1.Opis projektu oraz zastosowanie

Projekt jest programem, który służy nam do przeglądania katalogów, poruszania się w nich, wyszukiwania plików w różnych formach oraz tworzenia i usuwania plików. Pliki przechowywać będziemy w tablicy, których miejsce będziemy wybierać poprzez hashowanie. Rozwiązaniem kolizji w przypadku tablicy plików będzie metoda próbkowania liniowego tzn. jeżeli miejsce, w którym chcemy zamieścić nowe dane do tablicy będzie zajęte będziemy przeglądać tablicę, aż trafimy na dostępne miejsce. Jeżeli takiego nie ma program wyświetla stosowny komunikat o braku miejsca na dane. Katalogi przechowujemy w osobnej tablicy. W tym przypadku metodą rozwiązywania kolizji będzie metoda łańcuchowo tzn. jeżeli miejsce w tabeli jest zajęte tworzymy w tym miejscu listę jednokierunkową. Ta metoda gwarantuje nam, że zawsze możemy dodać następny element, więc w tym wypadku tablica będzie miała stosunkowo niską ilość elementów(list), ale te listy mogą przechowywać bardzo wiele elementów. Program posiada dwie opcje wyświetlania oraz pobierania danych do tabel. Pierwsza opcja to wyświetlanie tylko obecnego folderu oraz zaznaczanie jakie zawiera katalogi. W opcji drugiej program wyświetli i pobierze wszystkie pliki oraz katalogi z obecnego katalogu oraz podkatalogów tworząc odpowiednie zagnieżdżenia, aby można było czytelnie przeglądać wartości.

Struktura plików:

```
typedef struct filedata{
    char* fullname;
    char* filename;
    char* filetype;
    int depth;
}FILEDATA;
```

Struktura katalogów:

```
typedef struct catalog{
    char* name;
    int depth;
    struct catalog* next;
}CATALOG;
```

Tablice obu struktur są tablicami globalnymi, które inicjujemy w specjalnej funkcji.

```
void init()
{
    arr = (FILEDATA*)malloc(sizeof(FILEDATA) * MAXFILES);
    catarr = (CATALOG*)malloc(sizeof(CATALOG) * MAXCAT);
    int i;
    for(i = 0; i < MAXFILES;i++)
    {
        arr[i].fullname = NULL;
        arr[i].filename = NULL;
        arr[i].filetype = NULL;
        arr[i].depth = 0;
    }
    for(i = 0; i < MAXCAT;i++)
    {
        catarr[i].name = NULL;
        catarr[i].depth = 0;
        catarr[i].next = (CATALOG*)NULL;
    }
}
```

2.Opis algorytmów

Teraz przedstawię kilka z zastosowanych algorytmów.

- **Algorytm odczytywania plików w obecnej lokalizacji**

```
void printdir(char* name,int depth)
{
    init();
    DIR *d;
    struct dirent *dir;
    d = opendir(name);
    if (d)
    {
        while ((dir = readdir(d)) != NULL)
        {
            if(strcmp(dir->d_name,".") && strcmp(dir->d_name,".."))
            {
                char* path = (char*)malloc(sizeof(char)*(strlen(name)+1+strlen(dir->d_name)));
                strcpy(path,name);
                strcat(path,"/");
                strcat(path,dir->d_name);
                if(is_regular_file(path))
                {
                    printf("%*s", depth, "");
                    printf("%s\n", dir->d_name);
                    addtotab(dir->d_name,depth);
                }
                else
                {
                    printf("%*s", depth, "");
                    printf("%s (KATALOG)\n", dir->d_name);
                    addtocatarr(dir->d_name, depth);
                }
            }
        }
        closedir(d);
    }
}
```

W algorytmie tym korzystamy z biblioteki dirent, która posiada takie funkcje jak otwieranie katalogu, odczytywanie wartości, które może nam udostępnić takie dane jak nazwę wpisu oraz numer serii. Pliki odczytujemy w pętli while dopóki nie trafimy na pusty wskaźnik. Funkcja readdir pobiera również skróty do bieżącego folderu jakim jest kropka oraz do folderu poprzedzającego jakim są dwie kropki, tych danych nie potrzebujemy w tabeli, więc tworzymy zabezpieczenie logiczne AND w funkcji if(). Następnie tworzymy zmienną pomocniczą path, w której tworzymy ścieżkę do obecnie sprawdzanego elementu. Jeżeli funkcja ta zwróci wartość różną od 0 to plik nie jest katalogiem i wtedy dodajemy go do tablicy plików. W przeciwnym wypadku element umieszczamy w tablicy katalogów. Do obu funkcji przesyłamy również wartość depth jakim jest głębokość zagnieżdżenia folderu.

