UNIVERSITATEA DIN BUCUREȘTI
VIRTUTE ET SAPIENTIA

LazyWGET

Universitatea din București
Departamentul de Informatică
Instrumente și tehnici de bază în informatică
2025-2026

STUDENTS:
CRISTIAN BUDALĂ & RUSLAN GAITUR

GROUP:
143

1. Cerință

Write a shell program (script) named lwget that uses the wget command NON-RECURSIVELY to download HTML documents from the internet to the local computer.
lwget downloads a SINGLE HTML file from the internet at a time using the wget command. It then parses the content of that file and recursively downloads the HTML resources referenced within the document, emulating the behavior of the wget -r command. The difference compared to wget -r is that the recursive downloading of resources referred to in the HTML document is done lazily, in the sense that all HTML tags that define resources to be downloaded are defined as promises. These promises are only evaluated when the lwget command forces them through a new call.

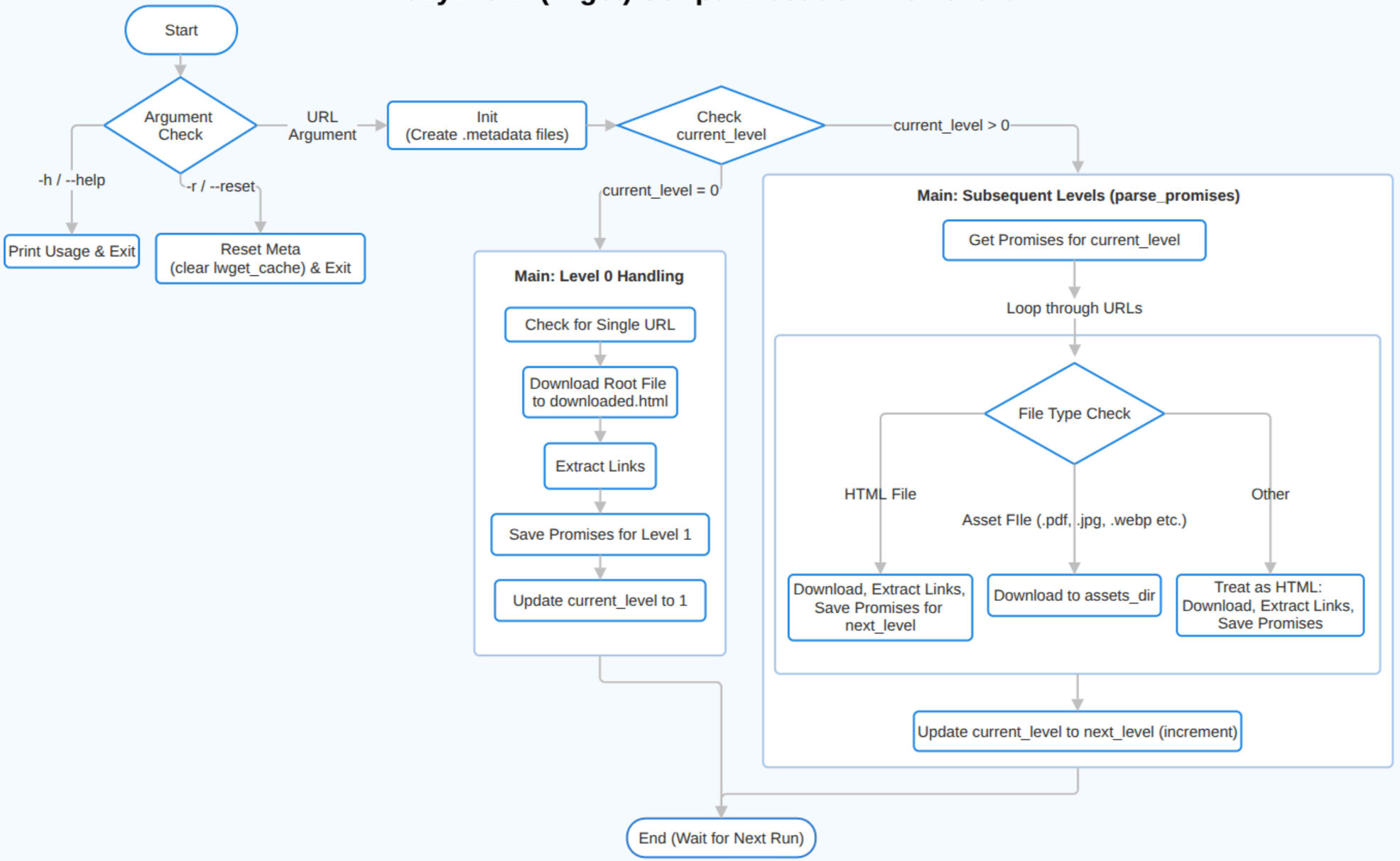
Specifically, each command lwget forces the evaluation of the current level of recursion in the document tree:

