

# Podstawy programowania - wykłady

## Wykład 1

### Algorytmy

Algorytm - wg wykładu jest to przepis podający sposób rozwiązania określonego zadania w skończonej liczbie kroków

Algorytm zapisany w języku programowania to program

Program nie jest algorytmem

Podziały:

- sekwencyjne - kolejność czynności jest określona jednoznacznie
- niesekwencyjne - następstwo między pewnymi operacjami nie jest określona
- numeryczne - operują na liczbach
- nienumeryczne - operują na obiektach nieliczbowych

### Języki nisko- i wysokopoziomowe

Niskie - jedna zapisana instrukcja odpowiada jednemu rozkazowi wykonywanemu przez procesor

Wysokie - jedna instrukcja może odpowiadać kilkunastu rozkazom procesora

Języki programowania: Pascal, ANSI C/C++; Definicja: zbiór zasad składni, instrukcji, dzięki którym powstaje kod źródłowy programu

Kod źródłowy - program komputerowy zapisany w języku programowania.

Kod maszynowy - język zrozumiały przez procesor, składa się z ciągu wartości binarnych, które oznaczają jednocześnie instrukcje i dane.

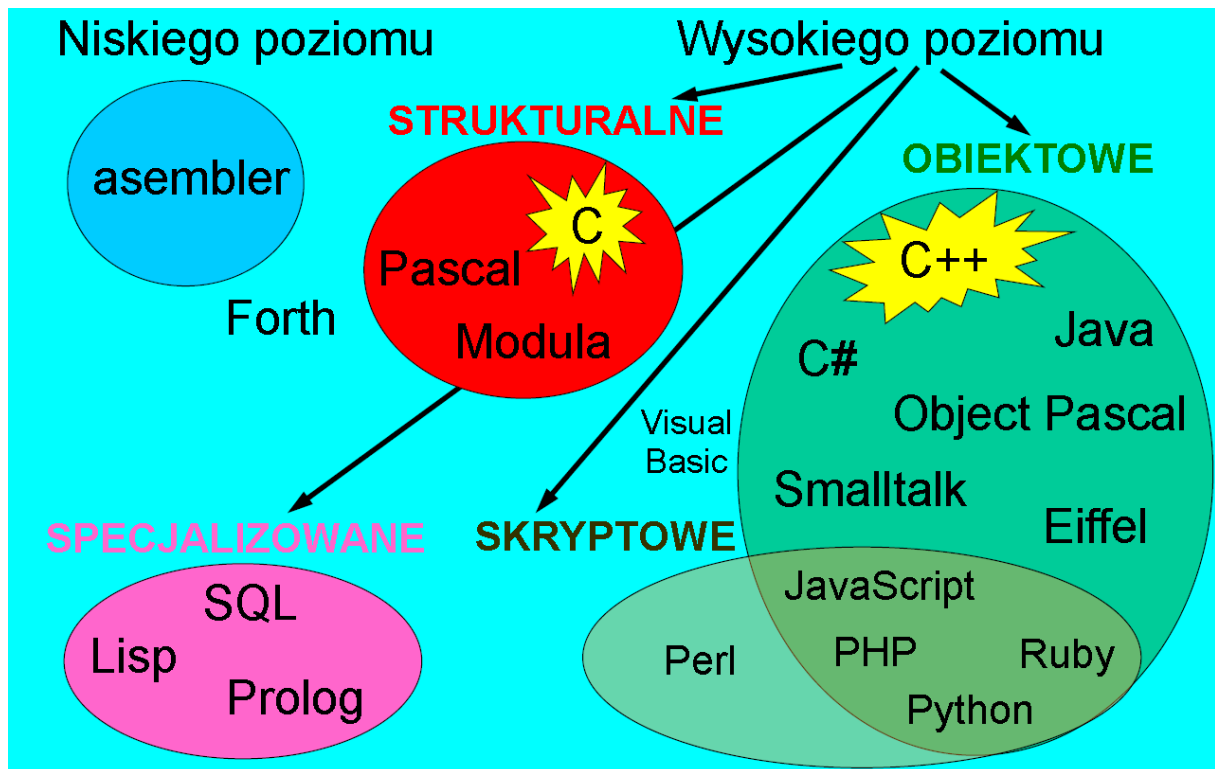
Rodzaje:

- strukturalne i obiektowe,
- kryterium związane z zastosowaniem

Niektóre języki są bardziej uniwersalne niż inne. Najpopularniejsze: C/C++, Java, Delphi.

Programowanie strukturalne - program jest podzielony na niewielkie moduły, bądź funkcje. Ułatwia projektowanie, testowanie a także utrzymywanie kodu programu. Przyk: Pascal, C

Programowanie obiektowe - dane i wykonywane na nich operacje są połączone. Umożliwia szybsze pisanie większych programów. Przyk: C++, Java



## Narzędzia Programisty

Translator/interpreter - tłumaczy kod na bieżąco

Kompilator - tłumaczy cały, potem rusza program

Program binarny - program przetłumaczony na język binarny

Desassembler - przetwarza kod z wersji binarnej na język assemblera

Dekompilacja - odtworzenie postaci źródłowej z kodu wynikowego

Program wynikowy - program przetłumaczony gotowy do wykonania

Konsolidacja - łączenie modułów prowadzące do utworzenia programu ładownego  
( statyczne - wykonywanie programu łączącego; dynamiczne - dołączanie modułów bibliotecznych)

Preprocesor - program wykonujący pierwszy przebieg analizowania pliku z kodem źródłowym, dokonuje wstępnych przekształceń w danych poddanych dalszej obróbce

System programowania - oprogramowanie, w którego skład wchodzi kompilator/interpreter, system wykonawczy i biblioteki

IDE - zespolone środowisko, połączenie programów do wytwarzania oprogramowania:  
edytor, kompilator, konsolidator(Linker)

RAD - metodologia polegająca na udostępnieniu programiście dużych możliwości prototypowania oraz dużego zestawu gotowych komponentów

API - polecenia dzięki którym użytkownik ma dostęp do funkcji [systemu operacyjnego](#), takich jak na przykład tworzenie [interfejsu graficznego](#).

## Typy danych

Zmienna - obiekt będący instancją danego typu, realizacją praktyczną – miejscem w pamięci komputera, którego zawartość interpretuje się wg typu zmiennej

Typ - formalny opis operacji i rodzaju informacji ( proste i złożone )

Typy proste:

- zmiennoprzecinkowe - float, double
- całkowitoliczbowe - bool, char, int, enum
- specjalne - void, wskaźnik, referencja

Typy złożone:

- struct - struktura
- union - unia, zajmuje ten sam obszar pamięci
- class - klasa
- tablica

Literały - dane zapisane w sposób dosłowny

Stała - zmienna o konkretnej wartości

Komentarze - //

## Zmienne

- void - pusty typ danych
- char - 8 bitowy, przechowuje znaki signed <-128;127>, unsigned <0,255>
- int - 16 bitów liczby całkowite
- float - 32 bity, zmiennoprzecinkowy
- double - 64 bity

Zmienna globalna ma domyślnie wartość 0.