- **Algorytm dodawania do tablicy plików**

```
void addtotab(char* name, int depth)
{
    int len = strlen(name);
    int ftypelen = 0,fnamelen;
    int sum = 0;
    int dotindex;
    int i;
    for(i = len - 1;i>=0;i--)
    {
        if(name[i] == '.')
        {
            sum+=name[i];
            dotindex = i;
            break;
        }
        else
        {
            ftypelen++;
            sum+=name[i];
        }
    }
    if(ftypelen == len)
    {
        insert(name,name,"none",0,sum % MAXFILES);
        return;
    }
    i++;
    fnamelen = dotindex;
    char* filetype = (char*)malloc(sizeof(char)*ftypelen);
    char* filename = (char*)malloc(sizeof(char)*fnamelen);
    memcpy(filename,name,fnamelen);
    memcpy(filetype,name+dotindex+1,ftypelen);
    filetype[ftypelen] = '\0';
    filename[fnamelen] = '\0';
    int index = sum % MAXFILES;
    insert(name, filename, filetype, depth, index);
}
```

Wstępnym algorytmem jest tutaj algorytm wydzielania nazwy pliku na nazwę i rozszerzenie(o ile takie istnieje). Również zauważamy tutaj haszowanie wartośći. Sumujemy wszystkie znaki pliku, a następnie wykonujemy działanie modulo na ilość miejsc w tabeli i te dane przekazujemy bezpośrednio do funkcji wstawiającej insert. Funkcja insert sprawdza czy miejsce o wartości funkcji haszującej w tabeli jest zajęte. Jeżeli nie jest to umieszcza w nim dane. Jeżeli miejsce jest zajęte funkcja szuka dopóki nie znajdzie wolnego miejsca lub nie przeglądnie wszystkich lokalizacji. Jeżeli znajdzie wolne miejsce to umieszcza w nim element jeżeli nie ma już wolnego miejsca to funkcja wyświetla komunikat o braku miejsca na dane.

```
void insert(char* fullname,char* filename, char* filetype, int depth, int loc)
{
    if(arr[loc].filename == (char*)NULL)
    {
        arr[loc].fullname = (char*)malloc(sizeof(char)*strlen(fullname));
        arr[loc].filename = (char*)malloc(sizeof(char)*strlen(filename));
        arr[loc].filetype = (char*)malloc(sizeof(char)*strlen(filetype));
        strcpy(arr[loc].fullname, fullname);
        strcpy(arr[loc].filename, filename);
        strcpy(arr[loc].filetype, filetype);
        arr[loc].depth = depth;
        return;
    }
    else
    {
        int startloc = loc++;
        while(arr[loc].filename != (char*)NULL)
        {
            loc++;
            if(loc == MAXFILES)
                loc = 0;
            if(loc == startloc)
            {
                printf("BRAK MIEJSCA NA WIEKSZA ILOSC DANYCH\n");
                return;
            }
        }
        arr[loc].fullname = (char*)malloc(sizeof(char)*strlen(fullname));
        arr[loc].filename = (char*)malloc(sizeof(char)*strlen(filename));
        arr[loc].filetype = (char*)malloc(sizeof(char)*strlen(filetype));
        strcpy(arr[loc].fullname, fullname);
        strcpy(arr[loc].filename, filename);
        strcpy(arr[loc].filetype, filetype);
        arr[loc].depth = depth;
    }
}
```

- **Algorytm dodawania katalogów do tablicy**

```
void addtocatarr(char* name, int depth)
{

    int i,loc;
    for(i = 0; i < strlen(name); i++)
        loc += name[i];
    loc %= MAXCAT;
    if(catarr[loc].name == (char*)NULL)
    {

        catarr[loc].name = (char*)malloc(sizeof(char)*strlen(name));
        strcpy(catarr[loc].name, name);
        catarr[loc].depth = depth;
        catarr[loc].next = (CATALOG*)NULL;
        return;
    }
    else
    {

        CATALOG* temp;
        temp = catarr+loc;
        while(temp->next != (CATALOG*)NULL)
        {
            temp = temp->next;
        }
        CATALOG* next = (CATALOG*)malloc(sizeof(CATALOG));
        next->name = (char*)malloc(sizeof(char)*strlen(name));
        strcpy(next->name, name);
        next->depth = depth;
        next->next = (CATALOG*)NULL;
        temp->next = next;
    }
}
```

Tutaj wartość haszowania obliczamy podobnie jak w przypadku plików. Jeżeli komórka jest pusta to dodajemy w niej nasz element. Jeżeli dane miejsce jest zajęte to przeszukujemy listę w tej lokalizacji, aż znajdziemy następne miejsce wolne. Gdy znajdziemy wolne miejsce dołączamy w jego lokalizację ten element.