- The first call downloads the HTML file furnished as a parameter in the command line (the root of the document tree).
- The second call parses this file and locally downloads the documents referred to by the HTML file.
- The third call parses all the HTML documents downloaded in the previous stage and downloads the HTML documents referred to by those documents, and so on.

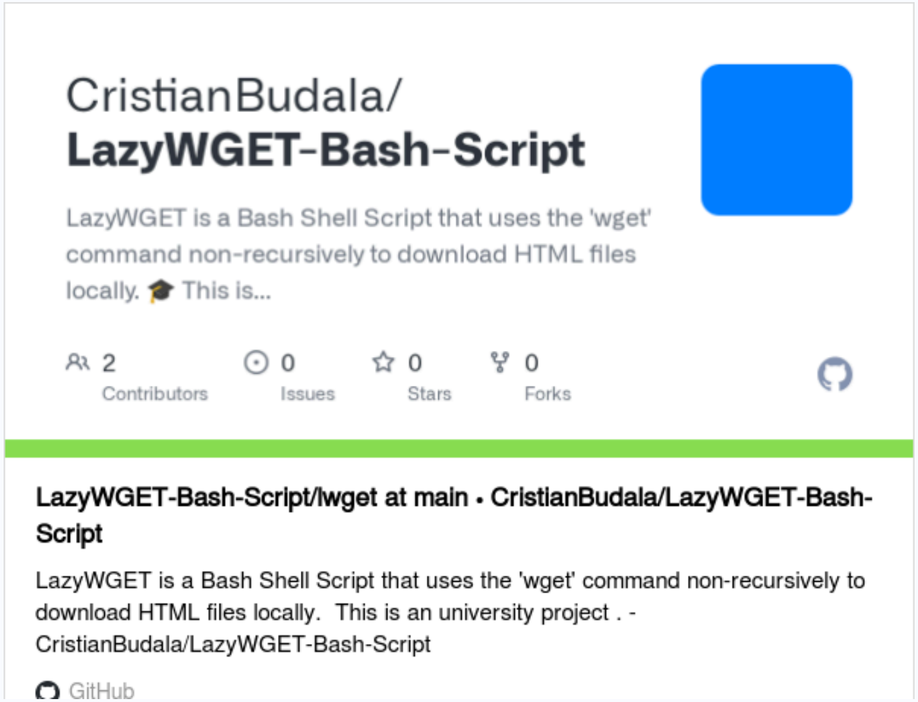
After each stage, you can use common commands like ls or tree to check the behavior of the lwget command.