### Operatory arytmetyczne

- przypisania:  $a = b$
- dodawania:  $a + b$
- odejmowania:  $a - b$
- mnożenia:  $a * b$
- dzielenia:  $a / b$
- modulo (reszta z dzielenia):  $a \% b$
- post i preinkrementacji:  $a++$  i  $++a$
- post i predekrementacji:  $a--$  i  $--a$

14.26.57

### Operatory porównania

- mniejsze:  $a < b$
- większe:  $a > b$
- mniejsze lub równe:  $a \leq b$
- większe lub równe:  $a \geq b$
- różne:  $a != b$
- równe:  $a == b$

14.26.57

### Operatory logiczne i bitowe

- logiczne
  - logiczna negacja:  $!a$
  - logiczne i:  $a \&\& b$
  - logiczne lub:  $a \|\ b$
- bitowe:
  - bitowa negacja:  $\sim a$
  - bitowe i:  $a \& b$
  - bitowe lub:  $a | b$
  - bitowe rozłączne lub (xor):  $a \wedge b$
  - przesunięcie bitowe w lewo:  $a \ll b$
  - przesunięcie bitowe w prawo:  $a \gg b$

### Operatory skrócone

- dodawanie:  $a += b$ 
  - $a = a + b$
- odejmowanie:  $a -= b$ 
  - $a = a - b$
- mnożenie:  $a *= b$ 
  - $a = a * b$
- dzielenie:  $a /= b$ 
  - $a = a / b$
- reszta z dzielenia:  $a \% = b$ 
  - $a = a \% b$

14.26.57

### Operatory skrócone

- bitowe i:  $a \&= b$ 
  - $a = a \& b$
- bitowe lub:  $a |= b$ 
  - $a = a | b$
- bitowe rozłączne lub:  $a \wedge = b$ 
  - $a = a \wedge b$
- bitowe przesunięcie w lewo:  $a \ll = b$ 
  - $a = a \ll b$
- bitowe przesunięcie w prawo:  $a \gg = b$ 
  - $a = a \gg b$

14.26.57

## Wykład 2

### Instrukcje

- printf()
- puts()
- scanf()
- getche()
- gets()

### Printf():

- %s znaki
- %d liczby całkowite
- %f float
- %x hex

### Instrukcje warunkowe if i if\_else

pętle: for, while, do\_while

wyboru: switch

### Operatory Logiczne:

Operator	Symbol	Przykład
I (AND)	&&	wyrażenie1 && wyrażenie2
LUB (OR)		wyrażenie1    wyrażenie2
NIE (NOT)	!	!wyrażenie

Continue - umożliwia powrót i wykonanie ponownie zestawu instrukcji

Break - Natychmiastowe wyjście z pętli i przejście do dalszych instrukcji

## Wykład 3

### Funkcje

Zmienne muszą być globalne.

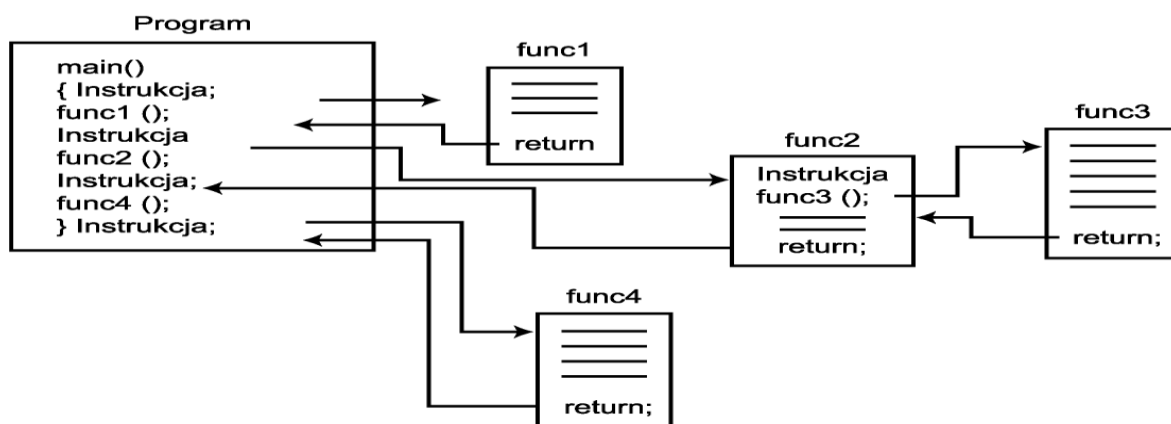
Dają możliwość podzielenia programu na mniejsze kawałki. Zwiększa to przejrzystość kodu.

