (Keyboard)

https://user.oc-static.com/ftp/mateo21/sdlkeysym.html



- **Event** = Signal sent by peripheral (or OS) to the application
- SDL_Event event; //Variable to deal with events
- SDL_WaitEvent(&event); //Wait event by blocking the program (0% of CPU)
- SDL_PollEvent(&event); //Wait event without blocking the program (100% of CPU)
  - event.type = SDL_QUIT //Event of quitting
  - event.type = SDL_KEYDOWN //When a keyboard button is clicked
  - event.type = SDL_KEYUP //When a keyboard button is released
    - event.key.keysym.sym = SDLK_character //Get the value of the button used
- SDL_EnableKeyRepeat(durationMS,delayMS) //Activate the penetration of buttons; duration=delay
  - Enable it in the beginning, and disable it in the end
- screen = SDL_SetVideoMode(|SDL_DOUBLEBUF); //Activate double buffering → Stop image glittering
- PS: We have to delete the old graph before drawing a new graph

Events on SDL (Mouse)

- event.type = SDL_MOUSEBUTTONDOWN //When mouse is clicked
- event.type = SDL_MOUSEBUTTONUP //When mouse is released
  - event.button.button = SDL_BUTTON_LEFT/RIGHT/MIDDLE/WHEELUP/WHEELDOWN //Get the value of the button used
  - event.button.x/y //Get the position of the mouse when clicked
- event.type = SDL_MOUSEMOTION //When mouse is moved
  - event.motion.x/y //Get the position of the mouse when moved → Follow the mouse
- SDL_ShowCursor(SDL_DISABLE/SDL_ENABLE); //Disable/Enable cursosr inside window
- SDL_WarpMouse(x,y); //Place the mouse in a specific position
  - PS: Event of type SDL_MOUSEMOTION will be generated
- Good holding technique:
  1. MOUSEBUTTONDOWN → currentClick = 1 on met un booléenclicEnCoursà 1.
  2. MOUSEMOTION → if currentClick = 1, we are holding the mouse
  3. MOUSEBUTTONUP → currentClick = 0

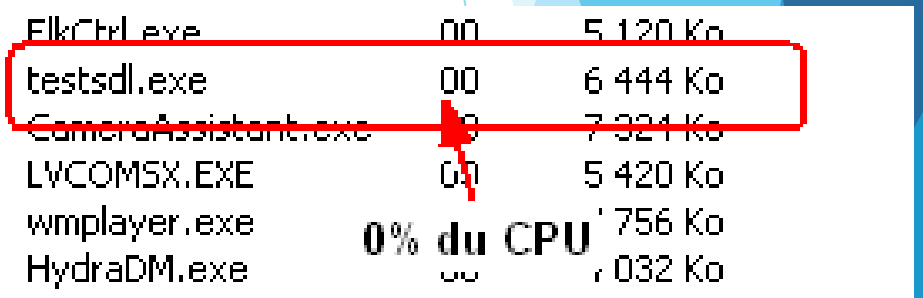
Events on SDL (Window)

- screen = SDL_SetVideoMode(|SDL_RESIZABLE); //Make screen resizable
  - event.type = SDL_VIDEORESIZE //When window is resized
    - event.resize.w/h //Get the new values of height and width
  - event.type = SDL_ACTIVEEVENT //When window's visibility is changed
    - PS: When an application has **focus**, mouse and keyboard only interact with it
    - event.active.gain //0 if lost focus, 1 if gain focus
    - event.active.state = SDL_APPMOUSEFOCUS //Mouse interaction with window
    - event.active.state = SDL_APPINPUTFOCUS //Keyboard interaction with window
    - event.active.state = SDL_SDL_APPACTIVE //Window reduced
  - PS: if ((event.active.state & SDL_APP*) == SDL_APP*) //Code to test if '*' interacted with window

Time on SDL

```
tempsActuel = SDL_GetTicks();
if (tempsActuel - tempsPrecedent > 30) /* Si 30 ms se sont écoulées */
{
    positionZozor.x++; /* On bouge Zozor */
    tempsPrecedent = tempsActuel; /* Le temps "actuel" devient le temps "precedent" pour nos futurs calculs */
}
else /* Si ça fait moins de 30 ms depuis le dernier tour de boucle, on endort le programme le temps qu'il faut */
{
    SDL_Delay(30 - (tempsActuel - tempsPrecedent));
}
```

