Wykład 7

- Funkcje – ciąg dalszy
- Operacje dyskowe

Instytut Informatyki

dr inż. Marcin Radom

Wskaźniki do funkcji

- Funkcje w języku C nie są zmiennymi, ale można definiować wskaźniki do funkcji.
- Takim wskaźnikom można nadawać wartości, umieszczać je w tablicach, przekazywać do funkcji, zwracać jako wartość funkcyjną etc.
- **Przykład definicji wskaźnika do funkcji:**

```
int ( *pf )( );
```

- Taką definicję odczytuje się następująco:
  - Najpierw nazwę funkcji.
  - Następnie należy poruszać się w prawo (ponieważ po prawej stronie mogą stać tylko operatory **()** lub **[]** i są one „najsilniejsze” z możliwych).
  - Po dojściu do nawiasu zamykającego poruszamy się w lewo.
  - Jeśli odczytaliśmy wszystko w obrębie danego nawiasu to wychodzimy na zewnątrz niego i znów poruszamy się w prawo.

Wskaźniki do funkcji

- Zatem powyższą definicję czytamy następująco:
  - nazwa funkcji to **`pf`**,
  - w prawo jest nawias zamykający, więc idziemy w lewo - **`( *pf )`** - jest **wskaźnikiem**, wychodzimy z nawiasu zamykającego
  - Idziemy znów w prawo gdzie stoi operator wywołania funkcji – **`( *pf )( )`** - **do funkcji wywoływanej bez żadnych argumentów** (nawias był pusty)
  - teraz w lewo – **zwracającą** - **`int ( *pf )( )`** - wartość typu ***int***.

- Gdybyśmy pominęli nawiasy okrągłe - **`int *pf ( )`** - to byłaby to deklaracja funkcji zwracająca wskaźnik do typu ***int*** i wywoływanej bez żadnych argumentów (dlatego, że nawiasy są „silniejsze” niż operator adresowania pośredniego ***).

- **Nazwa funkcji jest inaczej jej adresem w pamięci** (adres miejsca w pamięci, gdzie zaczyna się kod będący instrukcjami danej funkcji).

- Dzięki temu można go łatwo ustawić, aby pokazywał na daną funkcję:

```
wskaźnik = nazwa_funkcji; // to nie jest wywołanie funkcji
// wskaźnik = nazwa_funkcji() – błąd, próba wywołania funkcji
// i przypisania wyniku pod wskaźnik do funkcji
```

Wskaźniki do funkcji

- **Przykład:**

```
int ( *pf )( int, int );  // typ wyniku i sygnatura muszą być zgodne

int Ki( int x, int y ){
   return (x + y) * (x + 2 * y);
}

long Kl( int x, int y ){
   return (long)(x + 3 * y) * (x - y);
}

pf = Ki;        // poprawnie (kompilator odkryje, że pf jest
                // wskaźnikiem i poprawnie go ustawi)
pf = & Ki;      // poprawnie (użycie operatora & w celu uzyskania
                // adresu funkcji jest opcjonalne)
pf = Kl;        // błąd, nieodpowiedni wskaźnik (zły typ wartości
                // przekazywanej z funkcji)
```

Wskaźniki do funkcji

- **Wywołanie funkcji - przykład:**

```
void bubbleSort ( float T[ ], int n ) { ... }
void mergeSort ( float T[ ], int n ) { ... }
void heapSort ( float T[ ], int n ) { ... }
// można nadać wskaźnikowi do funkcji początkową wartość
void ( *wS )( float[ ], int ) = bubbleSort;
void ( *nS )( float[ ], int ) ;

nS = wS; // ns i ws wskazują na funkcję bubbleSort
float TAB[ 150 ];

/* równoważne wywołania funkcji bubbleSort: */

wS ( TAB, 150 ); // można użyć identyfikatora wskaźnika do
                 // funkcji tak jak nazwy funkcji
// wywołanie tego na co pokazuje wskaźnik, czyli funkcji bubbleSort
( *nS )( TAB, 150 );        // nawiasy są konieczne w celu zapewnienia
                            // prawidłowego powiązania składowych
```

Tablice wskaźników do funkcji

- Często korzysta się z tablicy wskaźników do funkcji, przykład:

```
void (*(twf[10]) )();
```

  - Definicję taką czyta się następująco:
    - **twf** – nazwa tablicy *twf*
    - **[10]** – jest 10 elementową tablicą
    - ***** - wskaźników
    - **()** - do funkcji wywoływanej bez żadnych argumentów
    - **void** - zwracającą typ **void** (czyli nic)

- W ramach takiej tablicy możemy przechowywać wybrane funkcje naszego programu (np. systemu menu w programie i łatwo je modyfikować w trakcie wykonywania programu).