Musimy ich używać, bo main sam w sobie jest funkcją.

Rodzaje:

- globalna - poza obiektami
- składowa - wewnątrz obiektów wykonując ich pracę

Każda funkcja ma nazwę. Gdy zostanie napotkana przez program, to przechodzi on do kodu zawartego w niej (wywołanie), po tym wraca do dalszych instrukcji.



Pierwszy wiersz funkcji: nazwa funkcji, typ zwracanej zmiennej, lista argumentów przekazywanych do funkcji.

Odmiany:

- wbudowane: stworzone przez producenta
- zdefiniowane: pisane samodzielnie

Deklaracja - informuje kompilator o nazwie funkcji, typie zwracanej przez nią wartości, oraz o jej parametrach.(prototyp)

Definicja - informuje, w jaki sposób dana funkcja działa.

Sposoby deklaracji:

- zapisanie prototypu w pliku, w którym dana funkcja jest używana,
- zdefiniowanie funkcji zanim zostanie wywołana przez inne funkcje. Jeśli tego dokonasz, definicja będzie pełnić jednocześnie rolę deklaracji funkcji,
- zapisanie prototypu funkcji w pliku, a następnie użycie dyrektywy `#include` w celu dołączenia go do swojego programu

### Prototyp funkcji

Wiele posiada gotowe prototypy, dołączone za pomocą #include. Gdy piszemy samodzielnie, samodzielnie musimy je dołączyć.

Definicja składa się z nagłowa (typ, nazwa, parametry) i ciała (między klamrami).

Parametry funkcji - opisuje typ wartości jaka jest przekazywana funkcji podczas wywołania.

### Argumenty funkcji

Static- Wartość zmiennej static pamiętana jest pomiędzy wywołaniami funkcji. Przy kolejnym wejściu do funkcji mamy do dyspozycji ostatnią wartość zmiennej static.

Extern - Jeżeli mamy program napisany w postaci 2 plików, a zmienna globalna zadeklarowana w jednym z nich ma być używana w drugim, to w drugim pliku jej deklaracja musi być poprzedzona słówkiem **extern**

Register - używamy gdy chcemy przyspieszyć szybkość wykonywania programu. informuje kompilator, że jeżeli to możliwe należy zapisać zmienną w rejestrach. Pojemność jest niewielka, więc nie zawsze jest to możliwe. Deklaracja register jest tylko sugestią dla kompilatora.

### Makroinstrukcje

tworzone za pomocą #define, NAZWA(PARAMETRY) ciąg\_instrukcji, nakazuje kompilatorowi zamianę stałej na ciąg instrukcji

### Argumenty MAIN() (argc i argv)

- argc podaje ilość argumentów wprowadzanych z wiersza poleceń
- Tablica argv jest tablicą wskaźników do typu char

element argv[0] zawiera zawsze nazwę programu, argc najmniej ma wartość 1

## Wykład 4

Tablica - zbiór danych tego samego typu, do którego można odwołać się przez wspólną nazwę. Rozróżniamy rodzaje:

- jednowymiarowe
- wielowymiarowe

typ\_danych: typ tablicy

nazwa\_tablicy: nazwa

lista\_rozmiarów: rozmiar tablicy

lista\_wartości: zbiór danych

Tablice mogą należeć do klas: automatic, static i external. Nie do register. Przy staticu jeżeli nie poda się wartości inicjalizujących kompilator zapełni tablicę zerami. Podobnie przy tablicy globalnej.