- SDL_Delay(MS); //Pause the program for a certain number of MS → Program **sleep**
  - Not to be trusted → Not too precise
- SDL_GetTicks(); //Return the number of MS since the program started
- $FPS = \frac{1000}{Frequency}$: Frames Per Second
- Timer : System that calls a function every X MS
  - Timers use « pointers on functions »
- SDL_Init(|SDL_INIT_TIMER); #Initialize timers in the beginning
- SDL_TimerID timer = SDL_AddTimer(X_MS, SDL_NewTimerCallback callbackFunction, void *paramCallbackFunction);
  - //Create a timer; param points on void (any type is possible); param is only 1 parameter
  - Uint32 X_MS callbackFunction(Uint32 X_MS, void *paramCallbackFunction);
  - We need to replace void with the type of paramCallbackFunction inside callbackFunction
- SDL_RemoveTimer(timer); //Stop timer



Text on SDL

Text that changes often

Text that doesn't change often and background

Text that doesn't change often and no background

- SDL_ttf needs FreeType library to read .tff fonts files
- #include <SDL/SDL_ttf.h> //Import SDL_ttf library
- TTF_Init(); TTF_Quit(); //Start and stop SDL_ttf
- TTF_Font *font = TTF_OpenFont("fontName.ttf",fontSize); //Open font file
- TTF_CloseFont(font); //Close font file
- For « Latin1 » character type: //Choose a character type and write on SDL_Surface
  - myTextSurface = TTF_RenderText_**Solid**(font,"text",color);
  - myTextSurface = TTF_RenderText_**Shaded**(font,"text",textColor,backgroundColor);
  - myTextSurface = TTF_RenderText_**Blended**(font,"text",color);
- SDL_Color color = {R, G, B}; //Create a color variable for SDL_ttf
  - SDL uses Uint32 thanks to SDL_MapRGB & SDL_ttf uses SDL_Color
- TTF_SetFontStyle(font, TTF_STYLE_ITALIC/BOLD/UNDERLINE/NORMAL); //Change font style

Sound on FMOD

SDL_audio	FMOD
Low level library	High level library
Supports WAV only (not compressed)	Supports all types
Has bugs	Doesn't have bugs
Doesn't have 3D effects	Have 3D effects
Free + LGPL	Free + No LGPL

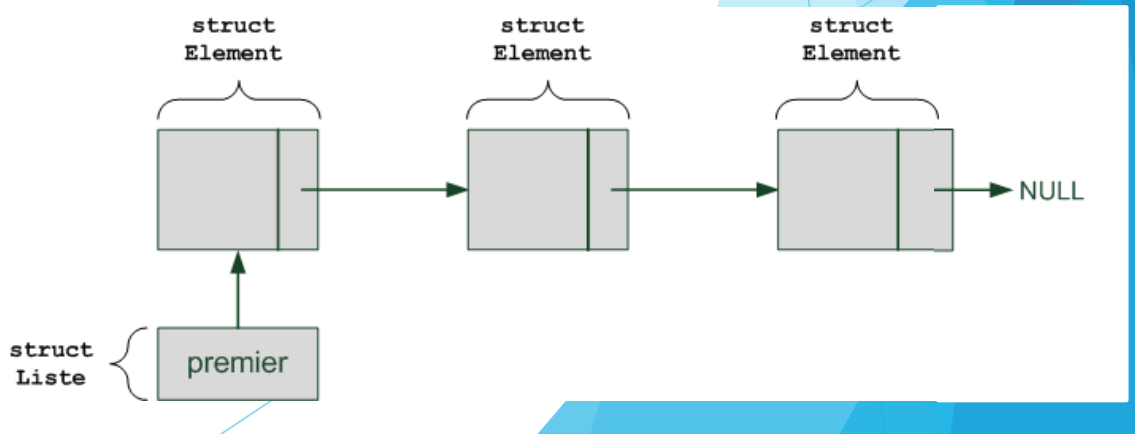
- #include <fmod.h> //Import FMOD library
- FMOD_SYSTEM *system; //Create system object
- FMOD_System_Create(&system); //Give space to system object
- FMOD_System_Init(system, maxChannels{32}, FMOD_INIT_NORMAL, NULL); //Initialize system object
- FMOD_System_Close(system); //Close system object
- FMOD_System_Release(system); //Release system object
- FMOD_SOUND *sound = NULL; //Create pointer on sound object
- FMOD_System_CreateSound(system, "soundDirectory", FMOD_CREATESAMPLE{short sound}, 0, &sound);
  - //Create sound object
- FMOD_System_PlaySound(system, FMOD_CHANNEL_FREE{channel}, sound, 0, NULL); //Play sound
- FMOD_Sound_Release(sound); //Release sound from system