- Po zdefiniowaniu tablicy należy zdefiniować jej wskaźniki udostępniające np. czynności wybierane z menu:

```
twf[1] = wyszukaj;    // ustaw drugi wskaźnik z tablicy twf na funkcję wyszukaj
```

- **Przykład wywołania:**

```
twf[1]();                          // wywołaj drugą funkcję z tablicy twf
```

Tablice wskaźników do funkcji - przykład

- **Przykłady:**

```
int fA(double x) { ... }
int fB(double y) { ... }
int fC(double z) { ... }
      // inicjowanie - przypisanie adresów funkcji do wskaźników w tablicy
int ( *fX [3] )( double ) = { fA, fB };
fX[2] = fC;

fX[1]( 37.18 );           // wywołanie fB

      // definicja z ustawieniem wartości początkowych
int ( *Test [10] )( char* ) = { /* ..... */ };
int Wyniki [10];          // wyniki funkcji z tablicy wskaźników Test
char* Napisy[10];         // argumenty funkcji z tablicy wskaźników Test
for (int i = 0; i < 10; ++i )
    Wyniki[i] = Test[i] ( Napisy[i] ); // wywołania funkcji
```

Tablice wskaźników do funkcji

- Można wprowadzić synonim skomplikowanego typu za pomocą instrukcji **typedef**.

- **Przykład:**
  - Moglibyśmy wprowadzić synonim typu *wskaźnik do funkcji z dwoma parametrami typu **float** zwracającej **int***.
  - Następnie użyć go w deklaracji/definicji funkcyjnych zmiennych wskaźnikowych.

```
typedef int ( *PF )( float, float ); // PF to nazwa typu

int Fun( float a, float b ) { ... }

PF f1, f2 = Fun;
PF Tfun[15]; // tablica wskaźników do funkcji
```

Wskaźniki funkcji jako argument funkcji

- Wskaźnik do funkcji może być również argumentem funkcji.

- **Przykład:**

```
// argumentem jest konkretna metoda sortowania
void Sort( float[ ], int, void (*) ( float[ ], int ) = mergeSort );

void Sort( float TAB[ ], int rozmiar, void (*fS) (float[ ], int) )
{  ..................
   fS( TAB, rozmiar );
   ..................
}

float T1[100], T2[200];
void ( *wF )( float[ ], int ) = heapSort;

Sort( T1, 100, bubbleSort );        // bubbleSort
Sort( T2, 200, wF );                // heapSort
Sort( T2, 200 );                    // domyślnie mergeSort
```

Wskaźniki funkcji jako argument funkcji

- Wskaźnik do funkcji może też być wartością zwracaną przez funkcję.
- **Przykład:**

```
typedef char* ( *NP )( char*, char* ); // NP to nazwa typu

struct FUNKCJE {
      int cecha; // na jej podstawie wybór funkcji do wywołania
      NP funkcja; // wskaźnik do funkcji
} TABLICA[15];

// zwraca wskaźnik do funkcji
NP Szukaj( struct FUNKCJE TAB[ ], int rozmiar, int wzorzec ) {
      for (int i = 1; i < rozmiar; ++i)
          if (TAB[i].cecha == wzorzec)
                return TAB[i].funkcja;
      return TAB[0].funkcja;
}

// wywołanie wybranej funkcji
printf( "%s\n" , Szukaj ( TABLICA, 15, 1527 )( "Alf","Ogi") );
```

Wskaźniki do funkcji - Przykład

- **Przykład – Kalkulator**

```
#include <math.h>

// Program oblicza wartości funkcji sin, cos, tan, cotan, sqrt, log, recip, sqr

double FSin ( double x, bool &err)
{ return sin(x); }

double FCos ( double x, bool &err)
{ return cos(x); }

double FTan ( double x, bool &err) {
      if (cos(x) != 0)
          return tan(x);
      else {
          err = true;
          return 0;
      }
}
```

Wskaźniki do funkcji – Przykład cd.

```
double FCotan ( double x, bool &err){
      if (sin(x) != 0)
          return 1 / tan(x);
      else {
          err = true;
          return 0;
      }
}

double FSqrt ( double x, bool &err){
      if (x >= 0)
          return sqrt(x);
      else {
          err = true;
          return 0;
      }
}
```

Wskaźniki do funkcji – Przykład cd.

```
double FLog ( double x, bool &err) {
      if (x > 0)
          return log(x);
      else {
          err = true;
          return 0;
      }
}

double FRecip ( double x, bool &err){
      if (x != 0)
          return 1 / x;
      else {
          err = true;
          return 0;
      }
}

double FSqr ( double x, bool &err)
{ return x * x; }
```

