سامانه اعزام واحدهای اضطراری

پروژه پایانترم درس مبانی برنامه
نویسی ترم بهار 1404

استاد درس: دکتر ذاکری

طراحان پروژه:
عماد فردوسی
علرضا نیکوی
امیرحسین عقیقی

طراح این بخش:
شکرالله ضیایی

تاریخ تحول روژه:
15 تیر 1404

فهرست مطالب

1.مقدمه

به روژه شبیه سازی اعزام اضطراری خوش آمدید! ان فقط یک کلیف برنامه نویسی دیگر نیست - آن یک فرصت عملی است ا مهارتهای برنامه نویسی C خود را با ساختن یک دنیای کوچک که در آن آتش نشانها، بیمارستانها و ایستگاههای پلیس برای رسیدگی به موارد اضطراری در یک شهر شلوغ باهم کار میکنند، به کار ببرید. از طریق آن پروژه، موقعیتهای واقعی را شبیه سازی خواهید کرد که در آن هر تصمیمی مهم است، منابع محدود هستند و زمانبندی بسیار مهم است. شما مسئولیت مدیریت حوادث، اعزام واحدهای اورژانس و اطمینان از اجرای روان همه چیز را به نوبت بر عهده خواهید گرفت.

هدف آن روژه کمک به شما در به کارگیری هر آنچه در مورد برنامه نویسی C آموخته اید به روشی عملی و جذاب است. شما با داده های ساختار یافته کار خواهید کرد، جریان برنامه را مدیریت خواهید کرد و نوشتن کد تمیز و ماژولار را که به راحتی قابل درک و گسترش است، تمرن خواهید کرد. تا زمانی که کار خود را تمام کنید، فقط یک برنامه کاربردی نخواهید داشت - سیستی ایجاد خواهید کرد که رفتارها و چالشهای دنیای واقعی را مدل سازی میکنید.

برای شروع، باید از قبل با اصول اولیه C آشنا باشید: کار با ساختارها و اشاره گرها، استفاده از آرایه ها و حلقه ها و نوشتن منطق شرطی. در طول مسیر، شما همچنین یاد خواهید گرفت که چگونه فایلها را برای پیکربندی و ثبت وقایع مدیریت کنید، کد خود را به ماژولهای سازمان یافته تقسیم کنید و برنامههای طراحی کنید که آزمایش و اشکال زدای آنها آسان باشد. اگر هنوز در همه ان موارد ماهر نیستید، نگران نباشید - این پروژه به گونه ای طراحی شده است که به شما کمک کند این مهارتها را گام به گام بسازید.

2.هدف پروژه

هدف این پروژه، شبیه‌سازی یک **سامانه اعزام واحدهای اضطراری** است که در شرایط بحرانی مانند حوادث جاده‌ای، آتش‌سوزی یا اتفاقات پزشکی، بتواند نزدیک‌ترین واحدهای امدادی (مانند آمبولانس، آتش‌نشانی یا پلیس) را به محل حادثه اعزام کند.
این پروژه با استفاده از زبان برنامه‌نویسی C طراحی شده و هدف آن آشنایی دانشجویان با مفاهیم پایه برنامه‌نویسی مانند:

- استفاده از ساختارهای داده‌ای مانند آرایه‌ها و ساختارها (`struct`)
- پیاده‌سازی الگوریتم‌های تصمیم‌گیری ساده
- تعامل با کاربر از طریق منوی متنی
- ثبت رویدادها در فایل لاگ
- نمایش وضعیت نقشه و موقعیت واحدها و حوادث

در کنار اهداف آموزشی، این پروژه سعی دارد تفکری سیستماتیک برای طراحی و تحلیل نرم‌افزارهای مبتنی بر زمان و عملیات‌محور (event-driven) را در دانشجو تقویت کند.

3.ساختار برنامه

برنامه از چندین فایل اصلی تشکیل شده است که هر کدام مسئول بخشی از عملکرد کلی سیستم هستند. این ساختار ماژولار باعث شده است که کد خواناتر، منظم‌تر و قابل توسعه‌تر باشد.

فایل‌های اصلی پروژه:

1. **main.c**
   نقطه‌ی شروع اجرای برنامه است. وظیفه آن مقداردهی اولیه، بارگذاری پیکربندی، آغاز شبیه‌سازی و مدیریت حلقه اصلی اجرا می‌باشد.