Własne typy (typedef)

wzór: typedef std\_nazwa\_typu własna\_nazwa\_typu

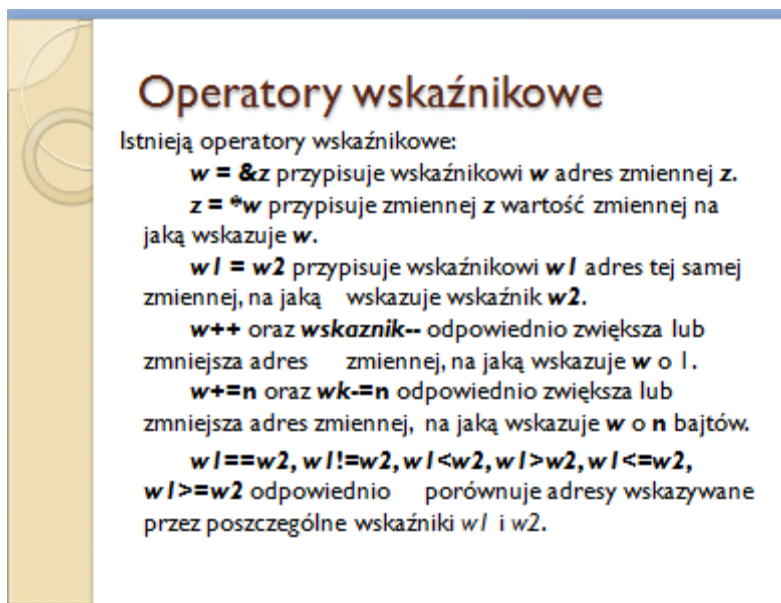
Struktura - zbiór danych różnych typów, dostępnych pod wspólną nazwą (struct)

Uzyskanie dostępu: zmienna.pole1

Struktury tych samych danych możemy wzajemnie przypisać.

Można deklarować tablice struktur.

Wskaźniki - adres obszaru pamięci komputera. typ \*nazwa; Możemy deklarować wskaźniki do wskaźników itd., ale jest to ryzykowne.



**Operatory wskaźnikowe**

Istnieją operatory wskaźnikowe:

- w = &z** przypisuje wskaźnikowi **w** adres zmiennej **z**.
- z = \*w** przypisuje zmiennej **z** wartość zmiennej na jaką wskazuje **w**.
- w1 = w2** przypisuje wskaźnikowi **w1** adres tej samej zmiennej, na jaką wskazuje wskaźnik **w2**.
- w++** oraz **wskaznik--** odpowiednio zwiększa lub zmniejsza adres zmiennej, na jaką wskazuje **w** o 1.
- w+=n** oraz **wk-=n** odpowiednio zwiększa lub zmniejsza adres zmiennej, na jaką wskazuje **w** o **n** bajtów.
- w1==w2, w1!=w2, w1<w2, w1>w2, w1<=w2, w1>=w2** odpowiednio porównuje adresy wskazywane przez poszczególne wskaźniki **w1** i **w2**.



Adres i rozmiar zmiennej

operator rozmiaru: `sizeof(nazwa_typu)`

operator adresu: `&`

(rozmiary zmiennych w bajtach: slajd 20-21 Wykład 4)

Odwoływanie przez wskaźnik:

`int x;` - deklaracja zmiennej

`int *px;` - deklaracja wskaźnika

Przypisywanie wskaźnika

1. `int *px1, *px2;`

2. `int x1;`

3. Przypisanie adresu zmiennej do wskaźnika: `px1 = &x1;`

4. Przypisanie wskaźnika: `px1=px2;`

Wskaźnik zajmuje 4 bajty.

