Sprawozdanie – Saper

Grzegorz Swęd 18.01

1. Zasady gry

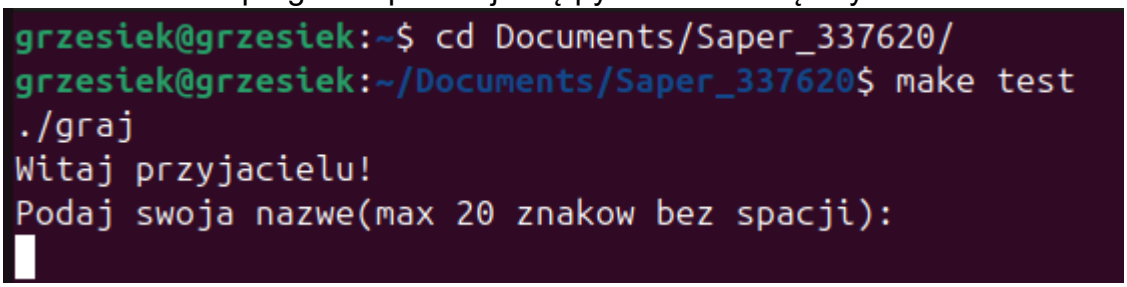
   Gra polega na odkrywaniu na planszy poszczególnych pól w taki sposób, aby nie natrafić na minę. Na każdym z odkrytych pól napisana jest liczba min, które bezpośrednio stykają się z danym polem (od jeden do ośmiu; jeśli min jest zero to na polu nie ma wpisanej liczby). Należy używać tych liczb by wydedukować gdzie schowane są miny. Jeśli oznaczymy dane pole flagą, jest ono zabezpieczone przed odsłonięciem, dzięki czemu nie odsłonimy miny przez przypadek.
   Źródło: https://gra-saper.pl/zasady

2. Sposób uruchomienia programu

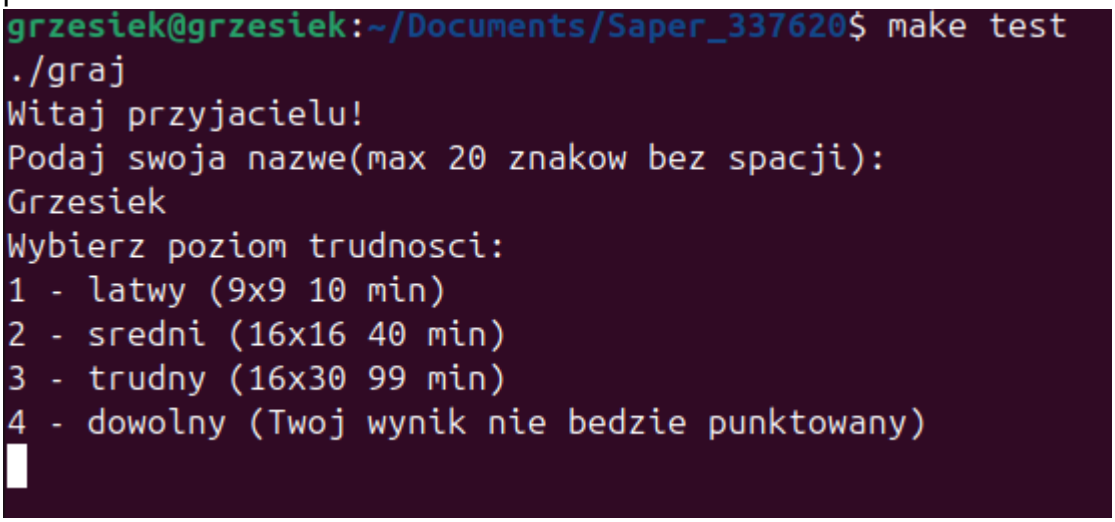
   Aby uruchomić program należy w terminalu przejść do katalogu Saper_337620, a następnie wpisać komendę „make test”.