Wskaźniki do funkcji – Przykład cd.

```
int main() {
     double (*TabFun[8])(double, bool&) = { FSin, FCos, FTan, FCotan, FSqrt,
     FLog, FRecip, FSqr};

     int opcja;
     double arg, wynik;
     bool dalej = true, nieodpowiedni;

     while (dalej) {
          printf("\nWybierz funkcje "
             "\n0 - sin,"
             "\n1 - cos,"
             "\n2 - tan,"
             "\n3 - cotan,"
             "\n4 - sqrt,"
             "\n5 - log,"
             "\n6 - recip,"
             "\n7 - sqr,"
             "\ninna  - koniec : ");
          scanf("%d", &opcja);
```

Wskaźniki do funkcji – Przykład cd.

```
        if(opcja < 0 || opcja > 7){
              printf("\nKoniec.\n");
              dalej = false;
        } else {
              printf("Podaj wartosc x : ");
              scanf("%lf", &arg);

              nieodpowiedni = false;
              // wywołanie odpowiedniej funkcji
              wynik = TabFun[opcja](arg, nieodpowiedni);

              if (nieodpowiedni)
                  printf("Nieodpowiedni argument.\n");
              else
                  printf("Wynik = %lf\n", wynik);
              }
        }
        return 0;
}
```

Operacje dyskowe

- Pliki, dostęp, zapis i odczyt

Pliki dyskowe

- **Plik** jest wydzielonym fragmentem pamięci (najczęściej dyskowej) posiadającym nazwę.
- Z punktu widzenia C plik jest ciągiem bajtów, z których każdy może zostać oddzielnie odczytany.
- Zgodnie ze standardem ANSI dwoma sposobami postrzegania plików przez program są widok *binarny* i widok *tekstowy*.
- W **widoku binarnym** każdy bajt pliku dostępny jest dla programu.
  - Elementem jednostkowym takiego pliku jest bajt.
  - Format pliku binarnego: [bajty pliku][EOF]
- W **widoku tekstowym** to co „widzi” program może się różnić od tego, co rzeczywiście znajduje się w pliku (np. program odczytujący plik na komputerze Macintosh zamienia przy odczycie \r na \n, a przy zapisie \n na \r -> w języku C koniec wiersza zawsze reprezentowany jest przez znak \n).
  - Elementem jednostkowym takiego pliku jest znak.
  - Format pliku tekstowego:
    [znaki w 1 linii pliku][/r][/n]
    …
    [znaki w ostatniej linii pliku][EOF]

Pliki dyskowe – Pliki tekstowe i binarne

- **Przykład:** zapis liczby 25
  - **Plik tekstowy**
    - Zostanie zapisana za pomocą dwóch kodów ASCII cyfr: '2' oraz '5' plus np. znak spacji.

32H	35H	20H

    - Funkcje (np. **fprintf**) zapisujące wartości numeryczne do pliku tekstowego przetwarzają je na łańcuchy.
    - Może to prowadzić do utraty dokładności (np. wartość 0.33 zostanie zapisana za pomocą 4 znaków i nie ma możliwości odzyskania dalszych miejsc po przecinku przy odczytywaniu pliku)
  - **Plik binarny**
    - Najbardziej precyzyjną metodą przechowania liczby jest wyrażenie jej za pomocą tego samego układu bitów, z którego korzysta program, np. wartość typu **double** powinna być zapisana w obszarze o rozmiarze typu **double**.

Pliki dyskowe – Pliki tekstowe i binarne

- **Gdy dane w pliku są przedstawione w sposób zgodny z ich reprezentacją w programie, mówimy, że są one zapisane w *postaci binarnej*.**
- W takim przypadku nie zachodzi konwersja między postacią numeryczną a łańcuchem.
- W pliku binarnym liczba 25 (zakładając, że jest to typ *int*) jest zapisana za pomocą 4 bajtów w konwencji *little-endian* (forma zapisu danych, w której mniej znaczący bajt (zwany też dolnym bajtem) umieszczony jest jako pierwszy).

19H	0H	0H	0H

- W standardowym pakiecie wejścia/wyjścia wymianą danych w formie binarnej zajmują się funkcję **fread** i **fwrite**.

- W rzeczywistości wszystkie dane są zapisane w formie binarnej (nawet znaki).
- Jeśli wszystkie dane w pliku interpretowane są jako kody znaków to mówimy, że plik zawiera dane tekstowe.
- Jeśli niektóre bądź wszystkie dane interpretowane są jako dane numeryczne w postaci binarnej to mówimy, że plik zawiera dane binarne.

