الجمهورية الشعبية الديمقراطية الجزائرية

People's Democratic Republic of Algeria

وزارة التعليم العالي و البحث العلمي

Ministry of Higher Education and Scientific Research

معهد نوميديا للتكنولوجيا

Numidia Institute of Technology

Field of Study: **Computer Science**

Algorithmic Project

Presented by
GUELLATI Sidali Ilyes

Academic Year: 2024/2025

CONTENTS

Part I

Introduction

This is an algorithmic project in the issue of visualizing BTree and PoW algorithms, where we use a non-regular graphical library (like SDL, GTK, OpenGL), instead an interactive ASCII renderer which is coded FROM SCRATCH. The goal (which is achieved) is to develop an optimized C renderer without the need for a graphical interface (like SDL, X11, VNC) and global compatibility in running it.

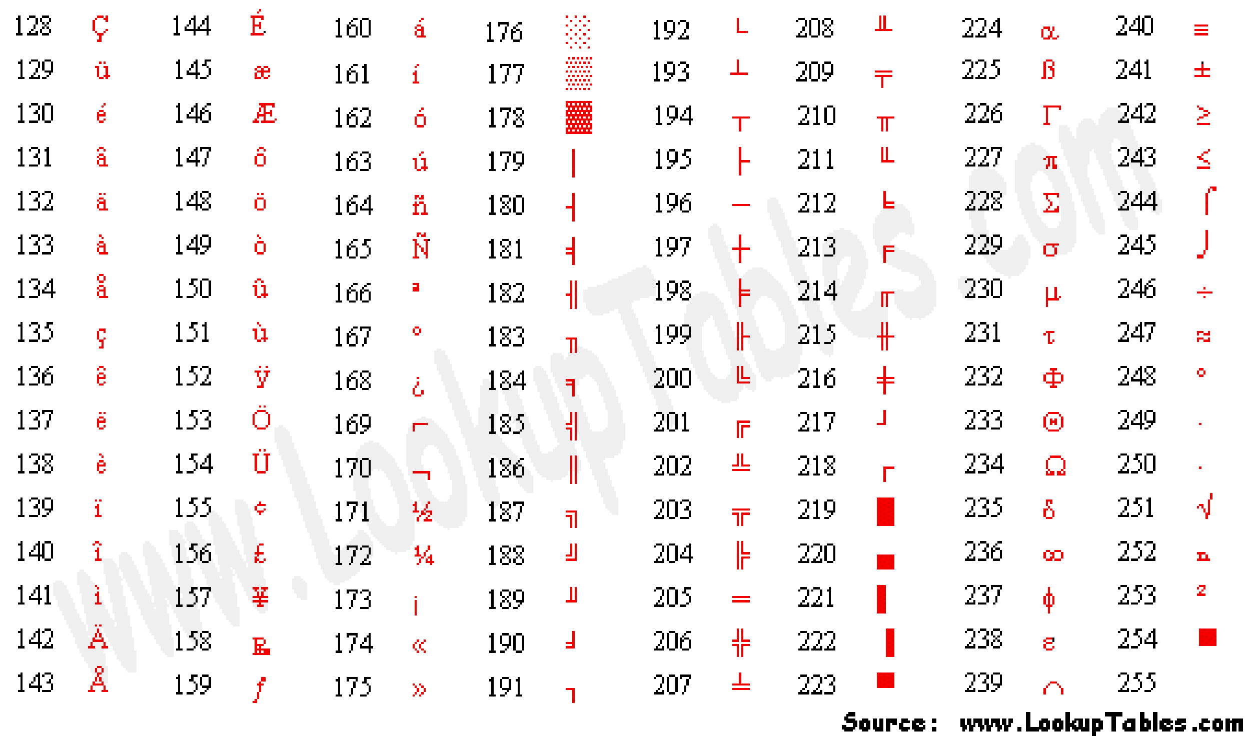
Part II

Kurt C Renderer

This an interactive (mouse & keyboard listener) ASCII renderer is a fast C renderer based on matrix manipulation in single buffer dimension, it is based on three layers:

- ASCII Canvas: The primary canvas (used for direct string insertion) one-dimensional buffer (length: WxH+1).
- UTF8 Canvas: The secondary canvas (used to translate ASCII to UTF8 (strict 24bit), (so length: 3xWxH+1).
- Color Canvas: The top-level canvas that gets color from the structured array and prints it simultaneously with buffer.