- Algorytm otwierania folderu

```
void openfolder(char* name)
{
    if (searchforfolder(name) == 1)
    {
        char* temp = (char*)malloc(sizeof(char)*strlen(currentloc));
        char* temp2 = (char*)malloc(sizeof(char)*strlen(currentloc));
        strncpy(temp, currentloc,strlen(currentloc));
        strncpy(temp2, currentloc,strlen(currentloc));
        temp[strlen(temp)] = '/';
        strcpy(currentloc, temp);
        strcat(currentloc, name);
        if(is_regular_file(temp))
        {
            strcpy(currentloc, temp2);
            printf("\nNie ma folderu %s !\n\n", name);
        }
        else
        {
            printf("\nWchodze do folderu %s !\n\n", name);
            chdir(name);
        }

    }
    else
        printf("\nNie ma folderu %s !\n\n", name);
}
```

Funkcja najpierw sprawdza czy istnieje folder o takiej nazwie. Tworzymy odpowiednio nową ścieżkę, w której będziemy się znajdować. Następnie dla pewności sprawdzamy czy ścieżka ta wskazuje na folder(folder może być w tabeli, ale jest bardziej zagnieżdżony wtedy należy to uwzględnić wykonując odpowiednie wywołanie). Jeżeli wystąpił błąd znalezienia folderu w danej ścieżce to zostajemy w bieżącej lokalizacji. Jeżeli błędu nie ma przeskakujemy poleceniem chdir() do nowej lokalizacji.

- Algorytm szukania folderu

```
int searchforfolder(char* name)
{
    int i;
    for(i = 0; i < MAXCAT; i++)
    {
        if(catarr[i].name != (char*)NULL)
        {
            if(strcmp(catarr[i].name, name) == 0)
                return 1;
            CATALOG* temp = catarr+i;
            while(temp->next != (CATALOG*)NULL)
            {
                temp = temp->next;
                if(strcmp(temp->name, name) == 0)
                    return 1;
            }
        }
    }
    return 0;
}
```

Algorytm ten przegląda tablicę wszystkich list zawierających katalogi. Gdy nazwa katalogu pokryje się z nazwą przekazaną do funkcji algorytm zwraca 1 dla potwierdzenia, że taki

katalog istnieje obecnym folderze lub jednym spod folderów. Jeżeli zwrócone zostaje 0 takiego folderu nie ma.

- Algorytm szukania pliku

```
int searchforfile(char* fname)
{
    int i;
    for(i = 0; i < MAXFILES; i++)
    {
        if(arr[i].fullname != (char*)NULL && strcmp(arr[i].fullname, fname) == 0)
        {
            return 1;
        }
    }
    return 0;
}
```

W tym przypadku poszukiwania przeglądamy tablicę plików. Sprawdzamy czy dana pozycja nie jest pusta i jeżeli nie jest pusta to sprawdzamy czym nazwa pliku pokrywa się z tą która trafiła do metody. Jeżeli nazwy się pokrywają to zwraca 1 dla potwierdzenia, że taki plik istnieje w naszym folderze lub jednym spod folderów. Jeżeli zwrócone zostaje 0 takiego pliku nie ma.

- Algorytm tworzenia nowego pliku

```
int addfile(char* name)
{
    int check = searchforfile(name);
    int i = 1;
    while(check != 0)
    {
        char c[10];
        itoa(i, c, 10);
        char* temp = (char*)malloc(sizeof(char) * (strlen(name)+strlen(c)));
        strcat(temp, c);
        strcat(temp, name);
        check = searchforfile(temp);
        if(check == 0)
        {
            strcpy(name, temp);
        }
        else
        {
            memset(c,0,strlen(c));
            i++;
        }
    }
    FILE* fn = fopen(name, "w");
    if(fn == NULL)
    {
        return 0;
    }
    fclose(fn);
    addtotab(name, 0);
    return 1;
}
```

Algorytm ten sprawdza czy nie istnieje w obecnej lokalizacji plik o takie samej nazwie jak ten który chcemy utworzyć. Jeżeli istnieje to na początku nazwy pliku dodaje cyfrę. Następnie nazwę przekazuje do funkcji, która doda plik do tabeli plików.