Pliki dyskowe

- Do reprezentowania plików w programach służą zmienne określane strumieniami plikowymi (strumieniami).
- Strumienie w C reprezentowane są przez zmienne typu **FILE**.
- Struktura o nazwie **FILE** zadeklarowana jest w pliku nagłówkowym **<stdio.h>** i zawiera następujące informacje o pliku:
  - położenie bufora,
  - bieżąca pozycja znaku w buforze,
  - rodzaj dostępu do pliku (czytanie, pisanie itp.),
  - sygnały o wystąpieniu błędów lub napotkaniu końca pliku etc.
- Wszystkie operacje na strumieniu wymagają podania wskaźnika na strukturę typu **FILE** (zdefiniowana za pomocą *typedef*).
- Deklarację dla wskaźnika pliku przedstawia przykład:

```
FILE *fp; //fp jest wskaźnikiem do struktury typu FILE
```

Pliki dyskowe

- Plik **<stdio.h>** definiuje trzy wskaźniki plikowe przyporządkowane trzem standardowym plikom automatycznie otwieranym przez programy w języku C:
  - **stdin** – standardowe wejście (zwykle klawiatura)
  - **stdout** – standardowe wyjście (zwykle ekran)
  - **stderr** – standardowe wyjście dla błędów (zwykle ekran)

- Powyższe wskaźniki należą do typu FILE, mogą więc służyć jako argumenty standardowych funkcji wejścia/wyjścia, tak samo jak wskaźnik *fp* (zdefiniowany na poprzednim slajdzie).

- **Fazy pracy z plikiem:**
  - Otwarcie pliku
  - Zapis lub odczyt danych (zawsze od pozycji aktualnej, przesunięcie tej pozycji w kierunku końca pliku o liczbę przeczytanych lub zapisanych bajtów)
  - Zamknięcie pliku

Pliki dyskowe

- Do operacji na plikach dyskowych wykorzystuje się w C standardowy pakiet wejścia/wyjścia z pliku **<stdio.h> (#include <stdio.h>).**

- **Pakiet ten oferuje 3 grupy funkcji :**
  - **fopen, fclose, fcloseall** (otwieranie i zamykanie plików)
  - **ftell, fseek, rewind, feof** (ustalanie aktualnej pozycji pliku, wykrywanie końca pliku)
  - **fread, fwrite, fgetc, fputc, fgets, fputs, fscanf, fprintf** (odczyt i zapis danych)

- **Otwarcie pliku:**
  - W celu otwarcia pliku należy wywołać funkcję ***fopen()***. Przy wywołaniu należy podać ścieżkę do pliku oraz tryb otwarcia.
  - Funkcja **fopen** zwraca wskaźnik do pliku (wskaźnik do struktury typu **FILE**).
  - Funkcja **fopen** zwraca adres NULL jeśli nie dojdzie do otwarcia strumienia.

Pliki dyskowe – otwarcie pliku

- Funkcja **fopen**

```
FILE* fopen(const char* nazwa, const char* tryb);
```

Funkcja: otwarcie pliku ”nazwa”
Tryb :

r	:	odczyt istniejącego pliku
w	:	utworzenie pliku do zapisu
a	:	zapis na końcu istniejącego pliku
r+	:	zapis lub odczyt istniejącego pliku
w+	:	utworzenie pliku do zapisu i odczytu
a+	:	zapis lub odczyt na końcu istniejącego pliku

Dodatkowo w pewnych systemach rozróżnia się pliki tekstowe i binarne

t	:	plik tekstowy
b	:	plik binarny

- Otwarcie nieistniejącego pliku do pisania/dopisywania stworzy go
- Otwarcie istniejącego pliku do pisania skasuje jego starą zawartość!

Pliki dyskowe – otwarcie pliku - fopen

- **Przykład:**

```
#include <stdio.h>

int main(void)
{
                ...
     FILE *spis;
     spis = fopen("SPIS.TXT","rt+");
     if (spis == NULL)
       printf("\nNie można otworzyć pliku.");
                ...
                return 0;
}
```

Pliki dyskowe – zamknięcie pliku

- Funkcja **fclose**

```
int fclose ( FILE *plik );
```

Funkcja: zamknięcie pliku

Wynik: 0 – jeśli akcja zakończyła się pomyślnie **lub EOF** – w przeciwnym wypadku

Przykład:

```
#include <stdio.h>
int main(void)
{ ...
      FILE *personel = fopen ( "PER.TXT",  "rt" );
      ...
      fclose ( personel );
      ...
}
```

Pliki dyskowe – zamknięcie pliku

- Funkcja **fclose** 'zrywa' połączenie pomiędzy wskaźnikiem do pliku (ustanowione przez funkcję **fopen**) a nazwą pliku, zwalniając wskaźnik dla innego pliku.
- Jest to o tyle istotne, że większość systemów operacyjnych nakłada ograniczenia na maksymalną liczbę jednocześnie otwartych plików w jednym programie.
- Należy więc zwalniać wskaźniki plików, gdy nie są już dłużej potrzebne.
- Funkcja **fclose** jest wywoływana automatycznie dla wszystkich otwartych jeszcze plików, jeśli program zakończy się normalnie.
- Funkcja **fcloseall**

```
int fcloseall ( void );
```

Funkcja: zamkniecie wszystkich otwartych plików dyskowych

Liczba zamkniętych plików – jeśli akcja zakończyła się pomyślnie **lub EOF** – w przeciwnym wypadku.