3. Jak grać?

   Po uruchomieniu programu pokazuje się pytanie o nazwę użytkownika:



   Należy podać nazwę (można wpisać więcej niż 20 znaków, ale nie będą one przechowane), a następnie nacisnąć „enter”. Potem pojawi się menu wyboru poziomu trudności:



   Należy wpisać poziom i wcisnąć „enter”, jeśli wybierzemy poziom dowolny, musimy podać rozmiar planszy (wiersze x kolumny) i ilość min (ilość min nie może przekraczać połowy liczby pól), należy podać trzy liczby po spacjach:

```
Witaj przyjacielu!
Podaj swoja nazwe(max 20 znakow bez spacji):
Grzesiek
Wybierz poziom trudnosci:
1 - latwy (9x9 10 min)
2 - sredni (16x16 40 min)
3 - trudny (16x30 99 min)
4 - dowolny (Twoj wynik nie bedzie punktowany)
4
Podaj wymiary i ilosc min (uwaga ilosc min nie moze przekraczac polowy pol)
12 12 35
```

Potem zaczyna się gra, pojawia się plansza i można zacząć grać, aby wykonać ruch należy wpisać r(aby odsłonić pol) lub f(aby postawić flagę) oraz po spacji podać wiersz i kolumnę:

```
Witaj przyjacielu!
Podaj swoja nazwe(max 20 znakow bez spacji):
Grzesiek
Wybierz poziom trudnosci:
1 - latwy (9x9 10 min)
2 - sredni (16x16 40 min)
3 - trudny (16x30 99 min)
4 - dowolny (Twoj wynik nie bedzie punktowany)
1
Twoj wynik: 0
   1  2  3  4  5  6  7  8  9
1 [ ][ ][ ][ ][ ][ ][ ][ ]
2 [ ][ ][ ][ ][ ][ ][ ][ ]
3 [ ][ ][ ][ ][ ][ ][ ][ ]
4 [ ][ ][ ][ ][ ][ ][ ][ ]
5 [ ][ ][ ][ ][ ][ ][ ][ ]
6 [ ][ ][ ][ ][ ][ ][ ][ ]
7 [ ][ ][ ][ ][ ][ ][ ][ ]
8 [ ][ ][ ][ ][ ][ ][ ][ ]
9 [ ][ ][ ][ ][ ][ ][ ][ ]
Twoj pierwszy ruch:
r 1 1
```

```
Twoj wynik: 58
   1  2  3  4  5  6  7  8  9
1 [0][1][ ][1][0][0][0][0][0]
2 [0][1][ ][1][0][0][0][0][0]
3 [0][1][1][1][0][0][0][1][1]
4 [0][0][0][0][0][0][0][1][ ]
5 [0][0][0][0][0][0][1][2][ ]
6 [0][0][0][0][1][1][3][ ][ ]
7 [0][0][0][0][1][ ][ ][ ][ ]
8 [1][2][2][1][1][ ][ ][ ][ ]
9 [ ][ ][ ][ ][ ][ ][ ][ ][ ]
Twoj ruch:
```

Gra może skończyć się zwycięstwem lub porażką

```
Przegrales/as
Twoj wynik: 70
Grzesiek
Top 5 najlepszych wynikow:
1: Agnieszka 294
2: twojstary 184
3: ktos 130
4: wecvi 126
5: abcdefghijk 120
```

```
Wygrales/as
Twoj wynik: 71
Grzesiek
Top 5 najlepszych wynikow:
1: Agnieszka 294
2: twojstary 184
3: ktos 130
4: wecvi 126
5: abcdefghijk 120
```

Na koniec wyświetlane jest 5 najlepszych wyników, jeśli udało się osiągnąć taki wynik zostanie wyświetlony.