2. Structura proiectului

LazyWGET(lwget) Script Execution Flowchart

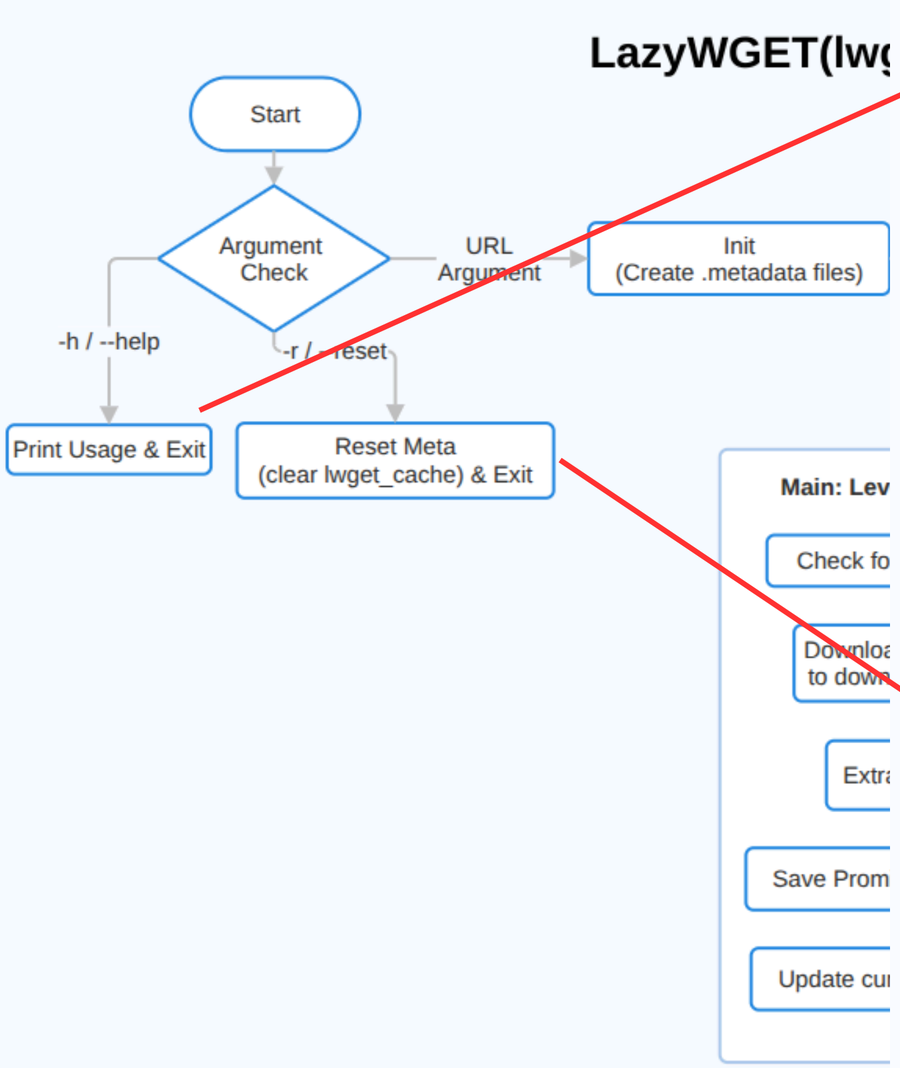


3. Codul


CristianBudala/
LazyWGET-Bash-Script
LazyWGET is a Bash Shell Script that uses the 'wget' command non-recursively to download HTML files locally. 🎓 This is...
2
Contributors
0
Issues
0
Stars
0
Forks
LazyWGET-Bash-Script/lwget at main • CristianBudala/LazyWGET-Bash-Script
LazyWGET is a Bash Shell Script that uses the 'wget' command non-recursively to download HTML files locally. This is an university project . - CristianBudala/LazyWGET-Bash-Script
GitHub

```bash
if [[ "$1" == "-h" || "$1" == "--help" ]]; then
    usage
    exit 0
fi

if [[ "$1" == "-r" || "$1" == "--reset" ]]; then
    reset_meta
    exit 0
fi

if [ ! -d "$WORK_DIR" ] || [ ! -f "$LEVEL_FILE" ]; then
    if [ $# -eq 0 ]; then
        echo "You must enter an URL!"
        echo "Usage: $0 <URL>"
        exit 1
    fi

    if [ $# -gt 1 ]; then
        echo "You must enter a single <URL>!"
        echo "Usage: $0 <URL>"
        exit 1
    fi
fi

init "$@"
main "$@"
```

