ARP Optimizations

M.Shaaban

Università di Genova

DIBRIS DIPARTIMENTO
DI INFORMATICA, BIOINGEGNERIA,
ROBOTICA E INGEGNERIA DEI SISTEMI

Storing and Reading Configurations

- Macros could be used to have constant configurations
- Changing a value requires a rebuild

```
#define NUM_PROCESSES 4
#define NUM_PROC_GENERATED 1
#define PROCESS_SLEEPS_US {100000, 500000, 300000, 350000}
#define PROCESS_SIGNAL_INTERVAL 5
#define WATCHDOG_SLEEP_US 200000
#define PROCESS_TIMEOUT_S 10
```

Storing and Reading Configurations

- We can store the configuration in text format
- We read and parse the configuration file in real-time

```
# appsettings.conf
# Configuration File for the Game. Lines
without tokens are ignored
PlayerName=Mohamad
Difficulty=Medium
StartingLevel=5
```

Read the Configuration File

```
FILE *file;
char line[MAX_LINE_LENGTH];
char playerName[MAX_LINE_LENGTH];
char difficulty[MAX_LINE_LENGTH];
int startingLevel = 0;

file = fopen("appsettings.conf", "r");//read mode "r", write "w"

if (file == NULL) {
    perror("Error opening the file");
    return EXIT_FAILURE;//1
}
```

Parsing the Configuration

```
// loop on all the lines
    while (fgets(line, sizeof(line), file)) {
        // Removing line terminator
        line[strcspn(line, "\n")] = 0;

        // Tokenize the line to extract key-value pairs
        char *token = strtok(line, "=");
        if (token != NULL) {
            if (strcmp(token, "PlayerName") == 0) {
                token = strtok(NULL, "=");
                strcpy(playerName, token);
            } else if (strcmp(token, "Difficulty") == 0) {
                token = strtok(NULL, "=");
                strcpy(difficulty, token);
            } else if (strcmp(token, "StartingLevel") == 0) {
                token = strtok(NULL, "=");
                startingLevel = atoi(token);
            }
        }
    }

    fclose(file);//closing the file
```

Is it Really Efficient?

- Not possible to represent objects
- Not possible to have nested objects
- Not easy to have arrays
- Parsing configuration is not the most joyful thing :B
- Alternatively, we can use **JSON** data structure

JavaScript Object Notation

- Text-based way to store data (**Not Binary**)
- Could be used for communication or configuration
- Language independent
- Can represent complex structures
- It is not a programming language