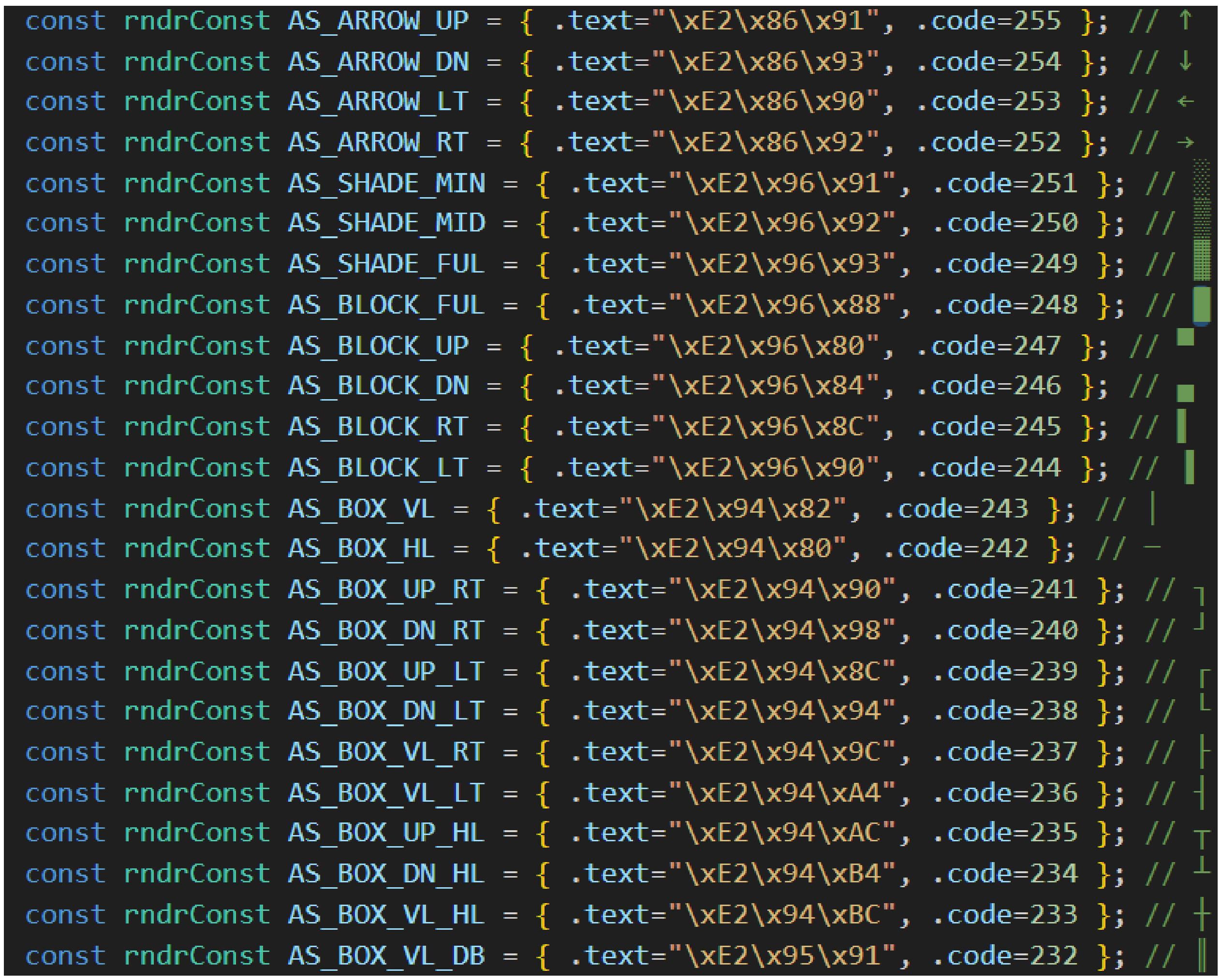
The idea behind using UTF8 is the global use of Unicode terminals, and to simplify border drawing, by getting the 40 most important ASCII art characters and add them in structure that allows inserting **8bit** non-Unicode characters directly in the primary canvas without shifting, then using **toUTF8(char *str)** to escalate **8bit** to **24bit** chars by translating table chars to **24bit** (3 Byte UTF8) by their ids, and filling last two bytes of normal ASCII chars (alpha Digits) by **0x01**, and each program has its canvas with two screens (Unicode canvas)...
Ex:



128	Ç	144	É	160	á	176	░	192	└	208	╨	224	α	240	≡
129	ü	145	æ	161	í	177	▒	193	┴	209	╤	225	ß	241	±
130	é	146	Æ	162	ó	178	▓	194	┬	210	╥	226	Γ	242	≥
131	â	147	ô	163	ú	179	│	195	├	211	╙	227	π	243	≤
132	ä	148	ö	164	ñ	180	┤	196	─	212	╘	228	Σ	244	⌠
133	à	149	ò	165	Ñ	181	╡	197	┼	213	╒	229	σ	245	⌡
134	å	150	û	166	ª	182	╢	198	╞	214	╓	230	µ	246	÷
135	ç	151	ù	167	º	183	╖	199	╟	215	╫	231	τ	247	≈
136	ê	152	ÿ	168	¿	184	╕	200	╚	216	╪	232	Φ	248	°
137	ë	153	Ö	169	⌐	185	╣	201	╔	217	┘	233	Θ	249	∙
138	è	154	Ü	170	¬	186	║	202	╩	218	┌	234	Ω	250	·
139	ï	155	¢	171	½	187	╗	203	╦	219	█	235	δ	251	√
140	î	156	£	172	¼	188	╝	204	╠	220	▄	236	∞	252	ⁿ
141	ì	157	¥	173	¡	189	╜	205	═	221	▌	237	φ	253	²
142	Ä	158	₧	174	«	190	╛	206	╬	222	▐	238	ε	254	■
143	Å	159	ƒ	175	»	191	┐	207	╧	223	▀	239	∩	255	