Verificarea argumentului în caz de “help” sau “reset”.
(înainte de intrarea în init() sau main())

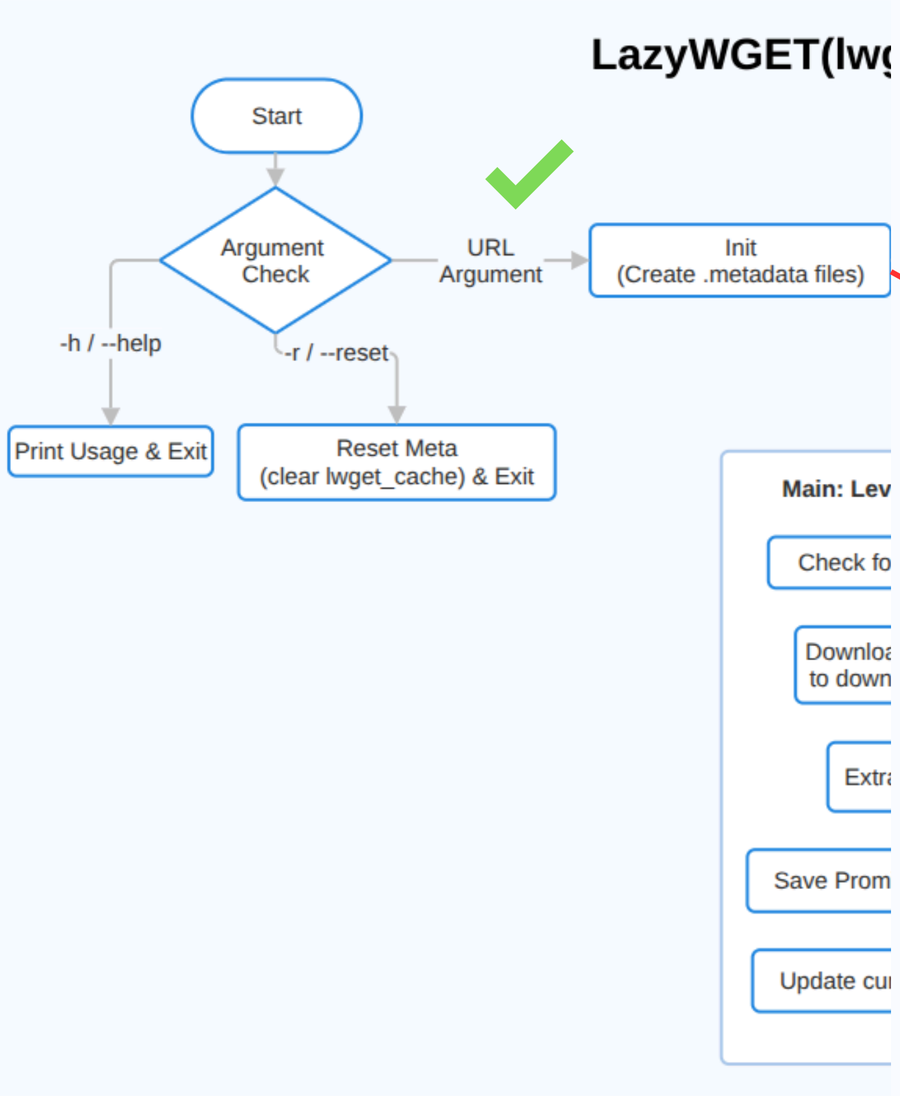



```bash
usage(){
    cat << EOF

USAGE:
   ./lwget <URL>

OPTIONS:
   <URL>          The base URL where download starts. Only HTTP and HTTPS are accepted. (REQUIRED)
   -h, --help     Print help instructions.
   -r, --reset    Deletes working directory (lwget_cache) and resets progress.

DEPENDENCIES:
   wget, grep, awk, cut

EOF
}

if [[ "$1" == "-h" || "$1" == "--help" ]]; then
    usage
    exit 0
fi
```

```bash
reset_meta() {
    if [ -d "$WORK_DIR" ]; then
        rm -rf "$WORK_DIR"
        echo -e "Cleared working directory: $WORK_DIR\n"
    else
        echo -e "No working directory to clear.\n"
    fi
    exit 0
}

if [ "$1" == "-r" ]; then
    reset_meta
fi
```