4. Podział na moduły

Program składa się z poszczególnych modułów:

```
grzesiek@grzesiek:~/Documents/Saper_337620$ ls
graj  gra.txt  main.c  main.o  Makefile  plansza.c  plansza.h  plansza.o  plik.c  plik.h  plik.o  wyniki.txt
```

5. Implementacja

Program na początku pyta się o nazwę, a następnie sprawdza jej poprawność

```
while(flag == 0){//sprawdzenie czy nazwa podana przez uzytkownika jest poprawna
    gets(nazwa);
    flag2 = 0;
    c = 0;
    while(nazwa[c] != 0){
        if(nazwa[c] == ' '){
            flag2 = 1;
        }
        c++;
    }
    if(flag2 == 0){
        flag = 1;
    }
    else{
        printf("Bledna nazwa\n");
    }
}
```

Następnie pyta i sprawdza poprawności poziomu trudności:

```
printf("Wybierz poziom trudnosci:\n");
printf("1 - latwy (9x9 10 min)\n");
printf("2 - sredni (16x16 40 min)\n");
printf("3 - trudny (16x30 99 min)\n");
printf("4 - dowolny (Twoj wynik nie bedzie punktowany)\n");
flag = 0;
while(flag == 0){//sprawdzenie czy poziom jest dobry
    flag3 = 0;
    while(flag3 == 0){
        gets(tryb);
        if(tryb[0] == 49 || tryb[0] == 50 || tryb[0] == 51 || tryb[0] == 52){
            poziom = tryb[0] - 48;
            flag3 = 1;
        }
        else{
            printf("Bledny poziom\n");
        }
    }
```

Następnie za pomocą switch case ustawia poziom trudności – odpowiednie wartości:

```
switch(poziom){//ustawienie poziomu trudnosci
    case 1:
        x = 9;
        y = 9;
        ilosc_min = 10;
        flag = 1;
        break;
    case 2:
        x = 16;
        y = 16;
        ilosc_min = 40;
        flag = 1;
        break;
    case 3:
        x = 16;
        y = 30;
        ilosc_min = 99;
        flag = 1;
        break;
    case 4:
        flag2 = 0;
        while(flag2 == 0){
```

Jeśli został wybrany poziom 4 to pyta i sprawdza wymiary ilość min (maksymalna wielkość planszy to 99x99, a ilość min nie może być większa niż połowa pól):

```
printf("Podaj wymiary i ilosc min (uwaga ilosc min nie moze przekraczac polowy pol)\n");
flag3 = 0;
while(flag3 == 0){//sprawdzenie czy wymiary sa dobre
    gets(rozmiar);
    c = 0;
    flag4 = 0;
    x = 0;
    y = 0;
    ilosc_min = 0;
    while(flag4 == 0){
        while(rozmiar[c] != ' ' && rozmiar[c] != 0){
            if((rozmiar[c]-48)>=0 && (rozmiar[c]-48)<10){
                x = x*10 + rozmiar[c]-48;
                c++;
            }
            else{
                c++;
                flag4 = 1;
            }
        }
        c++;
        while(rozmiar[c] != ' ' && rozmiar[c] != 0){
            if((rozmiar[c]-48)>=0 && (rozmiar[c]-48)<10){
                y = y*10 + rozmiar[c]-48;
                c++;
            }
            else{
                c++;
                flag4 = 1;
            }
        }
        c++;
```

```
        while(rozmiar[c] != ' ' && rozmiar[c] != 0){
            if((rozmiar[c]-48)>=0 && (rozmiar[c]-48)<10){
                y = y*10 + rozmiar[c]-48;
                c++;
            }
            else{
                c++;
                flag4 = 1;
            }
        }
        c++;
        while(rozmiar[c] != 0){
            if((rozmiar[c]-48)>=0 && (rozmiar[c]-48)<10){
                ilosc_min = ilosc_min*10 + rozmiar[c]-48;
                c++;
            }
            else{
                c++;
                flag4 = 1;
            }
        }
        if(flag4 == 0){
            flag4 = 2;
        }
    }
    if(flag4 == 2){
        flag3 = 1;
    }
    else{
        printf("Bledne dane\n");
    }
```

```
            }
            if(ilosc_min > x*y/2 || x<3 || y<3 || ilosc_min<1 || x>99 || y>99){
                printf("Nie mozna utworzyc takiej planszy\n");
            }
            else{
                poziom = 0;
                flag = 1;
                flag2 = 1;
            }
        }
        break;
    default:
        printf("Nieprawidowe wprowadzenie\n");
        break;
```