Całość funkcjonowania programu operuje funkcja menu(), która przedstawia nam obecną lokalizację w której się znajdujemy. Jest ona zapętlona za pomocą instrukcji switch case. Dodatkowo program informuje użytkownika jeżeli wprowadzi niepoprawną wartość.

```
void menu()
{
    printf("\n\nYour current location: %s \n\n", currentloc);
    printf("-----------MENU(enter number between 0 and 9)-----------\n");
    printf("0.GO up\n");
    printf("1.Read current dir\n");
    printf("2.Open folder by name\n");
    printf("3.Search for file by fullname\n");
    printf("4.Search for file by starting part\n");
    printf("5.Search for file by type\n");
    printf("6.Create file\n");
    printf("7.Remove file by name\n");
    printf("8.Read current dir with subdirs\n");
    printf("9.Exit\n");
    int i;
    scanf("%d",&i);
    switch(i)
    {
        case 0:
            goup();
            menu();
            break;
        case 1:
            {
                printdir(currentloc, 0);
                menu();
                break;
            }
```

3.Ilustracja działania programu

Przy uruchomieniu programu dostajemy wyświetlenie obecnego folderu oraz jego lokalizację.

```
D:\STUDIA\AiSD\oraganizerkatalogow-hash\organizer.exe
nowy.jpg
nowyp.txt
nowyplik.txt
organizer.dev
organizer.exe
organizer.layout
plik.txt
pliktestowy.txt
Project3.dev
Project3.layout
test (KATALOG)
test2.txt
~$kumentacja_organiezer_ks.docx
~WRL1206.tmp


Your current location: .

------------MENU(enter number between 0 and 9)------------
0.GO up
1.Read current dir
2.Open folder by name
3.Search for file by fullname
4.Search for file by starting part
5.Search for file by type
6.Create file
7.Remove file by name
8.Read current dir with subdirs
9.Exit
```

Opcją 0 przechodzimy do folderu poprzedzającego folder w hierarchii.

```
0

Wychodze folder wyzej!



Your current location: ../.

------------MENU(enter number between 0 and 9)------------
0.GO up
1.Read current dir
2.Open folder by name
3.Search for file by fullname
4.Search for file by starting part
5.Search for file by type
6.Create file
7.Remove file by name
8.Read current dir with subdirs
9.Exit
```

Opcją 8 wyświetlimy pliki wraz z zagnieżdżeniami.

```
8
AiSD_funkcje do dyskusji.c
calka.c
calka.exe
Katalog codeblock:
    Katalog sort:
        algorytmy.c
        algorytmy.h
        Katalog bin:
            Katalog Debug:
                sort.exe
        danedowykresu_posortowane_zestaw2.txt
        main.c
        Katalog obj:
            Katalog Debug:
                algorytmy.o
                main.o
        sort.cbp
        sort.depend
        sort.layout
dokumentacja_calki_ks.docx
dokumentacja_calki_ks.pdf
Katalog kartkowka:
    kartkowka.dev
    kartkowka.exe
    kartkowka.layout
    main.c
    main.o
    Makefile.win
Katalog oraganizerkatalogow-hash:
    1nowy.jpg
    2nowy.jpg
    dokumentacja_organiezer_ks.docx
    main.c
    main.o
    Makefile.win
    nowy.jpg
    nowyp.txt
    nowyplik.txt
    organizer.dev
    organizer.exe
    organizer.layout
    plik.txt
    pliktestowy.txt
    Project3.dev
    Project3.layout
    Katalog test:
        DANE1.txt
        DANE2.txt
        DANEout.txt
        Katalog foldertestowy:
```

Opcją 1 odczytamy folder bez zagnieżdżeń.

```
1
AiSD_funkcje do dyskusji.c
calka.c
calka.exe
codeblock (KATALOG)
dokumentacja_calki_ks.docx
dokumentacja_calki_ks.pdf
kartkowka (KATALOG)
oraganizerkatalogow-hash (KATALOG)
prk (KATALOG)
rekurencja.c
rekurencja.exe
sortowanie (KATALOG)
stolarzkonrad.c


Your current location: ../.

------------MENU(enter number between 0 and 9)------------
0.GO up
1.Read current dir
2.Open folder by name
3.Search for file by fullname
4.Search for file by starting part
5.Search for file by type
6.Create file
7.Remove file by name
8.Read current dir with subdirs
9.Exit
```