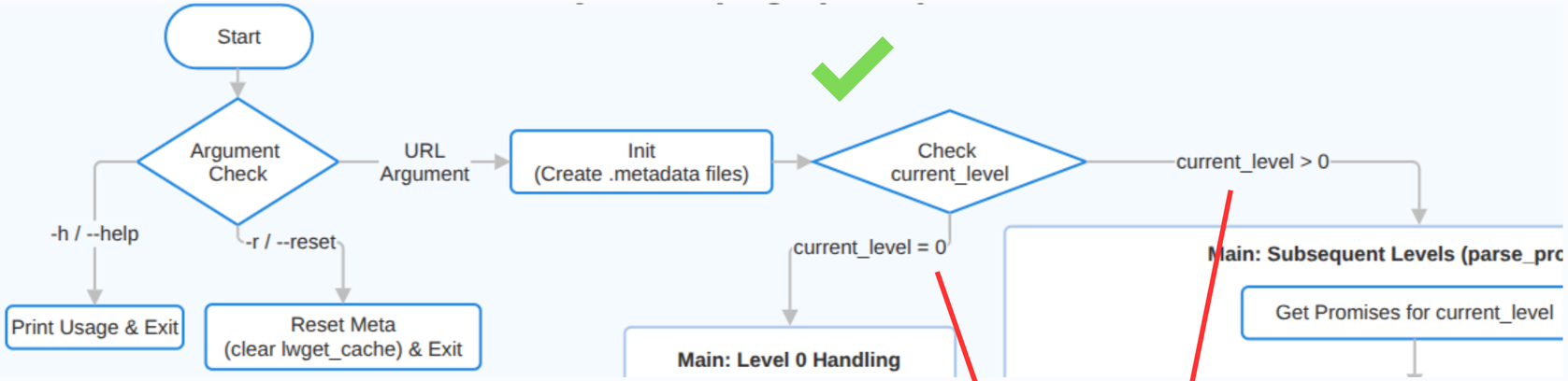
```bash
if [[ "$1" == "-h" || "$1" == "--help" ]]; then
    usage
    exit 0
fi

if [[ "$1" == "-r" || "$1" == "--reset" ]]; then
    reset_meta
    exit 0
fi

if [ ! -d "$WORK_DIR" ] || [ ! -f "$LEVEL_FILE" ]; then
    if [ $# -eq 0 ]; then
        echo "You must enter an URL!"
        echo "Usage: $0 <URL>"
        exit 1
    fi

    if [ $# -gt 1 ]; then
        echo "You must enter a single <URL>!"
        echo "Usage: $0 <URL>"
        exit 1
    fi
fi

init "$@"
main "$@"
```

Verificare pentru conformitatea argumentelor introduse.
(înainte de intrarea in înit() sau main())