Jeśli wszystko zostało podane poprawnie przechodzi do generowania planszy, do tego wykorzystałem strukturę pole:

```
typedef struct pole{
    int czy_mina;//wartosc 0 jesli na polu nie ma miny, wartosc 1 jesli jest
    int czy_odkryte;//wartosc 0 jesli pole nie zostalo odkryte przez gracza
    int wartosc;//wartosc -1 jesli na polu jest mina lub ilosc min dookola
    int odkrycie;//wartosc -2 jesli wyswietlona ma byc flaga lub wartosc liczbowa min dookola
}pole_t;
```

```
pole_t **genruj_plansze(int x, int y){//funkcja tworzy pusta plansze o podanych wymiarach
    pole_t **plansza = malloc(sizeof(pole_t*)*y);
    for(int i = 0; i < y; i++){
        plansza[i] = malloc(sizeof(pole_t)*x);
        for(int j = 0; j < x; j++){
            plansza[i][j].czy_mina = 0;
            plansza[i][j].czy_odkryte = 0;
            plansza[i][j].wartosc = 0;
            plansza[i][j].odkrycie = 0;
        }
    }
    return plansza;
}
```

Następnie uruchamia funkcję graj, która odpowiada za obsługę gry:

```
pole_t **plansza = genruj_plansze(x,y);//generowanie planszy
graj(x,y,ilosc_min,poziom,nazwa,plansza);//gra
```

Funkcja graj prosi o podanie pierwszego ruchu, sprawdza jego poprawność:

```
printf("Twoj pierwszy ruch:\n");
flag3 = 0;
while(flag3 == 0){
    gets(wpis);
    flag4 = 0;
    while(flag4 == 0){
        c = 2;
        if(wpis[0] == 'r'){
            if(wpis[1] == ' '){
                while(wpis[c] != ' ' && wpis[c] != 0){
                    if((wpis[c]-48)>=0 && (wpis[c]-48)<10){
                        c++;
                    }
                    else{
                        c++;
                        flag4 = 1;
                    }
                }
                c++;
                while(wpis[c] != 0){
                    if((wpis[c]-48)>=0 && (wpis[c]-48)<10){
                        wybrane_y = wybrane_y*10 + (wpis[c]-48);
                        c++;
                    }
                    else{
                        c++;
                        flag4 = 1;
                    }
                }
            }

                flag4 = 1;
            }
        }
        else{
            flag4 = 1;
INSERT --
```

```
                }
            }
            else{
                flag4 = 1;
            }
        }
        else{
            flag4 = 1;
        }
        if(flag4 == 0){
            flag4 = 2;
        }
    }
    if(flag4 == 2 && wybrane_x-1 < x && wybrane_y-1 < y){
        flag3 =1;
    }
    else{
        printf("Bledny ruch\n");
    }
```

Następnie na jego podstawie generuje losowy układ min, ale tak, żeby na podanym polu jej nie było:

```
generuj_miny(x,y,wybrane_x-1,wybrane_y-1,ilosc,plansza);
ustaw_wartosci(x,y,plansza);
```

```
pole_t **generuj_miny(int x, int y, int x_czyste, int y_czyste, int ilosc, pole_t **plansza){//funkcja losuje ustawienie min na planszy, nie moze ustaw
c miny na polu wybranym przez gracza jako pierwszym
    int wygenerowane = 0;
    srand(time(NULL));
    while(wygenerowane < ilosc){
            int x_los = rand() % x;
            int y_los = rand() % y;
        if(plansza[y_los][x_los].czy_mina == 0 && x_los != x_czyste && y_los != y_czyste){
            plansza[y_los][x_los].czy_mina = 1;
            wygenerowane++;
        }
    }
    return plansza;
}
```