Opcją 2 program spyta nas o nazwę folderu do otworzenia oraz następnie wyświetli ten folder bez zagnieżdżeń.

```
2
Enter name of folder: sortowanie

Wchodze do folderu sortowanie !

algorytmy.c
algorytmy.h
algorytmy.o
Bubblediagram.png
bubblewrongdiag.png
DANE.txt
danelosowe.txt
daneodwrotnie.txt
daneposortowane.txt
dokumentacja_sortowanie_ks.docx
dokumentacja_sortowanie_ks.pdf
heapsortdiagram.png
insertdiagram.png
main.c
main.o
Makefile.win
quicksortdiagram.png
razem-sort-pliki.pdf
selectdiagram.png
shelldiagram.png
Sortowanie.dev
Sortowanie.exe
Sortowanie.layout
sortowanie_ST.pdf
stack.bat
stolarzkonrad.tar
wykreslosowedane (KATALOG)
Wykresy.xlsx


Your current location: ../sortowanie

------------MENU(enter number between 0 and 9)------------
0.GO up
1.Read current dir
2.Open folder by name
3.Search for file by fullname
4.Search for file by starting part
5.Search for file by type
6.Create file
7.Remove file by name
8.Read current dir with subdirs
9.Exit
```

Opcją 3 program spyta o nazwę pliku, który szukamy i sprawdzi czy taki istnieje.

```
D:\STUDIA\AiSD\oraganizerkatalogow-hash\organizer.exe
algorytmy.h
algorytmy.o
Bubblediagram.png
bubblewrongdiag.png
DANE.txt
danelosowe.txt
daneodwrotnie.txt
daneposortowane.txt
dokumentacja_sortowanie_ks.docx
dokumentacja_sortowanie_ks.pdf
heapsortdiagram.png
insertdiagram.png
main.c
main.o
Makefile.win
quicksortdiagram.png
razem-sort-pliki.pdf
selectdiagram.png
shelldiagram.png
Sortowanie.dev
Sortowanie.exe
Sortowanie.layout
sortowanie_ST.pdf
stack.bat
stolarzkonrad.tar
wykreslosowedane (KATALOG)
Wykresy.xlsx


Your current location: ../sortowanie

------------MENU(enter number between 0 and 9)------------
0.GO up
1.Read current dir
2.Open folder by name
3.Search for file by fullname
4.Search for file by starting part
5.Search for file by type
6.Create file
7.Remove file by name
8.Read current dir with subdirs
9.Exit
3
Enter name of file you are looking for: insertdiagram.png
Plik insertdiagram.png istnieje
```

Opcją 4 możemy np. wyszukać początek pliku. Program również wyświetli głębokość zagnieżdżenia jeżeli obecnie korzystamy z opcji z zagnieżdżeniami

```
4
Enter partname of file you are looking for: s
stack.bat 0
sortowanie_ST.pdf 0
selectdiagram.png 0
shelldiagram.png 0
stolarzkonrad.tar 0
```

Opcją 5 oraz korzystając z opcji zagnieżdżenia możemy wyszukać np. pliki tekstowe:

```
5
Enter type of file you are looking for: txt
DANE.txt 0
danelosowe.txt 0
daneodwrotnie.txt 0
daneposortowane.txt 0
danedowykresu_losowe_zestaw1.txt 1
danedowykresu_losowe_zestaw2.txt 1
danedowykresu_odwrotne_zestaw2.txt 1
danedowykresu_odwrotnie_zestaw1.txt 1
danedowykresu_posortowane_zestaw1.txt 1
danedowykresu_posortowane_zestaw2.txt 1
```

Opcją 6 tworzymy plik

```
Enter name of file you want to create: nowyplik.jpg
Created succesfully!

After add:

1nowy.jpg
2nowy.jpg
dodajemynowyplik.jpg
dokumentacja_organiezer_ks.docx
main.c
main.o
Makefile.win
nowy.jpg
nowyp.txt
nowyplik.jpg
nowyplik.txt
organizer.dev
organizer.exe
organizer.layout
plik.txt
pliktestowy.txt
Project3.dev
Project3.layout
test (KATALOG)
test2.txt
~$kumentacja_organiezer_ks.docx
~WRL1206.tmp
```

Opcją 7 usuwamy plik

```
Enter name of file you want to remove: nowyplik.jpg
Removed succesfully!
```

```
1
1nowy.jpg
2nowy.jpg
dodajemynowyplik.jpg
dokumentacja_organiezer_ks.docx
main.c
main.o
Makefile.win
nowy.jpg
nowyp.txt
nowyplik.txt
organizer.dev
organizer.exe
organizer.layout
plik.txt
pliktestowy.txt
Project3.dev
Project3.layout
test (KATALOG)
test2.txt
~$kumentacja_organiezer_ks.docx
~WRL1206.tmp
```

Opcją 9 opuszczamy program

```
9
Thanks for using
--------------------------------
Process exited after 90.69 seconds with return value 0
Press any key to continue . . .
```