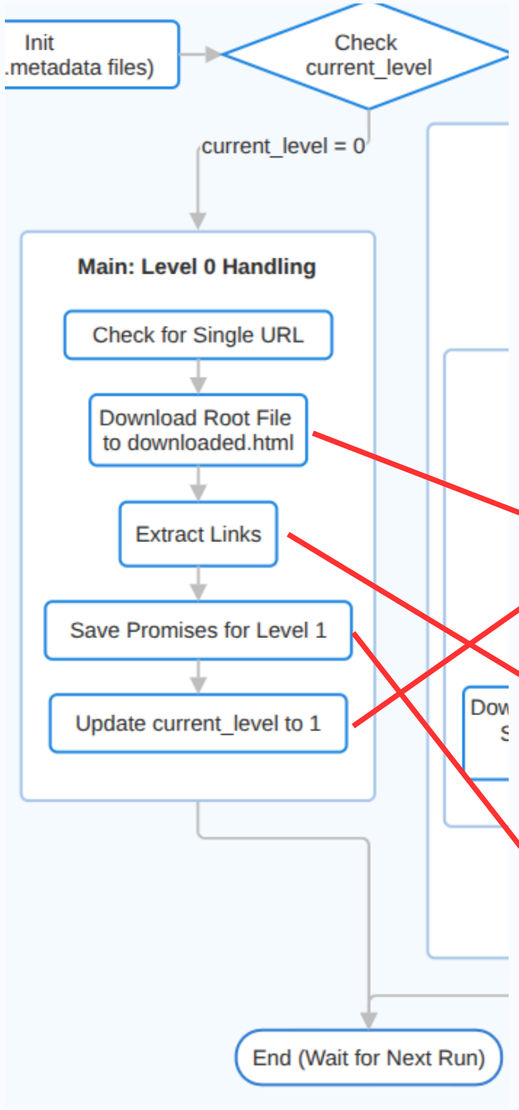

```
### CONFIG. FILES ###
WORK_DIR="./lwget_cache"
DOWNLOADS_DIR="$WORK_DIR/downloads"
METADATA_DIR="$WORK_DIR/.metadata" # hidden file
PROMISES_FILE="$METADATA_DIR/promises.txt"
LEVEL_FILE="$METADATA_DIR/current_level.txt"
# ------------------------------------------------------------
```

```
init(){
    mkdir -p "$DOWNLOADS_DIR"
    mkdir -p "$METADATA_DIR"

    if [ ! -f "$LEVEL_FILE" ]; then
        echo "0" > "$LEVEL_FILE"
        echo "Initialized level 0"
    fi

    if [ ! -f "$PROMISES_FILE" ]; then
        touch "$PROMISES_FILE"
        echo -e "Initialized promises file\n"
    fi
}
```




```bash
get_current_level(){
    cat "$LEVEL_FILE"
}
```

```bash
main(){
    local url="$1"
    local current_level=$(get_current_level)

    if [ "$current_level" -eq 0  ]; then
        echo -e "--- Level 0: Downloading root file ---\n"
        local output_file="$DOWNLOADS_DIR/downloaded.html"

        download_file "$url" "$output_file"
        echo "Download complete!"
        echo ""

        local urls=$(extract_links "$output_file")
        echo "Found links:"
        echo "$urls"
        echo ""

        save_promises "1" "$urls"
        echo "Promises saved to $PROMISES_FILE"
        echo "1" > "$LEVEL_FILE"
    else
        echo -e "--- Level $current_level: Processing promises ---\n"
        parse_promises
    fi
}
```