Music on FMOD

- FMOD_System_CreateSound(system, “musicDirectory",FMOD_SOFTWARE|FMOD_2D|FMOD_CREATESTREAM, 0, &music);
  - //Create music object → Music is longer, for memory reasons, it’ll be loaded bit by bit
- FMOD_CHANNEL{GROUP} *channel{Group}; //Create pointer on channel{s} object
- FMOD_System_Get{Master}Channel{Group}(system, channelID, &channel); //Get pointer on channel{s}
- FMOD_RESULT FMOD_Channel_SetVolume(channel,float volume); //Change volume [0.0 → 1.0]
- FMOD_Sound_SetLoopCount(music, n); //Set music in loop [n=1 two loops, n=-1 infinite loop]
  - PS: We need to add FMOD_LOOP_NORMAL in the third parameter of FMOD_System_CreateSound
- FMOD_Channel_GetPaused(channel, &state); //state=1 if paused, state=0 if not paused
- FMOD_Channel_SetPaused(channel, state); //Pause/Unpause a channel
- FMOD_Channel_Stop(channel); //Stop a channel
- FMOD_Sound_Release(music); //Release music from system
- PS: All these functions are applicable on groups by replacing Channel with Channel{Group}
- PS: FMOD_Channel_GetSpectrum(channel,float*array,arraySize{=2^n},0left/1right,FMOD_DSP_FFT_WINDOW_RECT);
  - array[0] //Lowest frequency; array[2^n-1] //Highest frequency; value of array ∈ [0.0 → 0.1]
  - PS: array values change every 25 MS

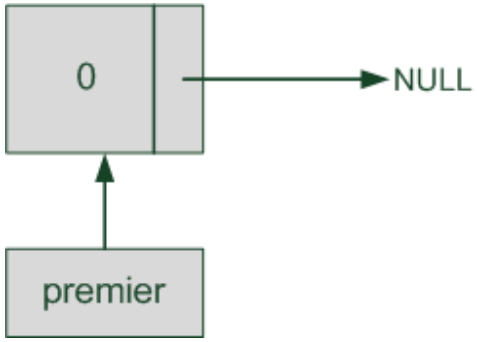
Linked List (1)

- Problem with arrays: Not possible to expand size, not possible to insert in the middle
  - typedef struct Element Element; //Structure for each Element
  - struct Element
  - {
  - variableType variableName;
  - Element *next; //Singly linked list → One way VS Doubly linked list → Two ways
  - };
  - typedef struct List List; //Structure for all elements i.e. for List
  - struct List
  - {
  - Element *first;
  - };

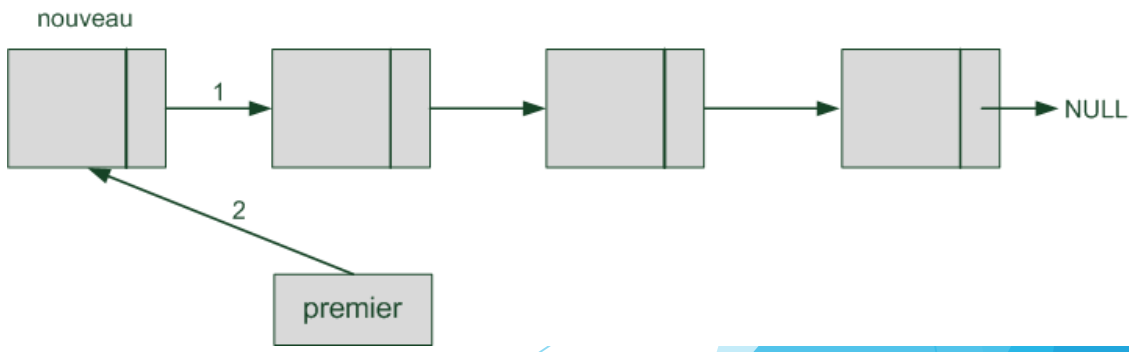


Linked List (2)

```
List *initialization()
{
    List *list = malloc(sizeof(*list));
    Element *element = malloc(sizeof(*element));
    element->variableName = initializationValue;
    element->next = NULL;
    list->first = element;
    return list;
}
```

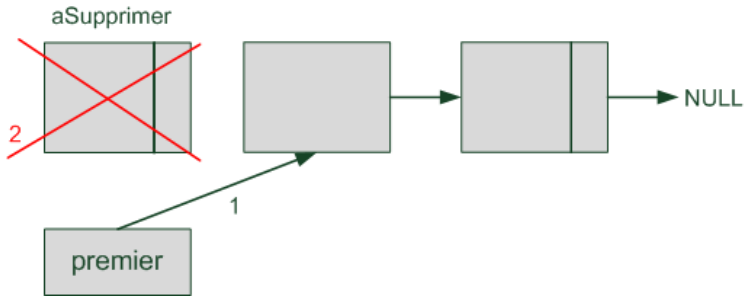



```
void appendFirst(List *list, variableType newElement)
{
    Element *new = malloc(sizeof(*new));
    new->variableName = newElement;
    new->next = list->first;
    list->first = new;
}
```