Następnie ustawia wartości pól na ilość sąsiadujących min:

```
pole_t **ustaw_wartosci(int x, int y, pole_t **plansza){//funkcja na podstawie wygenerowanych min ustawia wartosi pol na ilosc sasiadujacych min
    for(int i = 0; i < y; i++){
        for(int j = 0; j < x; j++){
            if(plansza[i][j].czy_mina == 1){
                plansza[i][j].wartosc = -1;
            }
            else{
                int licznik = 0;
                for(int k = -1; k <= 1; k++){
                    for(int l = -1; l <= 1; l++){
                        if(i+k >= 0 && i+k < y && j+l >= 0 && j+l < x){
                            if(plansza[i+k][j+l].czy_mina == 1){
                                licznik++;
                            }
                        }
                    }
                }
                plansza[i][j].wartosc = licznik;
            }
        }
    }
    return plansza;
}
```

Następnie wykonuje ruch:

```c
int dodaj = 0;
if(plansza[wybrane_x-1][wybrane_y-1].wartosc == 0){
    odkrywanie(x,y,&dodaj,wybrane_x-1,wybrane_y-1,plansza);
    odkryte+=dodaj;
}
else{
    plansza[wybrane_x-1][wybrane_y-1].czy_odkryte = 1;
    odkryte++;
    plansza[wybrane_x-1][wybrane_y-1].odkrycie = plansza[wybrane_x-1][wybrane_y-1].wartosc;
}
pisz_plansze(x,y,odkryte*poziom,plansza);
if(odkryte == x*y-ilosc){
    printf("Wygrales/as\n");
}
```

Funkcja odkrywanie aktywuje się, jeśli odkrywane jest pole o wartości 0:

```c
void odkrywanie(int x, int y, int* odkrycia, int x_podane, int y_podane, pole_t **plansza){//funkcja odkrywa wszystkie pola nie sasiadujace z mina i sas
iadujace do nich
    if(plansza[x_podane][y_podane].czy_odkryte == 0){
        plansza[x_podane][y_podane].czy_odkryte = 1;
        plansza[x_podane][y_podane].odkrycie = plansza[x_podane][y_podane].wartosc;
        (*odkrycia)++;
    }
    for(int k = -1; k <= 1; k++){
        for(int l = -1; l <= 1; l++){
            if(x_podane+k >= 0 && x_podane+k < x && y_podane+l >= 0 && y_podane+l < y && plansza[x_podane+k][y_podane+l].czy_odkryte == 0){
                if(plansza[x_podane+k][y_podane+l].wartosc == 0){
                    odkrywanie(x,y,odkrycia,x_podane+k,y_podane+l,plansza);
                }
                else{
                    if(plansza[x_podane+k][y_podane+l].czy_odkryte == 0){
                        plansza[x_podane+k][y_podane+l].czy_odkryte = 1;
                        plansza[x_podane+k][y_podane+l].odkrycie = plansza[x_podane+k][y_podane+l].wartosc;
                        (*odkrycia)++;
                    }
                }
            }
        }
    }
}
```

Odkrywa wszystkie pola o wartości 0 połączone takimi polami z odkrywanym polem, oraz sąsiadujące z nimi pola. Zwraca ilość odkrytych w ten sposób pól.
Funkcja graj na bieżąco zlicza ilość odkrytych pól.
Następnie uruchamia się pętla, która trwa, aż gra się nie skończy i wykonuje następujące czynności:
-pisze aktualny stan planszy

```c
void pisz_plansze(int x, int y, int wynik, pole_t **plansza){//funkcja wypisuje aktualny stan planszy
    printf("Twoj wynik: %d \n  ", wynik);
    for(int k = 0; k < x; k++){
        if(k<9){
            printf(" %d ",k+1);
        }
        else{
            printf("%d ",k+1);
        }
    }
    printf("\n");
    for(int i = 0; i < y; i++){
        if(i>8){
            printf("%d",i+1);
        }
        else{
            printf("%d ",i+1);
        }
        for(int j = 0; j < x; j++){
            if(plansza[i][j].czy_odkryte == 0){
                printf("[ ]");
            }
            else if(plansza[i][j].odkrycie == -2){
                printf("[f]");
            }
            else{
                printf("[%d]",plansza[i][j].odkrycie);
            }
        }
        printf("\n");
    }
}
```