Operacje na wskaźnikach: np. `p-1`; `p += 1`; `++p`

Dopuszczalne zapisy: jeśli `int *wsk;`

1. `wsk = 0`

2. `wsk = NULL` (równoważne z 1)

3. `wsk = &k;`

4. `wsk = (int *) 25550` (absolutny adres w pamięci)

Ograniczenia

- nie wskazywać na stałą (np. **`&7`** jest nielegalne)
- nie wskazywać na zwykłe wyrażenia (np. **`&(x - 100)`** jest nielegalne)
- nie wskazywać na zmienną rejestrową (np. **`register x; &x`** jest nielegalne)

Przekazywanie parametrów do funkcji:

- przez wartość (wcześniej było)

- przez wskaźnik

## Przekazywanie przez wskaźnik

Deklaracja funkcji ma postać:

```
void max(int,int,int*);
```

Definicja funkcji ma postać:

```
void max(int a,int b,int *v)  
{ *v = (a > b) ? a : b; }
```

Wywołanie tej funkcji ma postać:

```
int x,y,z;  
max(x,y,&z);
```

Odwołanie przez wskaźnik:

- parametr funkcji musi być zadeklarowany, jako wskaźnik,
- wewnątrz ciała funkcji musi nastąpić dereferencja wskaźnika,
- należy przekazać aktualny adres jako parametr funkcji.

Korzyści ze stosowania wskaźników:

- funkcja może zwrócić do środowiska wywołującego więcej niż jedną wartość.
- za pomocą wskaźników przekazywane są tablice i łańcuchy
- za pomocą wskaźników budowane są skomplikowane struktury danych
- za pomocą wskaźników można uzyskać użyteczne informacje techniczne (np. ilość wolnej pamięci).

## Wykład 5

Unia - struktura, której wszystkie składowe umieszczane są w tym samym obszarze pamięci. Deklaracja podobna do deklaracji struktury. Zostaje jej przydzielona taka ilość pamięci, aby mogła pomieścić największy typ zmiennej wchodzącej w skład uni.

### Definicja zmiennej

- union test
 

```
{
            long x1;
            unsigned char t[4];
        } zm1, zm2;
```
- union test zm1, zm2;

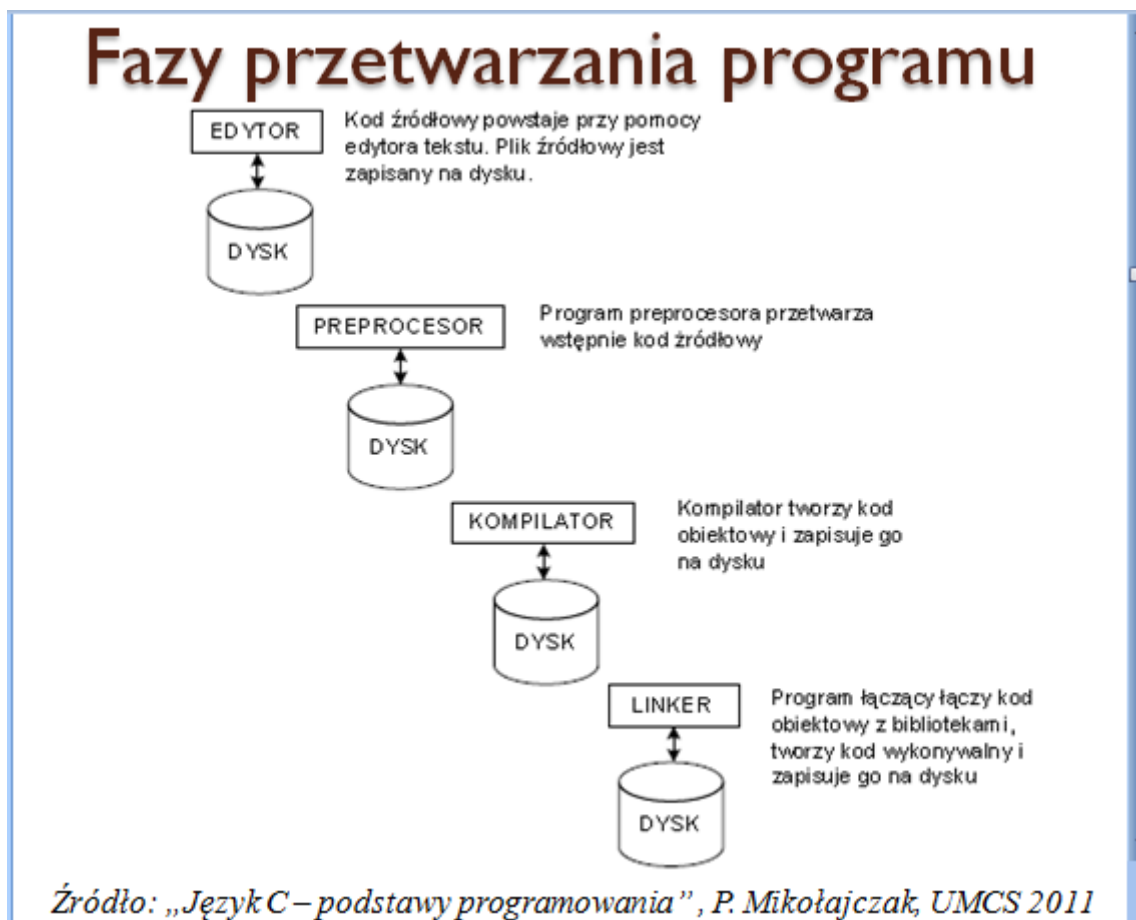
### Inicjowanie unii

```
union test
{
    int x1;
    unsigned char t[4];
} ut;
```

```
union test
{
    int x1;
    unsigned char t[4];
} ut={123};
```

```
int main(){
    ut.x1=123;
```

```
int main(){
    }
```



