NON-OOP EXAMPLE

This is a very NON-OOP example, to show you something of what the ancient C-style of programming looked like.

Note: **Structs** provide some encapsulation of data, but not of methods.

Note: all data is publicly visible.

Those are two of the major differences to be found, when the following is compared with modern OOP software.
/* explore.h */
#define FULL 100
#define EMPTY 0
#define SIP 10
#define NONE 0
#define LOADED 10
#define MAX_ROOMS 10

typedef int Flag;
typedef enum {N, S, E, W} Direction;
typedef enum {critical, fair, good, excellent} Health;

typedef struct Room {
    char name[100];
    struct Room *northExit;
    struct Room *southExit;
    struct Room *eastExit;
    struct Room *westExit;
    Flag monsterPresent;
    Flag foulAir;
    Flag goodWaterAvailable;
    Flag mushroomsAvailable;
} Room;
/* player.c */

#include <stdio.h>
#include "explore.h"
#include "cavern.h"
#include "player.h"

void printPlayerStatus(Player *player) {
    printf("player %s: %d, %d, %d, %d, %d, %d\n",
            player->name, player->hunger, 
            player->foodSupply, player->thirst, 
            player->waterSupply, player->health, 
            player->weapon);
}

void initializePlayer(Player *player, char *name, Room *room) {
    player->hunger      = NONE;
    player->foodSupply  = FULL;
    player->thirst      = NONE;
    player->waterSupply = FULL;
    player->health      = excellent;
    player->weapon      = FULL;
    strcpy(player->name, name);
    move(player, room);
    printPlayerStatus(player);
}
/* player.c (continued ...)*/

void move(Player *player, Room *room) {
    printf("move: moving %s to %s\n",
            player->name, room->name);
    player->currentRoom = room;
}

void drink(Player *player) {
    if (player->currentRoom->goodWaterAvailable) {
        player->thirst      = NONE;
        player->waterSupply = FULL;
    }
}
/ * cavern.c *

#include <stdio.h>
#include <stdlib.h>
#include "explore.h"
#include "cavern.h"

void initializeRoom(Room *room, char *name) {
    room->northExit = NULL;
    room->southExit = NULL;
    room->eastExit = NULL;
    room->westExit = NULL;
    strcpy(room->name, name);
}

char convertDirectionToChar(Direction d) {
    switch (d) {
    case N: return 'n';
    case S: return 's';
    case E: return 'e';
    case W: return 'w';
    default:
        printf("convertDirectionToChar: error\n");
        return 0;
    }
}
Direction convertCharToDirection(char c) {
    switch (c) {
        case 'n': return N;
        case 's': return S;
        case 'e': return E;
        case 'w': return W;
        default: printf("convertCharToDirection: error\n");
    }
}

Direction oppositeDirection(Direction d) {
    switch (d) {
        case N: return S;
        case S: return N;
        case E: return W;
        case W: return E;
        default: printf("oppositeDirection: error\n");
    }
}
Room *makeRoom(Room *oldRoom,
               Direction directionToNewRoom,
               char *name) {
    Room *newRoom = (Room*)malloc(sizeof(Room));
    if (newRoom == NULL) {
        printf("makeRoom: malloc failed, bye!\n");
        exit(0);
    }

    initializeRoom(newRoom, name);
    switch (directionToNewRoom) {
        case N:
            oldRoom->northExit = newRoom;
            newRoom->southExit = oldRoom;
            break;
        case S:
            oldRoom->southExit = newRoom;
            newRoom->northExit = oldRoom;
            break;
        case E:
            oldRoom->eastExit = newRoom;
            newRoom->westExit = oldRoom;
            break;
        case W:
            oldRoom->westExit = newRoom;
            newRoom->eastExit = oldRoom;
            break;
        default:
            printf("makeRoom: illegal direction\n");
    }
    return newRoom;
}
void initializeCavern(void) {
    int n;
    char name[100];

    initializeRoom(&entranceHall, "EntranceHall");
    rooms[0] = &entranceHall;
    for (n = 1; n < MAX_ROOMS; n++) {
        printf("initializing room %d\n", n);
        sprintf(name, "Room%d", n);
        rooms[n] = makeRoom(rooms[n - 1], E, name);
    }
}
void promptMove(void) {
    Room *currentRoom = tom.currentRoom;
    int n;
    char c;

    printf("Move player in which direction? (n, s, e, or w): ");
switch (c = readChar()) {
  case 'n':
    if (currentRoom->northExit != NULL)
      move(&tom, currentRoom->northExit);
    else
      printf("no north exit\n");
    break;
  case 's':
    if (currentRoom->southExit != NULL)
      move(&tom, currentRoom->southExit);
    else
      printf("no south exit\n");
    break;
  case 'e':
    if (currentRoom->eastExit != NULL)
      move(&tom, currentRoom->eastExit);
    else
      printf("no east exit\n");
    break;
  case 'w':
    if (currentRoom->westExit != NULL)
      move(&tom, currentRoom->westExit);
    else
      printf("no west exit\n");
    break;
  default:
    printf("promptMove: illegal input\n");
void printOptions(void) {
    printf("input:\n");
    printf("q to quit game\n");
    printf("m to move to another room then n, s, e, or w\n");
    printf("p to print player status\n");
    printf("? to print this message\n");
}

void main (void) {
    char c;
    initializeCavern();
    initializePlayer(&tom, "Tom Sawyer", &entranceHall);
    while (TRUE)
        switch (c = readChar()) {
            case 'q': exit(0);
            case 'm': promptMove(); break;
            case 'p': printPlayerStatus(&tom); break;
            case '?': printOptions(); break;
            default: printf("\nERROR: input either q, m, p or ?\n");
        }
}