-pyta i sprawdza poprawność ruchu

```
   while(flag2==0){
       printf("Twoj ruch:\n");
       flag3 = 0;
       while(flag3 == 0){
           gets(wpis1);
           flag4 = 0;
           wybrane_x = 0;
           wybrane_y = 0;
           while(flag4 == 0){
               c = 2;
               if(wpis1[0] == 'r' || wpis1[0] == 'f'){
                   rodzaj = wpis1[0];
                   if(wpis1[1] == ' '){
                       while(wpis1[c] != ' ' && wpis1[c] != 0){
                           if((wpis1[c]-48)>=0 && (wpis1[c]-48)<10){
                               wybrane_x = wybrane_x*10 + (wpis1[c]-48);
                               c++;
                           }
                           else{
                               c++;
                               flag4 = 1;
                           }
                       }
                       c++;
                       while(wpis1[c] != 0){
                           if((wpis1[c]-48)>=0 && (wpis1[c]-48)<10){
                               wybrane_y = wybrane_y*10 + (wpis1[c]-48);
                               c++;
                               }
                           else{
                               c++;
                               flag4 = 1;
                           }
                       }
                   }
                   else{
SERT --
```

```
                }
            }
            else{
                flag4 = 1;
            }
        }
        else{
            flag4 = 1;
        }
        if(flag4 == 0){
            flag4 = 2;
        }
    }
    if(flag4 == 2 && wybrane_x-1 < x && wybrane_y-1 < y){
        flag3 =1;
    }
    else{
        printf("Bledny ruch\n");
    }
```

-wykonuje ruch

```
        if(wybrane_x >= 0 && wybrane_x <= y && wybrane_y >= 0 && wybrane_y <= x){
            if(rodzaj == 'r' && (plansza[wybrane_x-1][wybrane_y-1].czy_odkryte == 0 || (plansza[wybrane_x-1][wybrane_y-1].czy_odkryte == 1 && plansz
a[wybrane_x-1][wybrane_y-1].odkrycie == -2 ))){
                if(plansza[wybrane_x-1][wybrane_y-1].wartosc != -1){
                    if(plansza[wybrane_x-1][wybrane_y-1].wartosc != 0){
                        plansza[wybrane_x-1][wybrane_y-1].czy_odkryte = 1;
                        odkryte++;
                        plansza[wybrane_x-1][wybrane_y-1].odkrycie = plansza[wybrane_x-1][wybrane_y-1].wartosc;
                        flag2=1;
                        printf("\033[2J");
                    }
                    else{
                        dodaj = 0;
                        odkrywanie(x,y,&dodaj,wybrane_x-1,wybrane_y-1,plansza);
                        odkryte+=dodaj;
                        flag2=1;
                        printf("\033[2J");
                    }
                }
                else{
                    printf("\033[2J");
                    printf("Przegrales/as\nTwoj wynik: %d\n",odkryte*poziom);
                    wynik_t rezultat = {nazwa,odkryte*poziom};
                    wpisanie_do_pliku(rezultat);
                    flag2=1;
                    flag=1;
                }
            }
            else if(rodzaj == 'f'){
                if(plansza[wybrane_x-1][wybrane_y-1].czy_odkryte == 0){
                    plansza[wybrane_x-1][wybrane_y-1].czy_odkryte = 1;
                    plansza[wybrane_x-1][wybrane_y-1].odkrycie = -2;
```

```
                printf("\033[2J");
                printf("Przegrales/as\nTwoj wynik: %d\n",odkryte*poziom);
                wynik_t rezultat = {nazwa,odkryte*poziom};
                wpisanie_do_pliku(rezultat);
                flag2=1;
                flag=1;
            }
        }
        else if(rodzaj == 'f'){
            if(plansza[wybrane_x-1][wybrane_y-1].czy_odkryte == 0){
                plansza[wybrane_x-1][wybrane_y-1].czy_odkryte = 1;
                plansza[wybrane_x-1][wybrane_y-1].odkrycie = -2;
                flag2=1;
                printf("\033[2J");
            }
            else{
                if(plansza[wybrane_x-1][wybrane_y-1].odkrycie == -2){
                    plansza[wybrane_x-1][wybrane_y-1].czy_odkryte = 0;
                    plansza[wybrane_x-1][wybrane_y-1].odkrycie = 0;
                    flag2=1;
                    printf("\033[2J");
                }
            }
        }
        else{
            printf("Nie mozesz wykonac tego ruchu\n");
        }
    }
    else{
        printf("Bledny ruch\n");
    }
}
```