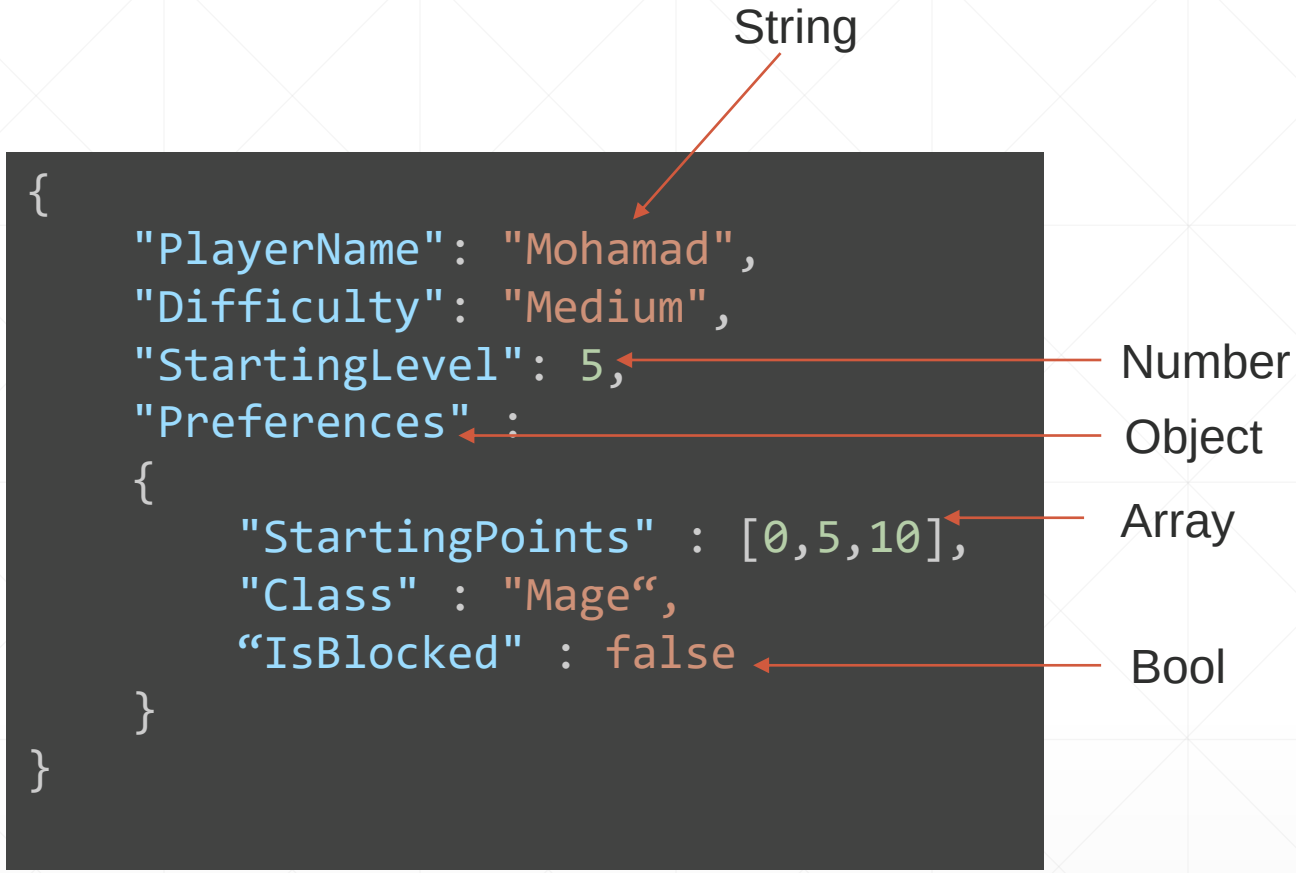
JSON Syntax

- Key-Value pairs separated by **:** `{"PlayerName": "Mohamad"}`
- Pairs are separated by commas
- Objects are contained by **{}**
- It can represent arrays
- Arrays contained by **[]**
- It can hold nested objects
- Key can’t contain spaces

```
{
    "PlayerName": "Mohamad",
    "Difficulty": "Medium",
    "StartingLevel": 5,
    "Preferences" :
    {
        "StartingPoints" : [0,5,10],
        "Class" : "Mage"
    }
}
```

JSON Data Types

- String
- Number (Int/floating point)
- Nested object
- NULL
- Bool
- Array of any of the above mentioned



Parsing JSON in C

- To parse JSON, we will be using third-party library
- Library is called **cJSON**
- We can install the pre-built version **sudo apt install libcjson-dev**
- We can build it from source

Using cJSON – VS Code

- We must include the header file `#include "cJSON/cJSON.h"`
- With VS we must modify **tasks.json**
- Tell the linker to link cJSON dynamic (**.so**) or static (**.a**) lib `-lcjson`
- Link the header file `-I${fileDirname}/cJSON/cJSON.h`

```
"args": [       "-fdiagnostics-color=always",
                "-g",
                "${file}",
                "-o",
                "${fileDirname}/${fileBasenameNoExtension}",
                "-I${fileDirname}/cJSON/cJSON.h",
                "-lcjson"],
```

Using cJSON – Make

- We must include the header file `#include "cJSON/cJSON.h"`
- Link cJSON `-lcjson`
- Export cJSON Directory `export cjsonDirname=/usr/include/cjson`
- Point to the header file `-I${cjsonDirname}/cJSON.h`
- Command should look like this

```
gcc file.c -o output –lcjson  -I${cjsonDirname}/cJSON.h
```

Parsing JSON in C

- We start by reading the whole file
- Storing it in a buffer
- Not to forgot closing the file
- Now we are ready to parse it

```
FILE *file;
char jsonBuffer[MAX_FILE_SIZE];
char playerName[MAX_LINE_LENGTH];
char difficulty[MAX_LINE_LENGTH];

file = fopen("appsettings.json", "r");

if (file == NULL) {
    perror("Error opening the file");
    return EXIT_FAILURE;//1
}

int len = fread(jsonBuffer, 1,
   sizeof(jsonBuffer), file);
fclose(file);
```