Source: www.LookupTables.com

Figure 1: Extended ASCII chars used in Table formatting



```
const rndrConst AS_ARROW_UP = { .text="\xE2\x86\x91", .code=255 }; // ↑
const rndrConst AS_ARROW_DN = { .text="\xE2\x86\x93", .code=254 }; // ↓
const rndrConst AS_ARROW_LT = { .text="\xE2\x86\x90", .code=253 }; // ←
const rndrConst AS_ARROW_RT = { .text="\xE2\x86\x92", .code=252 }; // →
const rndrConst AS_SHADE_MIN = { .text="\xE2\x96\x91", .code=251 }; // ░
const rndrConst AS_SHADE_MID = { .text="\xE2\x96\x92", .code=250 }; // ▒
const rndrConst AS_SHADE_FUL = { .text="\xE2\x96\x93", .code=249 }; // ▓
const rndrConst AS_BLOCK_FUL = { .text="\xE2\x96\x88", .code=248 }; // █
const rndrConst AS_BLOCK_UP = { .text="\xE2\x96\x80", .code=247 }; // ▀
const rndrConst AS_BLOCK_DN = { .text="\xE2\x96\x84", .code=246 }; // ▄
const rndrConst AS_BLOCK_RT = { .text="\xE2\x96\x8C", .code=245 }; // ▌
const rndrConst AS_BLOCK_LT = { .text="\xE2\x96\x90", .code=244 }; // ▐
const rndrConst AS_BOX_VL = { .text="\xE2\x94\x82", .code=243 }; // │
const rndrConst AS_BOX_HL = { .text="\xE2\x94\x80", .code=242 }; // ─
const rndrConst AS_BOX_UP_RT = { .text="\xE2\x94\x90", .code=241 }; // ┐
const rndrConst AS_BOX_DN_RT = { .text="\xE2\x94\x98", .code=240 }; // ┘
const rndrConst AS_BOX_UP_LT = { .text="\xE2\x94\x8C", .code=239 }; // ┌
const rndrConst AS_BOX_DN_LT = { .text="\xE2\x94\x94", .code=238 }; // └
const rndrConst AS_BOX_VL_RT = { .text="\xE2\x94\x9C", .code=237 }; // ├
const rndrConst AS_BOX_VL_LT = { .text="\xE2\x94\xA4", .code=236 }; // ┤
const rndrConst AS_BOX_UP_HL = { .text="\xE2\x94\xAC", .code=235 }; // ┬
const rndrConst AS_BOX_DN_HL = { .text="\xE2\x94\xB4", .code=234 }; // ┴
const rndrConst AS_BOX_VL_HL = { .text="\xE2\x94\xBC", .code=233 }; // ┼
const rndrConst AS_BOX_VL_DB = { .text="\xE2\x95\x91", .code=232 }; // ║
```

Figure 2: Structured data in C (of ASCII art)

Part III

BTree

A B-tree is a self-balancing search tree that maintains sorted data and allows searches, sequential access, insertions, and deletions in logarithmic time. It is widely used in databases and filesystems to manage large datasets that do not fit entirely in memory.
A BTree of order t (minimum degree) is a tree that satisfies the following properties:

- Every node can have at most $2t - 1$ keys.
- Every non-root node has at least $t - 1$ keys.
- The root has at least one key (unless it's an empty tree).
- All leaves are at the same level (B-tree is perfectly balanced).
- Keys within a node are sorted in ascending order.
- Nodes have $n + 1$ children, where n is the number of keys in the node.

Part IV

PoW

Proof of work (PoW) is form of cryptographic proofs that allows validating new generated Blockchains with high transparency in non-centralized transactions, it is used often in cryptocurrencies like Bitcoin, and it is based on SHA256 (Secure Hashing Algorithm of 256bit) algorithm (mining), where it checks for a specific number of leading zeroes in the generated hash in bruteforce nonce hashing, btw the first who validate the hash the first who get the reward.
SHA-256 (Secure Hash Algorithm 256-bit) is a cryptographic hash function that produces a fixed-size 256-bit (32-byte) hash value from an input of any size. It is widely used in security applications, including data integrity verification, password storage, and blockchain technology. This hash is represented in Hexadecimal format, and its role in PoW is hashing data and check if its leading zeros meets the difficulty set by the blockchain network.

Diff: 5 0000000000000756af69e2ffbdb930261873cd71 Valid
Data: “PreviousHash;Transactions;Time;Nonce”

Part V

Conclusion

We conclude this project by enumerating efforts and sacrifices done (by concerned parts)...
Even if it is not fully completed (ChatGPT is stubborn tool that wasted our times in first week), it is from scratch in pure C code even ncurses and notcurses were avoided in this project.

Thanks!!