Dyrektywy procesora:

- Odpowiadają za wstępne przetwarzanie programu,
- Dołączanie plików,
- Zamiana skrótów symbolicznych na ich definicje;
- Kod warunkowy.

#include:

- #include <conio.h>
- #include "plik2.h "
- #include "c:\temp\plik3.h"

#define

- Tworzy stałe symboliczne
- Tworzy makroinstrukcje

Przykłady:

- #define PI 3.14
- #define roz\_tab 100
- #define MAX(x,y) ((x)>(y)?(x):(y))

#ifdef

- Można realizować kompilację warunkową,
- Używamy:

#ifdef / #ifndef

#else

#endif

## #ifdef

- Można realizować kompilację warunkową,
- Używamy:

#ifdef / #ifndef

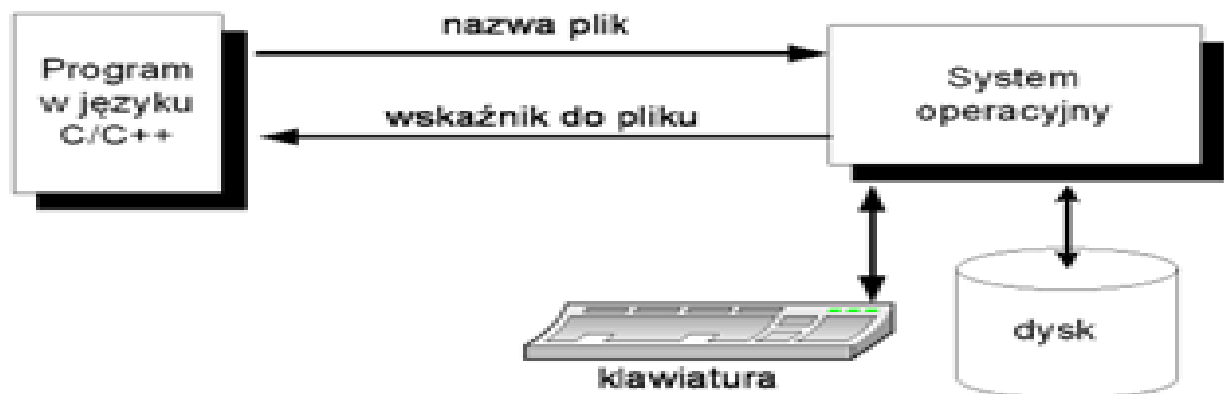
#else

#endif

Pliki:

- Umożliwiają przechowywanie danych na dysku;
- Dostęp przez wskaźnik zdefiniowany w stdio.h;
- Plik może zostać otwarty w trybie tekstowym lub binarnym

## System plików



Zapisywanie do pliku:

## Tryb tekstowy-przykład

```
#include <stdio.h>
#include <conio.h>
int main()
{
    FILE *f;
    char zn;
    f = fopen ("test35.txt","w");
    printf("Wprowadz tekst :\n");
    while ( (zn=getche()) != '\r')
        putc(zn,f);
    fclose(f);
    return 0;
}
```

## Co musimy zapamiętać

f = fopen ("test35.txt","w");

Nazwa pliku wraz  
ze ścieżką

Tryb otwarcia pliku

Parametry otwarcia pliku:

Tryb	Opis
"r"	Otwiera plik tekstowy do czytania
"w"	Otwiera plik tekstowy do zapisu. Jeżeli plik istnieje, usuwa zawartość i umieszcza nowe dane. Jeżeli plik nie istnieje, zostaje utworzony.
"a"	Otwiera plik do zapisu. Jeżeli plik istnieje, dopisuje nowe dane na końcu istniejących danych. Jeżeli plik nie istnieje zostaje utworzony.
"r+"	Otwiera plik tekstowy do uaktualnienia, zezwala na zapis i czytanie
"w+"	Otwiera plik tekstowy do uaktualnienia, zezwala na zapis i czytanie. Jeżeli plik istnieje, usuwa zawartość. Jeżeli plik nie istnieje jest tworzony.
"a+"	Otwiera plik tekstowy do uaktualnienia, zezwala na zapis i czytanie. Jeżeli plik istnieje, dopisuje nowe dane na końcu. Jeżeli plik nie istnieje jest tworzony. Odczyt obejmuje cały plik, zapis polega na dodawaniu nowego tekstu.
"rb", "wb", "ab", "rb+", "r+b", "wb+", "w+b", "ab+", "a+b"	Wymienione specyfikatory mają takie samo znaczenie jak powyższe, dotyczą <b>plików binarnych</b>

Kontrola otwarcia pliku:

Od funkcji fopen otrzymujemy wskaźnik na plik, czyli wystarczy sprawdzić czy mamy "prawidłowy" wskaźnik.

```
f = fopen("test35.txt","r");
if ( f == NULL) ←
{
    printf("\n Nie moge otworzyc pliku");
    exit(1);
}else{
    //operacje na pliku
    fclose(f);
}
```

Wprowadzanie łańcuchów	Czytanie łańcuchów w pliku
<pre>f = fopen("test35.txt","a"); char napis[100]; while (strlen ( gets(napis)) &gt; 0 ) {     fputs(napis,f);     fputs("\n",f); }</pre>	<pre>while (fgets ( napis, 80, f) != NULL )     printf("%s", napis);</pre>

Zapis i odczyt do pliku w trybie binarnym

- `size_t fread(void *ptr, size_t size, size_t n, FILE *stream);`
- `size_t fwrite(void *ptr, size_t size, size_t n, FILE`

Wskaźnik na miejsce z danymi

- `size_t fread(void *ptr, size_t size, size_t n, FILE *stream);`

Długość jednostkowa pozycji danych np. `sizeof(int)`

- `size_t fwrite(void *ptr, size_t size, size_t n, FILE`

- `size_t fread(void *ptr, size_t size, size_t n, FILE *stream);`

Ilość  
danych

- `size_t fwrite(void *ptr, size_t size, size_t n, FILE *stream);`

- `size_t fread(void *ptr, size_t size, size_t n, FILE *stream);`

Wskaźnik  
na plik

- `size_t fwrite(void *ptr, size_t size, size_t n, FILE *stream);`