Pliki dyskowe – ustalenie aktualnej pozycji pliku

- Funkcja **fseek** pozwala potraktować plik tak, jak gdyby był tablicą i przejść bezpośrednio do dowolnego przechowywanego w nim bajtu.

- Funkcja **fseek** ma 3 argumenty:
  - Wskaźnik do pliku.
  - Przesunięcie (pozycja, offset)
    - Określa odległość i kierunek przemieszczenia względem punktu początkowego.
    - Musi być typu **long**.
    - Może być *dodatni* (przejście do przodu), *ujemny* (przejście do tyłu) lub *równy zeru* (pozostanie w miejscu).
  - Tryb (cel), który określa punkt początkowy.
    - Plik nagłówkowy **stdio.h** definiuje dla trybów następujące stałe symboliczne:
      - **SEEK_SET** – początek pliku
      - **SEEK_CUR** – bieżąca pozycja
      - **SEEK_END** – koniec pliku

Pliki dyskowe – ustalenie aktualnej pozycji pliku

- Funkcja **fseek**

```
int fseek (FILE *plik, long pozycja, int cel);
```

Funkcja: ustalenie aktualnej pozycji pliku

cel :

SEEK_SET - początek pliku,

SEEK_CUR - aktualna pozycja pliku

SEEK_END - koniec pliku

pozycja : +/- , od 0, można poza koniec pliku

Wynik: 0 : poprawnie, !0 : błąd

Pliki dyskowe – ustalenie aktualnej pozycji pliku

- **Przykład:**

```
#include <stdio.h>
int main(void)
{
        ...
        long pp;
        FILE *opis = fopen("OP1.DOC","rt+");
                // powoduje przejście w miejsce położone 0 bajtów za
                // końcem pliku, czyli po prostu do końca pliku
        fseek(opis, 0L, SEEK_END);
        ...     // zapis na końcu pliku
                // powoduje przesunięcie o 0 bajtów względem początku pliku
        fseek(opis, 0L, SEEK_SET);
        ...     // zapis na początku pliku
        pp = 15453l;        // przesunięcie względem bieżącej pozycji
        fseek(opis, pp, SEEK_CUR);
        ...     // odczyt z pliku
        ...
}
```

Pliki dyskowe – ustalenie aktualnej pozycji pliku

- Funkcja **rewind** ustawia wskaźnik pliku na pozycję początkową.
- Wywołanie ***rewind****(fp)* jest równoważne wywołaniu: ***fseek****(fp, 0L, SEEK_SET)*

```
void rewind ( FILE *plik );
```

Funkcja: ustalenie aktualnej pozycji na początku pliku

Przykład:

```
#include <stdio.h>
int main(void)
{
     ...
     FILE *roboczy = fopen("R1.DAT","rt+");
     fseek(roboczy, 0L, SEEK_END);
     ...              // zapis na końcu pliku
     rewind(roboczy);   // początek pliku
     ...
}
```

Pliki dyskowe – funkcja pomocnicza feof

- Funkcja **feof** pozwala na testowanie osiągnięcia końca pliku.

```
int feof ( FILE *plik );
```

Funkcja: odczyt stanu znacznika końca pliku

Wynik: zwraca !0 : napotkano **EOF** (znacznik końca pliku), przeciwnie 0

Przykład:

```
#include <stdio.h>
int main(void)
{ ...
      FILE *plik1 = fopen("P1.MAN","rt");
      ...          // odczyt z pliku
      if (feof(plik1)!= 0)
         printf("\nKoniec pliku.");
      ...
}
```

Pliki dyskowe – odczyt danych

- Funkcja **fgetc** czyta ze strumienia *plik* następny znak i zwraca jego wartość potraktowaną jako *unsigned char* (i przekształconą do *int*). W przypadku wystąpienia błędu lub końca pliku zwraca **EOF**.

```
int  fgetc ( FILE *plik );
```

Funkcja: odczyt kolejnego znaku

Wynik: liczba całkowita 000 | kod_znaku

Przykład:

```
#include <stdio.h>
int main(void)
{
     ...
     char cc;
     FILE *info = fopen("INF.DOC","rt");
     cc = fgetc(info);
     ...
}
```

Pliki dyskowe – zapis danych

- Funkcja **fputc** wpisuje znak *znak* (przekształcony do *unsigned char*) do strumienia *plik*. Zwraca wypisany znak (jego kod ASCII) lub **EOF** w przypadku błędu.

```
int  fputc ( int znak,  FILE *plik );
```