2. **simulation.c / simulation.h**
   این فایل مسئول کنترل حلقه‌ی شبیه‌سازی (turns) است. در هر نوبت، ورودی از کاربر گرفته شده و عملیات‌هایی مانند ثبت حادثه جدید، حرکت واحدها، بررسی پایان عملیات و به‌روزرسانی وضعیت‌ها انجام می‌شود.

3. **map.c / map.h**
   مسئول نمایش نقشه و وضعیت لحظه‌ای واحدها و حوادث است. این ماژول با استفاده از توابع خاص، نقشه‌ای از محیط را در کنسول چاپ می‌کند که شامل محل ایستگاه‌ها،

حوادث فعال و واحدهای اعزام‌شده است.

4. **unit.c / unit.h**
   منطق مربوط به واحدهای امدادی مانند آمبولانس، پلیس و آتش‌نشانی در این فایل قرار دارد. این بخش شامل تعریف ساختار هر واحد، وضعیت‌های مختلف (آزاد، در حال اعزام، بازگشت از ماموریت)، و همچنین منطق حرکت آن‌ها در نقشه است.

5. **incident.c / incident.h**
   اطلاعات مربوط به حوادث در این فایل نگهداری و پردازش می‌شود. هر حادثه شامل نوع، موقعیت، سطح بحرانی بودن و واحدهای مورد نیاز است. همچنین وضعیت پیشرفت حادثه در این ماژول بررسی می‌شود.

6. **log.c / log.h**
   برای ثبت وقایع مهم برنامه مانند شروع شبیه‌سازی، اعزام واحدها، پایان عملیات، و جزئیات هر نوبت (turn) به یک فایل متنی استفاده می‌شود. این فایل می‌تواند به‌عنوان گزارش عملکرد برنامه استفاده شود.

7. **config.c / config.h**
   این ماژول پیکربندی اولیه سیستم را از یک فایل متنی (مانند `config.txt`) می‌خواند و اطلاعات اولیه مثل اندازه نقشه، تعداد ایستگاه‌ها، و واحدهای موجود را بارگذاری می‌کند.

4.نحوه اجرای برنامه

برای اجرای برنامه شبیه‌ساز اعزام واحدهای اضطراری، ابتدا لازم است محیط توسعه مناسب مانند **VS Code** یا **CLion** و کامپایلر زبان C نصب شده باشد. سپس با استفاده از دستور `make` یا اجرای فایل اجرایی (مثلاً `basic.exe`) برنامه شروع می‌شود.

مراحل اجرا:

1. **بارگذاری پیکربندی اولیه**
   برنامه هنگام شروع، اطلاعات اولیه را از فایل پیکربندی (مثلاً `config.txt` یا `sample_config.txt`) بارگذاری می‌کند. این فایل شامل موارد زیر است:

   - ابعاد نقشه (طول و عرض)

- موقعیت ایستگاه‌ها
- تعداد و نوع واحدهای امدادی

نمایش منوی اولیه به کاربر

پس از بارگذاری تنظیمات، برنامه در نوبت صفر (Turn 0) یک منو مانند زیر نمایش می‌دهد:

=====Turn 0=====

=== MENU ===

1. Add New Incident

2. Continue Simulation

3. Exit

Enter your choice:

- **گزینه ۱:** وارد کردن یک حادثه جدید با مشخصات (نوع، شدت، مختصات)
- **گزینه ۲:** ادامه شبیه‌سازی و اجرای یک نوبت (Turn)
- **گزینه ۳:** خروج از برنامه

2. **شبیه‌سازی هر نوبت (Turn)**

در هر نوبت:

- وضعیت واحدها بررسی می‌شود (آزاد، در حرکت، در عملیات، در حال بازگشت)
- وضعیت حوادث به‌روزرسانی می‌شود
- نقشه دوباره رسم می‌شود و تغییرات روی آن قابل مشاهده است
- لاگ مربوط به این نوبت در فایل ثبت می‌شود

3. **پایان شبیه‌سازی**
   برنامه می‌تواند تا هر زمان که کاربر بخواهد ادامه پیدا کند. در پایان، گزارش کامل عملکرد واحدها و حوادث در فایل لاگ باقی می‌ماند.

در صورتی که Makefile یا فایل اجرایی درست تنظیم شده باشد، اجرای برنامه تنها با یک دستور انجام می‌شود و تمام مراحل فوق به‌صورت خودکار طی می‌شوند.