Linked List (3)

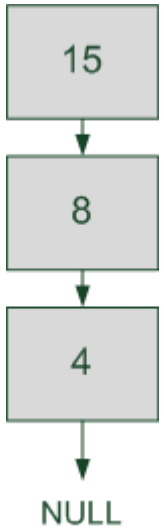
```
void deleteFirst(List *list)
{
    if (list->first != NULL)
    {
        Element *toDelete = list->first;
        list->first = list->first->next;
        free(toDelete);
    }
}
```



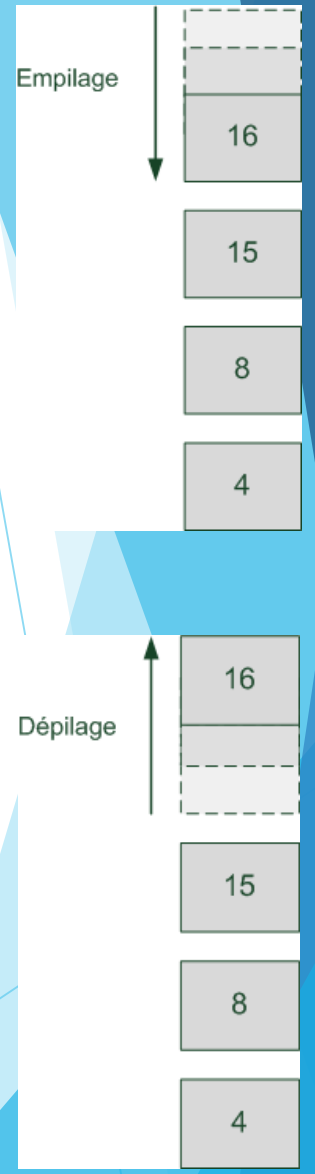

```
void printList(List *list)
{
    Element *current = list->first;
    while (current != NULL)
    {
        printf("%? -> ", current->variableName);
        current = current->next;
    }
    printf("NULL\n");
}
```

Stack

- LIFO algorithm: Last In First Out
  - typedef struct Element Element;
  - struct Element
  - { //Structure for each Element
  - variableType variableName;
  - Element *next;
  - };
  - typedef struct Stack Stack;
  - struct Stack
  - { //Structure for all elements i.e. for Stack
  - Element *first;
  - };

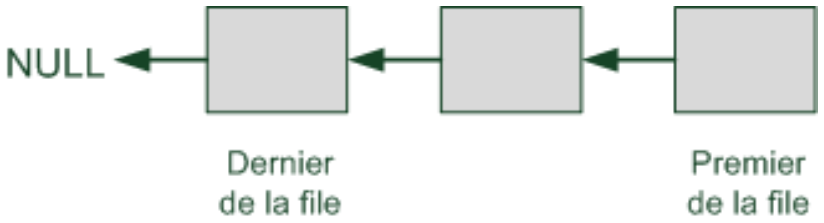



```
void toStack(Stack *stack, variableType newElement)
{
    Element *new = malloc(sizeof(*new));
    new->variableName = newElement;
    new->next = stack->first;
    stack->first = new;
}
variableType toUnstack(Stack *stack)
{
    variableType toDelete;
    Element *stackElement = stack->first;
    if (stack != NULL && stack->first!= NULL)
    {
        toDelete = stackElement->variableName;
        stack->first = stackElement->next;
        free(toDelete);
    }
    return toDelete;
}
```



Queue

- FIFO algorithm: First In First Out
  - typedef struct Element Element;
  - struct Element
  - { //Structure for each Element
  - variableType variableName;
  - Element *next;
  - };
  - typedef struct Queue Queue;
  - struct Queue
  - { //Structure for all elements i.e. for Queue
  - Element *first;
  - };