Funkcja: zapis kolejnego znaku (liczba całkowita : 000 | kod)

Wynik: znak lub **EOF**

Przykład:

```
#include <stdio.h>
int main (void)
{
    char cc = 'K';
    FILE *dane = fopen("DANE.DOC","rt+");
    ...
    fputc(cc, dane);
    ...
}
```

Pliki dyskowe – odczyt danych

- Funkcja **fgets** czyta co najwyżej *liczba-1* znaków i wstawia je do tablicy *napis*. Funkcja przerywa czytanie po napotkaniu znaku nowego wiersza – znak ten także wstawia do tablicy. Tekst zakończony jest znakiem '\0'.

```
char* fgets ( char *napis,  int liczba,  FILE *plik);
```

Funkcja: odczyt ciąg znaków, maksymalnie *liczba-1* znaków

Wynik: napis lub **NULL** w przypadku napotkania końca pliku lub błędu

Przykład:

```
#include <stdio.h>
int main(void)
{
    ...
    char nazwisko[16];
    FILE *teczka = fopen("TECZ.DOC","rt");
    fgets(nazwisko, 16, teczka);
    ...
}
```

Pliki dyskowe – zapis danych

- Funkcja **fputs** wypisuje tekst zawarty w tablicy *napis* do (nie musi zawierać znaku końca linii) do strumienia *plik*. Zwraca wypisany znak lub **EOF** w przypadku błędu.

```
int fputs ( char *napis, FILE *plik );
```

Funkcja: zapis ciągu znaków

Wynik: ostatni zapisany znak lub **EOF**

Przykład:

```
#include <stdio.h>
int main(void)
{
    ...
    char *spis = "Spis treści";
    FILE *dok = fopen("DOK1.DOC","rt+");
    fputs(spis, dok);
    ...
}
```

Pliki dyskowe – odczyt danych

- Funkcja sformatowanego wejścia **fscanf** działa tak samo jak *scanf*, z tą różnicą, że wymaga podania dodatkowego argumentu określającego plik.
- Funkcja kończy działanie, gdy zinterpretuje cały format.
- Funkcja zwraca **EOF**, jeśli przed jakimkolwiek przekształceniem napotka koniec pliku lub wystąpi jakiś błąd. W przeciwnym przypadku zwraca liczbę przekształconych i przypisanych danych wejściowych.
- Funkcja **fscanf**

D.
```
int fscanf(FILE *plik, const char *format,
                wskaźnik, wskaźnik, ...);
```

F. Odczyt ciągów znaków i konwersji na wartości binarne (podobnie jak scanf)

W. Liczba wczytanych ciągów znaków lub EOF

Pliki dyskowe – odczyt danych

- **Przykład:**

```
#include <stdio.h>
int main(void)
{
        ...
        int sztuki;
        float cena;
        FILE *towar = fopen("SPIS_TOW.DOC","rt");
        ...
        fscanf(towar, "%d%f", &sztuki, &cena);
        ...
}
```

Pliki dyskowe – zapis danych

- Funkcja sformatowanego wyjścia **fprintf** działa tak samo jak *printf*, z tą różnicą, że wymaga podania dodatkowego argumentu określającego plik.
- Funkcja zwraca liczbę ujemną jeśli wystąpił jakiś błąd. W przeciwnym przypadku zwraca liczbę wypisanych znaków.

```
int fprintf ( FILE *plik, const char *format,
    wyrażenie, wyrażenie, ...);
```

Funkcja: zapis ciągów znaków zadanych za pomocą wyrażeń (podobnie jak **printf**)

Wynik: liczba zapisanych bajtów lub **EOF**

Pliki dyskowe

- **Przykład:**

```
#include <stdio.h>
int main(void)
{
    ...
    int kod_waluty = 15;
    float kurs_biezacy = 0.23547;
    FILE *tabela_kursow = fopen("KURSY.TAB","wt");
    ...
    fprintf(tabela_kursow, "\n%3d\t%8.3f",
    kod_waluty, kurs_biezacy);
    ...
}
```

Pliki dyskowe

- Funkcja **fread** wczytuje ze strumienia *plik* do tablicy *wskaźnik* co najwyżej *liczba* obiektów o rozmiarze *rozmiar*.
- Funkcja **fread** powinna być stosowana do odczytania danych, które zostały zapisane za pomocą funkcji **fwrite**.

```
int fread (wskaźnik,  int rozmiar, int liczba,  FILE *plik);
```

Funkcja: odczytanie zadanej *liczby* struktur danych, każda o długości *rozmiar*

Wynik: liczba odczytanych struktur lub 0

Przykład:

```
double ceny[50];
...
fread(ceny, sizeof(double), 50, fp);
// powyższe wywołanie kopiuje 50 wartości typu double z
// pliku do tablicy ceny
```