5.قابلیت‌های اضافه‌شده

وضعیت استراحت واحدها پس از عملیات

پس از انجام عملیات، واحدهای اعزامی بلافاصله آماده‌به‌کار نمی‌شوند. آن‌ها وارد حالت **استراحت (RESTING)** می‌شوند و تنها پس از گذشت چند نوبت، دوباره به وضعیت آزاد (AVAILABLE) بازمی‌گردند.
این ویژگی باعث واقع‌گرایانه‌تر شدن سیستم و جلوگیری از اعزام مکرر یک واحد در نوبت‌های پشت سر هم می‌شود.

استفاده از Git و GitHub

برای مدیریت بهتر کد و نسخه‌های پروژه، از **Git** استفاده شده و مخزن پروژه در **GitHub** قرار گرفته است. این کار باعث شده:

- تغییرات به‌صورت مرحله‌ای ذخیره شود.
- امکان همکاری تیمی فراهم باشد.
- تاریخچه‌ی توسعه پروژه قابل پیگیری باشد.

استفاده از Makefile برای مدیریت ساخت پروژه

برای آسان‌تر کردن فرایند کامپایل و اجرای پروژه، یک فایل **Makefile** نوشته شد. این فایل به صورت خودکار تمامی فایل‌های کد منبع را کامپایل کرده و فایل اجرایی نهایی را می‌سازد.

6. توضیح توابع همراه با اسکرین‌شات

در این بخش، مهم‌ترین توابع پروژه معرفی شده‌اند. این توابع نقش کلیدی در عملکرد شبیه‌سازی دارند و در بخش‌های مختلف مانند دریافت ورودی، به‌روزرسانی وضعیت، نمایش نقشه و ثبت اطلاعات استفاده می‌شوند.

```
void init_log() {
    time_t now = time(NULL);
    struct tm *t = localtime(&now);
    char filename[128];
    strftime(filename, sizeof(filename), "log_%Y%m%d_%H%M%S.txt", t);
    log_file = fopen(filename, "w");
    if (log_file == NULL) {
        perror("Failed to create log file");
        exit(1);
    }
}
```

تابع `init_log()`

این تابع مسئول **ساخت و باز کردن یک فایل لاگ با نام اختصاصی بر اساس زمان اجرای برنامه** است. در واقع به جای اینکه همیشه در یک فایل ثابت بنویسیم (که باعث از بین رفتن لاگ‌های قبلی می‌شود)، این روش کمک می‌کند هر بار اجرای برنامه، لاگ مخصوص به خودش را داشته باشد.

```
void log_simulation_start() {
    if (!log_file) return;
    log_timestamp();
    fprintf(log_file, "Simulation Started.\n");
    fflush(log_file);
}
```

تابع `log_simulation_start()`

این تابع در آغاز اجرای برنامه فراخوانی می‌شود و هدف آن ثبت زمان و پیام **"شروع شبیه‌سازی"** در فایل لاگ است. این کار کمک می‌کند تا در بررسی فایل لاگ، دقیقاً مشخص شود که شبیه‌سازی از چه زمانی شروع شده است.

```c
void load_configuration(const char *filename) {

    char line[128];
    while (fgets(line, sizeof(line), file)) {
        if (strncmp(line, "MAP_SIZE", 8) == 0) {
            sscanf(line, "MAP_SIZE %d %d", &map_width, &map_height);
        }
        // ------------------------------------------------------------------------IS DONE ------------------------------------------------------------
        // Extract department type, index, coordinates, and unit count.
        // Initialize the department fields accordingly.
        // Then, initialize each unit for the department with appropriate values.
        // Hint: Use sscanf to parse values and a loop to initialize units.

        // Check if the line defines a department
        else if (strncmp(line, "DEPARTMENT", 10) == 0) {
            char type_str[10];
            int index, x, y, unit_count;

            // Parse the department line
            sscanf(line, "DEPARTMENT %s %d %d %d %d", type_str, &index, &x, &y, &unit_count);

            if (department_count < MAX_DEPARTMENTS) {
                // Get a pointer to the next available department slot
                Department *dept = &departments[department_count];

                // Initialize department properties
                dept->type = parse_department_type(type_str); // Convert string to enum
                dept->number = index;
                dept->x = x;
                dept->y = y;
                dept->unit_count = unit_count;

                // Initialize all units within this department
                for (int i = 0; i < dept->unit_count; i++) {
                    if (i < MAX_UNITS_PER_DEPARTMENT) {
                        Unit *unit = &dept->units[i];
                        unit->type = dept->type;
                        unit->departmentNumber = dept->number;
                        unit->unitNumber = i + 1; // Units are 1-indexed
                        unit->x = dept->x;        // Start at department's location
                        unit->y = dept->y;
                        unit->state = UNIT_WAITING; // Initial state
                        unit->target_x = -1;      // No target initially
                        unit->target_y = -1;
                    }
                }
                department_count++; // Increment the global department counter
            }
        }

    }

    fclose(file);
}
```