Parsing JSON in C

```
cJSON *json = cJSON_Parse(jsonBuffer);// parse the text to json object
// don’t forgot to free cJSON at the end cJSON_Delete(json)
if (json == NULL)
{
    perror("Error parsing the file");
    return EXIT_FAILURE;
}
```

Parsing JSON in C

```
{
    "PlayerName": "Mohamad",
    "Difficulty": "Medium",
    "StartingLevel": 5,
    "Preferences" :
    {
        "StartingPoints" : [0,5,10],
        "Class" : "Mage"
    }
}
```

```
strcpy(gameConfig->difficulty, cJSON_GetObjectItemCaseSensitive(json,
"PlayerName")->valuestring); //Mohamad
strcpy(gameConfig->playerName, cJSON_GetObjectItemCaseSensitive(json,
"Difficulty")->valuestring); //Medium
gameConfig->startingLevel = cJSON_GetObjectItemCaseSensitive(json,
"StartingLevel")->valueint; //5
```

String

Key

Int

Parsing JSON in C

```
{
    "PlayerName": "Mohamad",
    "Difficulty": "Medium",
    "StartingLevel": 5,
    "Preferences" :
    {
        "StartingPoints" : [0,5,10],
        "Class" : "Mage"
    }
}
```


```
cJSON *preferences = cJSON_GetObjectItemCaseSensitive(json,"Preferences");
//this is a child of parent object json
printf("Class: %s\n", cJSON_GetObjectItemCaseSensitive(preferences,
"Class")->valuestring);
```

String

Parsing JSON in C

```
{
    "PlayerName": "Mohamad",
    "Difficulty": "Medium",
    "StartingLevel": 5,
    "Preferences" :
    {
        "StartingPoints" : [0,5,10],
        "Class" : "Mage"
    }
}
```

```
cJSON *numbersArray = cJSON_GetObjectItem(preferences, "StartingPoints");//this is an array
int arraySize = cJSON_GetArraySize(numbersArray); //Size of the array 3
for (int i = 0; i < arraySize; ++i) {
    cJSON *element = cJSON_GetArrayItem(numbersArray, i); //One int element
    if (cJSON_IsNumber(element)) {
        printf("StartingPoint [%d]: %d\n", i+1 ,element->valueint);
     }
 }
```

StartingPoint [1]: 0
StartingPoint [2]: 5
StartingPoint [3]: 10

Optimizations

Macros

- Macro could be used to store constants
- Could be used to store codes
- It is evaluated during preprocessing stage, before compilation
- Substituted during the preprocessing

```
#define LOGCONFIG(config)                                          \
    {                                                              \
        printf("Player Name: %s\n", config->playerName);           \
        printf("Difficulty: %s\n", config->difficulty);            \
        printf("Starting Level: %d\n", config->startingLevel); \
    }
```

Macros

- They are faster than normal functions

BUT!!!