Pliki dyskowe

- **Przykład:**

```
#include <stdio.h>
int main(void)
{
    ...
    struct ksiazka
    {
        char autor[25];
        char tytul[50];
    } ksiazki[100];
    ...
    FILE *magazyn = fopen("MAG.DOC","rt");
    fread(ksiazki, sizeof(ksiazka), 100, magazyn);
    ...
}
// powyższe wywołanie kopiuje 100 obiektów typu
// ksiazka z pliku do tablicy ksiazki
```

Pliki dyskowe

```
void main(void)
{
     int liczba_odczytow;

     struct pozycja {
         double wspolrzedne[2];
         double wysokosc;
     } marszruta[1000];
     ...
     FILE *wycieczka = fopen("W1.DAT","rb");
     // odczytanie ile jest elementów w tablicy
     fread(&liczba_odczytow, sizeof(int), 1, wycieczka);
     // wczytanie elementów do tablicy marszruta
     fread(marszruta, sizeof(pozycja), liczba_odczytow, wycieczka);
     ...
}
```

Pliki dyskowe

- Funkcja **fwrite** wypisuje *liczba* obiektów o rozmiarze *rozmiar*, pochodzących z tablicy *wskaźnik* do strumienia *plik*.

```
int fwrite ( wskaźnik, int rozmiar, int liczba, FILE *plik
);
```

Funkcja: zapis wskazanej *liczby* struktur danych o długości *rozmiar*

Wynik: liczba zapisanych struktur lub 0

Przykład:

```
double ceny[50];
...
fwrite(ceny, sizeof(double), 50, fp);
// powyższe wywołanie przepisuje do pliku dane z tablicy
// ceny w 50 porcjach o rozmiarze double
```

Pliki dyskowe

- **Przykład:**

```
#include <stdio.h>
int main(void)
{
      ...
      long double pomiary[wie][kol];
      ...
      FILE *archiwum = fopen("ARCH.TAB","wb");
      fwrite(pomiary, sizeof(pomiary), 1, archiwum);
      ...
}
```

Przykłady programów

- Za samodzielnej analizy

Pliki dyskowe – Przykład 1

```
#include <stdio.h>

// analizuje plik Dane.txt zawierający liczby cakowite: liczby parzyste
// przepisuje do pliku Parzyste.txt, a nieparzyste do pliku Nieparzyste.txt
int main (int n, char* TS[ ]){ // Parzyste / Nieparzyste
        FILE *Dane, *Parzyste, *Nieparzyste;
        int Liczba;

        Dane = fopen ("Dane.txt", "rt");
        Parzyste = fopen ("Parzyste.txt", "wt");
        Nieparzyste = fopen ("Niepatrzyste.txt", "wt");

        if (Dane == NULL || Parzyste == NULL || Nieparzyste == NULL) {
                printf ("\nNie mozna otworzyc pliku.\n\n");
                return 0;
        }
        // ilość liczb w pliku Dane.txt
        fscanf(Dane, "%d", &Liczba);
```

Pliki dyskowe – Przykład 1 cd.

```
        while ( feof(Dane) == 0 ){
            if( Liczba & 1)
                 fprintf(Nieparzyste, "%d ", Liczba);
            else
                 fprintf(Parzyste, "%d ", Liczba);

            fscanf(Dane, "%d", &Liczba);
        }

        fcloseall();
        printf ("\nKoniec.\n\n");
        return 0;
}
```

Pliki dyskowe – Przykład 2

- Plik wejściowy zawiera ciąg liczb całkowitych oddzielonych znakami '#'. Opracować program, który otwiera do odczytu taki plik wejściowy (zapytać o nazwę) i przepisuje z niego liczby naprzemiennie do plików z rozszerzeniem *.fir i *.sec

```
#include <stdio.h>
#include <string.h>

int main (int n, char* TS[ ]) {// split using #
     char Ntxt[64];
     char Nfir[64];
     char Nsec[64];

     FILE *Ptxt, *Pfir, *Psec;

     bool Koniec = false, Ktory = true;
     int Znak;
     int Dot;
     char *Ptr;
```

Pliki dyskowe – Przykład 2 cd.

```
if ( n < 2 ){
    printf ("\nBrak nazwy pliku.\n„ "Wywolaj program podajac jako
                    parametr nazwe pliku.\n\n");
    return 0;
}
strncpy(Ntxt, TS[1], 58);
strncpy(Nfir, TS[1], 58);
strncpy(Nsec, TS[1], 58);
// szuka pierwszego wystąpienia znaku '.' i ustala odległość od pocz.
if ((Ptr = strchr(Ntxt, '.')) != NULL)
    Dot = Ptr - Ntxt;
else {
    printf("\nBledna nazwa pliku.\n\n");
return 0;
}

// nadaje nowe rozszerzenie zachowując nazwę pliku wejściowego
strcpy(Nfir + Dot + 1, "fir");
strcpy(Nsec + Dot + 1, "sec");

Ptxt = fopen (Ntxt, "r");
Pfir = fopen (Nfir, "w+");
Psec = fopen (Nsec, "w+");
```