تابع `Load_configuration("sample_cinfig.txt)`

این تابع وظیفه دارد اطلاعات اولیه مورد نیاز شبیه‌سازی را از فایل پیکربندی بارگذاری کند. این اطلاعات شامل موقعیت ایستگاه‌های پلیس، آتش‌نشانی، آمبولانس و سایر تنظیمات شبیه‌سازی است.

```c
void simulation_loop() {
    int turn_counter = 0;

    while (1) {
        if (turn_counter == 0) {
            printf("\n\n===== Turn %d =====\n", turn_counter);
            process_user_input(); // Show menu
            turn_counter++;
            continue;
        }

        // Turn without map execute.
        perform_turn_actions();

        // Showing maps, and maps with operation.
        printf("-------------------- Turn Actions --------------------\n");

        // Showing turn
        printf("==================\n");
        printf("--- Turn %d ---\n", turn_counter);
        printf("Press ENTER to proceed to the next turn...\n");

        if (turn_counter % 10 == 0) {
            process_user_input();
        }

        turn_counter++;

        // Waiting for Enter.
        int c;
        while ((c = getchar()) != '\n' && c != EOF);
    }
}
```

تابع `simulation_loop()`

این تابع قلب اصلی پروژه است و **حلقه‌ی بینهایتی** را اجرا می‌کند که در آن، عملیات شبیه‌سازی نوبت به نوبت (turn-by-turn) انجام می‌شود.
در هر نوبت، کاربر می‌تواند حادثه‌ی جدیدی وارد کند یا شبیه‌سازی را ادامه دهد و سیستم، وضعیت واحدها و حوادث را به‌روزرسانی کرده و نقشه را نمایش می‌دهد.

```c
void process_user_input() {
    int choice = 0;
    char input[100];

    while (choice != 2 && choice != 3) {
        printf("\n=== MENU ===\n");
        printf("1. Add New Incident\n");
        printf("2. Continue Simulation\n");
        printf("3. Exit\n");
        printf("Enter your choice: ");

        fgets(input, sizeof(input), stdin);
        sscanf(input, "%d", &choice);  // It gives the input.

        switch (choice) {
            case 1: {
                int type_in, priority_in, x, y;
                printf("Enter Incident Type (0:FIRE, 1:MEDICAL, 2:POLICE): ");
                fgets(input, sizeof(input), stdin);
                sscanf(input, "%d", &type_in);

                printf("Enter Priority (0:LOW, 1:MEDIUM, 2:HIGH): ");
                fgets(input, sizeof(input), stdin);
                sscanf(input, "%d", &priority_in);

                printf("Enter X and Y coordinates (e.g., 5 8): ");
                fgets(input, sizeof(input), stdin);
                sscanf(input, "%d %d", &x, &y);

                if (type_in >= 0 && type_in <= 2 && priority_in >= 0 && priority_in <= 2) {
                    create_new_incident((IncidentType)type_in, (Priority)priority_in, x, y);
                } else {
                    printf("Invalid input. Please try again.\n");
                }
                break;
            }
            case 2:
                printf("Continuing simulation...\n");
                break;
            case 3:
                printf("Exiting simulation...\n");
                log_simulation_end();
                exit(0);
            default:
                printf("Invalid choice. Please try again.\n");
        }
    }
}
```

تابع `process_user_input()`

این تابع **در ابتدای شبیه‌سازی و فقط در Turn 0 اجرا می‌شود** و وظیفه‌اش این است که یک **منوی انتخاب به کاربر نمایش دهد.**
کاربر می‌تواند از بین سه گزینه، یکی را انتخاب کند:

1. اضافه کردن حادثه جدید
2. ادامه شبیه‌سازی
3. خروج از برنامه

```c
void perform_turn_actions() {
    printf("-------------------- Turn Actions --------------------\n");

    // The order of operations in each turn is critical for correct logic.
    // 1. Update unit movements so they reach their destination.
    update_units_movement();

    // 2. Update incident statuses (e.g., start or finish operations) based on unit locations.
    update_incidents();

    // 3. Dispatch available units to new or ongoing incidents.
    dispatch_units();

    // 4. Render the final state of the map for this turn.
    update_and_render();
}
```

تابع `perform_turn_actions()`

این تابع در هر نوبت از شبیه‌سازی (به جز Turn 0) اجرا می‌شود و کارهایی مثل حرکت دادن واحدها، بررسی وضعیت حادثه‌ها، به‌روزرسانی وضعیت واحدها، رسم نقشه، و ثبت وقایع را انجام می‌دهد.

```
void update_and_render() {
    // ----------------------------------------------------------------------DONE: Determine the correct order of rendering the map.
// Call functions to clear the map, place departments, incidents, and units.
// Finally, call the function that renders the map to the console.



    // The rendering order is crucial for correct visual priority.
    // 1. Clear the map to have a blank slate.
    clear_map();

    // 2. Place units first. They have the lowest priority.
    place_units();

    // 3. Place departments. They will overwrite units if they are at the same location.
    place_departments();

    // 4. Place incidents last. They have the highest visual priority and will overwrite anything else.
    place_incidents();

    // 5. Finally, render the composed map to the console.
    render_map();

}
```

تابع `update_and_render()`

این تابع مسئول **تعیین ترتیب دقیق و درست نمایش نقشه** در کنسول است.
در این پروژه، روی نقشه باید سه نوع عنصر نمایش داده شوند:

- **واحدها (Units)**
- **ایستگاه‌ها (Departments)**
- **حوادث (Incidents)**

ترتیب رسم این عناصر بسیار مهم است، چون اگر دو عنصر در یک موقعیت قرار بگیرند، **عنصر بعدی، عنصر قبلی را می‌پوشاند**. بنابراین تابع `update_and_render()` این عناصر را **با اولویت مشخص و کنترل‌شده** رسم می‌کند.

```
void log_simulation_end() {
    if (!log_file) return;
    log_timestamp();
    fprintf(log_file, "Simulation Ended.\n");
    if (log_file) {
        fclose(log_file);
        log_file = NULL;
    }
}
```

تابع `log_simulation_end()`

این تابع معمولاً در پایان برنامه (قبل از `return 0`) فراخوانی می‌شود تا مشخص کند **شبیه‌سازی به پایان رسیده است**.
هدف آن، ثبت یک پیام متنی همراه با زمان در **فایل لاگ** است.

۷. نتیجه‌گیری

در این پروژه، یک سامانه شبیه‌سازی اعزام واحدهای اضطراری طراحی و پیاده‌سازی شد که قابلیت مدیریت حوادث و واحدهای امدادی را به صورت نوبتی و پویا فراهم می‌کند. این سامانه با استفاده از توابع مختلفی مانند بارگذاری پیکربندی، دریافت ورودی کاربر، به‌روزرسانی وضعیت واحدها و حوادث، و نمایش نقشه به صورت متنی، یک شبیه‌سازی ساده ولی کاربردی ارائه نمود.

استفاده از فایل‌های لاگ برای ثبت دقیق رخدادها و وضعیت سیستم باعث شد امکان بررسی عملکرد شبیه‌سازی در هر لحظه فراهم شود و به بهبود و رفع اشکالات کمک کند. همچنین طراحی تابعی مانند `update_and_render` که مسئول به‌روزرسانی و نمایش نقشه با اولویت‌بندی عناصر مختلف است، ساختار برنامه را ساده و قابل فهم‌تر کرد.

این پروژه نشان داد که با استفاده از زبان C و ساختارهای داده مناسب می‌توان سامانه‌های مدیریت حوادث را به صورت کارآمد و قابل توسعه طراحی کرد. در نهایت، این شبیه‌سازی می‌تواند به عنوان پایه‌ای برای پروژه‌های بزرگ‌تر در حوزه شهر هوشمند و مدیریت بحران مورد استفاده قرار گیرد.