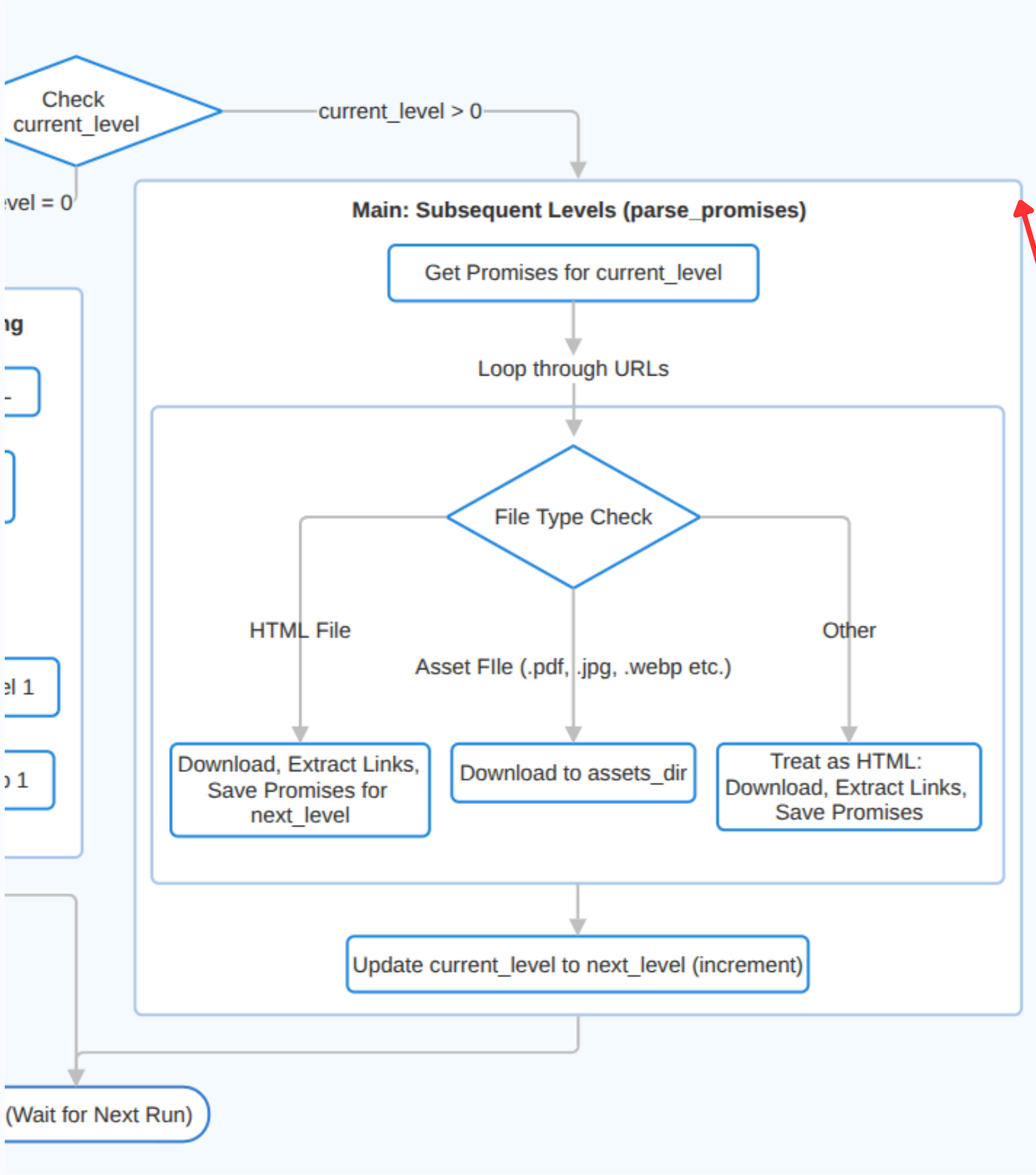

```bash
main(){
    local url="$1"
    local current_level=$(get_current_level)

    if [ "$current_level" -eq 0  ]; then
        echo -e "--- Level 0: Downloading root file ---\n"
        local output_file="$DOWNLOADS_DIR/downloaded.html"

        download_file "$url" "$output_file"
        echo "Download complete!"
        echo ""

        local urls=$(extract_links "$output_file")
        echo "Found links:"
        echo "$urls"
        echo ""

        save_promises "1" "$urls"
        echo "Promises saved to $PROMISES_FILE"
        echo "1" > "$LEVEL_FILE"
```

```bash
download_file(){
    local url="$1"
    local output_file="$2"
    wget -q -O "$output_file" "$url"
}
```

```bash
extract_links(){
    local filename="$1"
    grep -o 'href="http[^"]*"' "$filename" | awk -F'"' '{print $2}'
}
```

```bash
save_promises(){
    local level="$1"
    local urls="$2"

    while read -r url; do
        if ! grep -Fq "|$url" "$PROMISES_FILE"; then
            echo "$level|$url" >> "$PROMISES_FILE"
        fi
    done <<< "$urls"
}
```