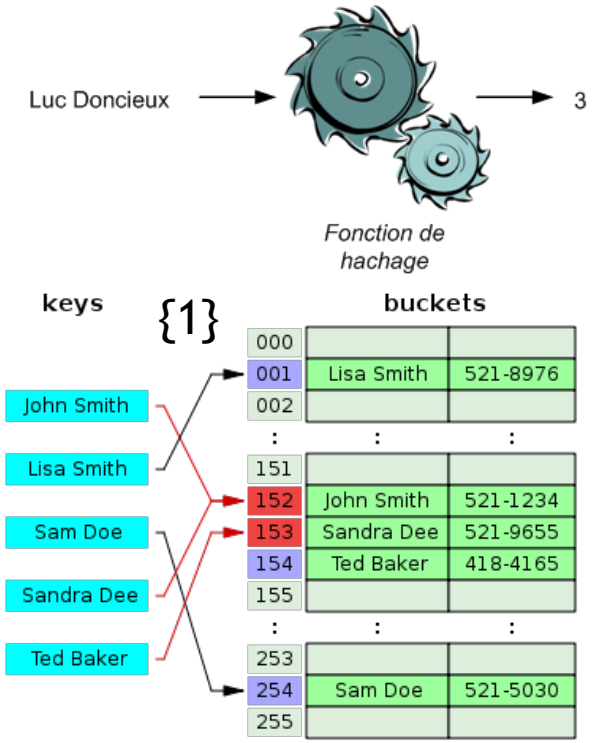

```
void toQueue(Queue *queue, variableType newElement)
{
    Element *new = malloc(sizeof(*new));
    new->variableName = newElement;
    new->next = NULL;
    if (queue->first! = NULL) //Queue not empty
    {
        Element *current = queue->first;
        while (current->next != NULL)
        {
            current = current->next;
        }
        current->next = new;
    }
    else //Queue empty
    {
        queue->first = new;
    }
}
```

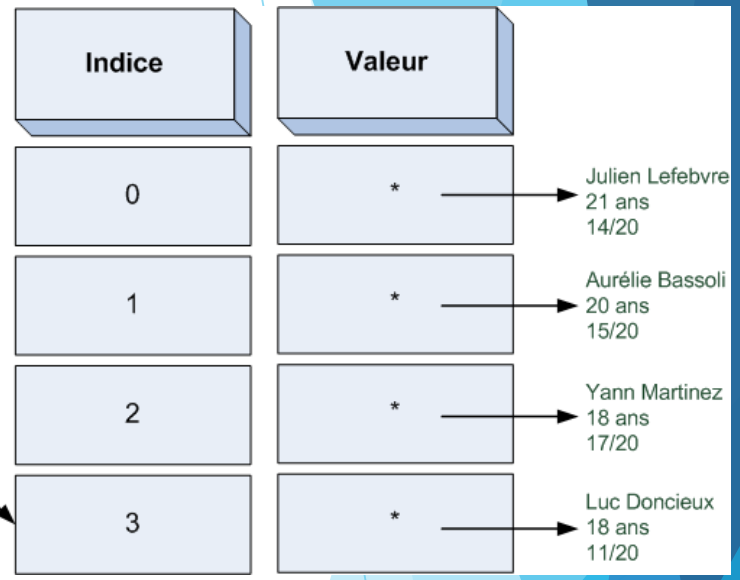
- variableType toUnqueue(Queue *queue); //Same as Stack

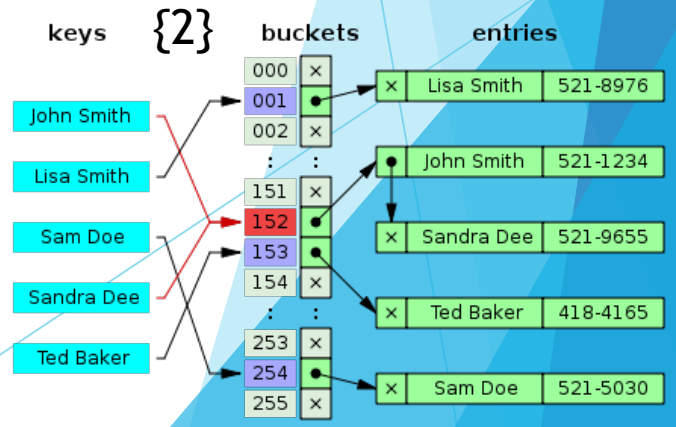
Hash Table

- Problem with linked lists: You have to go through all elements to get last element
- Hash Table = Array + Linked List
- Example: (Famous hash functions: **MD5** et **SHA1**)

```
int hash(char *string)
{
    int i = 0, hashNumber = 0;
    for (i = 0 ; string[i] != '\0' ; i++)
    {
        hashNumber += string[i];
    }
    hashNumber %= 100;
    return hashNumber;
}
```







- Solution to collisions:
  - {1} Open addressing = Linear hash: In case of collision, go to next box, etc…
  - {2} Separate chaining : In case of collision, create a linked list in that box