Pliki dyskowe – Przykład 2 cd..

```
        if (Ptxt == NULL || Pfir == NULL || Psec == NULL) {
            printf ("\nNie mozna otworzyc pliku\n\n");
            return 0;
        }

        while ( !Koniec ){
            // odczytuje plik wejściowy znak po znaku
            Znak = fgetc(Ptxt);

            // zapisuje na zmianę do dwóch plików
            if ( !(Koniec = (feof(Ptxt) != 0)))
                  if (Znak == '#')
                      Ktory = !Ktory;
                  else
                      if (Ktory)
                            fputc(Znak, Pfir);
                      else
                            fputc(Znak, Psec);
    }

    printf ("\nKoniec.\n\n");
    return 0;
}
```

Pliki dyskowe – Przykład 3

- Opracować program prowadzący spis pracowników firmy (max.. 50 pracowników). Każdy pracownik opisany jest za pomocą struktury zawierającej nazwisko i pensję. Program realizuje następujące polecenia:
  - R : wczytanie liczby pracowników i tablicy struktur opisujących pracowników z pliku dyskowego (zapytać o nazwę pliku),
  - N : nowy pracownik - wczytać dane opisujące pracownika i wprowadzić do kolejnej pozycji tabeli struktur,
  - W : wyświetlanie informacji o wszystkich pracownikach,
  - Z : zapis liczby pracowników i tabeli pracowników do pliku dyskowego (zapytać o nazwę pliku),
  - K : koniec programu.

Dla realizacji poszczególnych opcji zdefiniować funkcje.

Pliki dyskowe – Przykład 3 cd.

```
#include <stdio.h>
struct Pracownik{
          char Nazwisko[32];
          double Pensja;
};

void Z_Pliku(Pracownik Tab[], int& ile){
     FILE* plik;
     char nazwa[16];
     printf("Podaj nazwe pliku do odczytu : ");
     fflush(stdin);
     scanf("%15s", nazwa);
          plik = fopen(nazwa,"rt");
          if (plik == NULL) {
                    printf("Brak pliku.");
                    return;
          }
          fscanf(plik,"%d",&ile); // ilość pracowników w tablicy
          fread(Tab, sizeof Pracownik, ile, plik);
          fclose(plik);
}
```

Pliki dyskowe – Przykład 3 cd..

```
void Do_Pliku(Pracownik Tab[], int ile){
    FILE* plik;
    char nazwa[16];
    printf("Podaj nazwe pliku do zapisu : ");
    fflush(stdin);
    scanf("%15s", nazwa);

    plik = fopen(nazwa,"wt");

    // ilość pracowników w tablicy
    fprintf(plik,"%d",ile);
    // tablica pracowników
    fwrite(Tab, sizeof Pracownik, ile, plik);
    fclose(plik);
}
```

Pliki dyskowe – Przykład 3 cd..

```
void Nowy(Pracownik Tab[], int& ile){
     if (ile == 50){
          printf("Tablica pelna.\n");
          return;
     }
     printf("Podaj nazwisko : ");
     fflush(stdin);
     scanf("%31s",Tab[ile].Nazwisko);
     printf("Podaj pensje : ");
     scanf("%lf",&Tab[ile++].Pensja);
}

void Wyswietl(Pracownik Tab[], int ile){
     if (ile == 0){
          printf("Tablica pusta.\n");
          return;
     }
     for(int i = 0; i < ile; i++)
          printf("Pracownik %d : %s , %.2lf\n", i, Tab[i].Nazwisko, Tab[i].Pensja);
}
```

Pliki dyskowe – Przykład 3 cd..

```
int main(int argc, char* argv[]){
     Pracownik TaPa[50];
     bool dalej = true;
     int ile = 0;
     char opcja;
     while (dalej) {
           printf("Wybierz opcje [R,N,W,Z,K] : ");
           fflush(stdin);
           scanf("%c",&opcja);
           switch(opcja & 0x5F){
                     case 'R' :Z_Pliku(TaPa, ile); break;
                     case 'N' :Nowy(TaPa, ile); break;
                     case 'W' :Wyswietl(TaPa, ile); break;
                     case 'Z' :Do_Pliku(TaPa, ile); break;
                     case 'K' :dalej = false; break;
                     default : printf("Zla opcja.\m");
           }
     }
     return 0;
}
```

Pytania?