- Could not be debugged
- Not type-checked

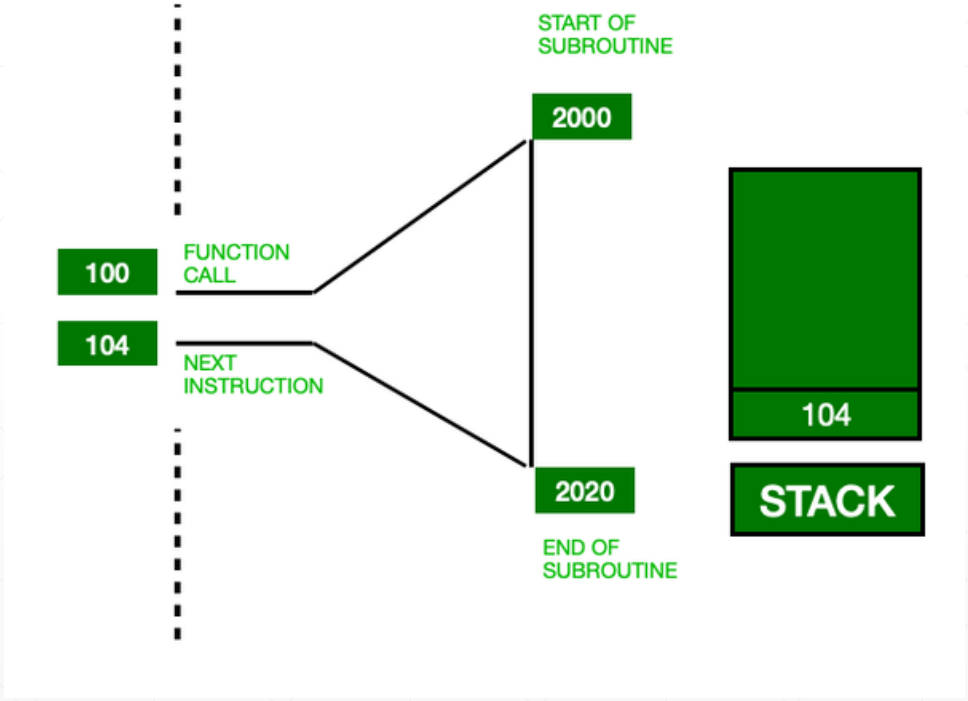
Inline keyword

- Type-checked functions
- Could be debugged
- It is evaluated during compilation stage
- Inline doesn’t guarantee that the compiler will replace the call with the code

```
static inline void logConfig(const Config *config)
{
    printf("Player Name: %s\n", config->playerName);
    printf("Difficulty: %s\n", config->difficulty);
    printf("Starting Level: %d\n", config->startingLevel);
}
```

Why Macro or Inline?

- Calling a normal function push to the stack return address
- Subroutine is going to be executed
- Address will be popped from stack
- Code will resume executing



Why Macro or Inline?

- Using Macro or Inline will speed up our code (no stack calls)
- Will allow us to print some logs with the line number

```
[38] at /home/mrshaaban/class/totalsense.c:
Player Name: Medium
Difficulty: Mohamad
```

Macro/Inline deep dive

```
.L4:
        movq    %rax, %rsi
        leaq    .LC8(%rip), %rdi
        movl    $0, %eax
        call    printf@PLT
        movq    -344(%rbp), %rax
        addq    $100, %rax
        movq    %rax, %rsi
        leaq    .LC9(%rip), %rdi
        movl    $0, %eax
        call    printf@PLT
        movq    -344(%rbp), %rax
        movl    200(%rax), %eax
        movl    %eax, %esi
        leaq    .LC10(%rip), %rdi
        movl    $0, %eax
        call    printf@PLT
        nop
```

```
movl    -364(%rbp), %edx
        movl    %edx, 200(%rax)
        movq    -352(%rbp), %rax
        movq    %rax, %rdi
        call    logConfig
        movq    -360(%rbp), %rax
```

```
void logConfig(const Config*)
```

```
logConfig:
.LFB6:
        movq    %rax, %rsi
        leaq    .LC0(%rip), %rdi
        movl    $0, %eax
        call    printf@PLT
        movq    -8(%rbp), %rax
        addq    $100, %rax
        movq    %rax, %rsi
        leaq    .LC1(%rip), %rdi
        movl    $0, %eax
        call    printf@PLT
        movq    -8(%rbp), %rax
        movl    200(%rax), %eax
        movl    %eax, %esi
        leaq    .LC2(%rip), %rdi
        movl    $0, %eax
        call    printf@PLT
        nop
        leave
        ret
```

```
Static inline void logConfig(const Config*)
```

gcc -S -o asm_output.s app.c

Logging Error Line

- Macros are replaced on preprocessing stage → `__LINE__` `__FILE__` will be replaced with the actual values

```
32      conf_ptr gameConfig = malloc(sizeof(conf_ptr));
33
34      strcpy(gameConfig->difficulty, cJSON_GetObjectItemCaseSensitive(json, "PlayerName")->valuestring);
35      strcpy(gameConfig->playerName, cJSON_GetObjectItemCaseSensitive(json, "Difficulty")->valuestring);
36      gameConfig->startingLevel = cJSON_GetObjectItemCaseSensitive(json, "StartingLevel")->valueint;
37
38      LOG(gameConfig);
```

Output

```
[38] at /home/mrshaaban/class/totalsense.c:
Player Name: Medium
Difficulty: Mohamad
Starting Level: 5
```

Replaced by 38

Replaced by /home/mrshaaban/class/totalsense.c

```
#define LOG(conf)                                                   \
    {                                                               \
        printf("[%d] at %s:\n", __LINE__, __FILE__);                \
        logConfig(gameConfig);                                      \
    }
#endif
```

Sample Code