- Dacă argumentul introdus este un URL valid, după apelarea funcției init care va crea directoarele de lucru, apelăm funcția main.
- Inițial, aceasta va descarcă local fișierul HTML de la linkul introdus de utilizator utilizând funcția “download_file”, urmând ca acesta să devină “fișierul rădăcină” (Level 0).
- Vom extrage noile linkuri din el utilizând funcția “extract_links”, le vom salva drept “promisiuni” cu funcția “save_promises” și vom incrementa nivelul curent de adâncime, stocat in fișierul “LEVEL_FILE”.




```bash
main(){
    local url="$1"
    local current_level=$(get_current_level)

    if [ "$current_level" -eq 0  ]; then
        echo -e "--- Level 0: Downloading root file ---\n"
        local output_file="$DOWNLOADS_DIR/downloaded.html"

        download_file "$url" "$output_file"
        echo "Download complete!"
        echo ""

        local urls=$(extract_links "$output_file")
        echo "Found links:"
        echo "$urls"
        echo ""

        save_promises "1" "$urls"
        echo "Promises saved to $PROMISES_FILE"
        echo "1" > "$LEVEL_FILE"
    else
        echo -e "--- Level $current_level: Processing promises ---\n"
        parse_promises
    fi
}
```

```bash
parse_promises() {
    local current_level=$(get_current_level)
    local next_level=$((current_level + 1))

    for item in $(get_promises_for_level "$current_level")
    {
        echo $item
        url=$item
        if [[ "$item" =~ \.(html)$ ]]; then
            resource_name=$(basename "$url")
            local assets_dir="$DOWNLOADS_DIR/assets"
            mkdir -p "$assets_dir"
            wget -q -nc -x -P "$assets_dir" "$url"
            local output_file="$DOWNLOADS_DIR/downloaded.html"
            download_file "$url" "$output_file"
            local urls=$(extract_links "$output_file")
            save_promises "$next_level" "$urls"
        elif [[ "$item" =~ \.(pdf|jpg|png|zip|css|js|svg|json|xml|txt|map|jpeg|gif|webp|bmp|ico|tiff|doc|
docx|xls|xlsx|csv|ppt|pptx|mp3|mp4|wav|webm|m4a)$ ]]; then
            resource_name=$(basename "$url")
            local assets_dir="$DOWNLOADS_DIR/assets"
            mkdir -p "$assets_dir"
            wget -q -nc -x -P "$assets_dir" "$url"
        else
            local output_file="$DOWNLOADS_DIR/downloaded.html"
            download_file "$url" "$output_file"
            local urls=$(extract_links "$output_file")
            save_promises "$next_level" "$urls"
        fi
    }

    echo -e "\nFinished parsing level $current_level... \n"
    echo "$next_level" > "$LEVEL_FILE"
}
```

```bash
get_promises_for_level(){
    local level="$1"
    grep "^$level|" "$PROMISES_FILE" | cut -d "|" -f 2
}
```

- Funcția parse_promises preia toate URL-urile salvate pentru nivelul curent, le procesează pe rând și pregătește nivelul următor al descărcării.
- Mai exact, ea citește nivelul actual din fișier (current_level), calculează nivelul următor (next_level), apoi, pentru fiecare URL din lista de promisiuni:
  - dacă este fișier HTML, îl descarcă, îi extrage link-urile și salvează noile link-uri ca promisiuni pentru nivelul următor;
  - dacă este fișier de tip resursă (imagini, PDF, arhive, audio, video etc.), îl descarcă direct în directorul de assets;
  - dacă nu se potrivește niciunui tip explicit, îl tratează ca pagină HTML.
- La final, afișează că nivelul curent a fost procesat și actualizează nivelul în fișierul de tracking.

4. Testăm scriptul

(https://fmi.unibuc.ro/)


Mulțumim!