- w przypadku przegranej kończy grę i wyświetla wynik analogicznie jak w przypadku wygranej(opis zaraz)
-sprawdza czy gra została wygrana i jeśli tak to generuje strukturę wynik i wywołuje funkcję wpisania wyniku do pliku i wyświetlenie 5 najlepszych graczy

```
 if(odkryte == x*y-ilosc){
     flag = 1;
     printf("\033[2J");
     printf("Wygrales/as\nTwoj wynik: %d\n",odkryte*poziom);
     wynik_t rezultat = {nazwa,odkryte*poziom};
     wpisanie_do_pliku(rezultat);

 }
```

```
typedef struct wynik{
    char* nazwa;//nazwa gracza
    int punkty;//punkty gracza
}wynik_t;
```

Funkcja wpisania pobiera wiersz po wierszu plik i wpisuje go do tablicy

```
void wpisanie_do_pliku(wynik_t rezultat){//funkcja czyta
iajac wlasnie osiagniety), na koniec dopisuje ostatni wyn
    FILE *f;
    f = fopen("wyniki.txt","r+");
    if(f == NULL){
        printf("Blad\n");
    }
    char wiersz[BUFSIZE];
    wynik_t wynik_tmp;
    wynik_t* wyniki = malloc(sizeof(wynik_t));
    int c = 0;
    while(fgets( wiersz, BUFSIZE, f ) != NULL){
        int c1 = 0;
        int wynik = 0;
        char* nazwa = malloc(sizeof(char)*20);
        while(nazwa[c1] != 0){
            nazwa[c1] = 0;
            c1++;
        }
        c1 = 0;
        while(wiersz[c1] != ' '){
            nazwa[c1] = wiersz[c1];
            c1++;
        }
        c1++;
        while(wiersz[c1] != '\n'){
            wynik = wynik * 10 + (wiersz[c1]-48);
            c1++;
        }
        wynik_tmp.nazwa = nazwa;
        wynik_tmp.punkty = wynik;
        wyniki = realloc(wyniki,sizeof(wynik_t)*(c+2));
        wyniki[c] = wynik_tmp;
        c++;
    }
```

Potem dopisuje ostatni wynik, sortuje tablice, wypisuje top 5 i dopisuje ostatni wynik do pliku

```
wyniki = realloc(wyniki,sizeof(wynik_t)*c);
wyniki[c] = rezultat;
qsort(wyniki, c+1 , sizeof(wynik_t),compare);
printf("Top 5 najlepszych wynikow:\n");
for(int i = 0; i < 5; i++){
    printf("%d: %s %d\n",i+1, wyniki[i].nazwa, wyniki[i].punkty);
}
fprintf(f,"%s %d", rezultat.nazwa, rezultat.punkty);
fprintf(f,"\n");
fclose(f);
```

6. Testy
   Wszystkie testy robiłem na bieżąco, wywoływaniem programu
7. Podział pracy
   Wszystko robiłem sam
8. Wnioski
   Nie udało mi się zaimplementować wszystkich funkcjonalności, nie starczyło mi nie tyle umiejętności co czasu. Samo napisanie saper, było dla mnie dosyć proste, największy kłopot sprawiła mi obsługa błędów wejścia z linii poleceń. Ostatecznie zdecydowałem się na użycie nie najlepszej funkcji gets, ale tylko ona pozwoliła mi na dokładną obsługę błędów.