

Podstawy Programowania II

Prof. dr hab. inż. **Anna Fabijańska**

Instytut Informatyki Stosowanej PŁ

Plan wykładu (1)

- 1. Obsługa plików dyskowych (strumienie i uchwyty).**
- 2. Struktury, unie, wyliczenia.**
- 3. Przydział pamięci i tablice dynamiczne 1D oraz 2D.**
- 4. Manipulacje na łańcuchach i blokach pamięci.**
- 5. Klasy pamięci, konsolidator.**
- 6. Modyfikatory: auto, static, extern, const, volatile, register, typedef.**
- 7. Wskaźniki do funkcji**

Plan wykładu (2)

- 11. Operator warunkowy, sizeof i przecinka.**
- 12. Funkcje konwersji i klasyfikacji.**
- 13. Algorytmy iteracyjne i rekurencyjne.**
- 14. Argumenty funkcji main().**
- 15. Funkcje ze zmienną liczbą argumentów.**
- 16. Wprowadzenie do list, kolejek i drzew.**
- 17. Zmienne środowiskowe i systemowe.**
- 18. Podstawy programowania hybrydowego.**

Zaliczenie przedmiotu

▶ **Zaliczenie laboratorium**

- ▶ dwa kolokwia sprawdzające umiejętność programowania

▶ **Zaliczenie prac domowych (e-learning)**

- ▶ wykonanych 15 punktów w każdym z 10 działów w systemie Dante (w tym dwa zadania obowiązkowe)

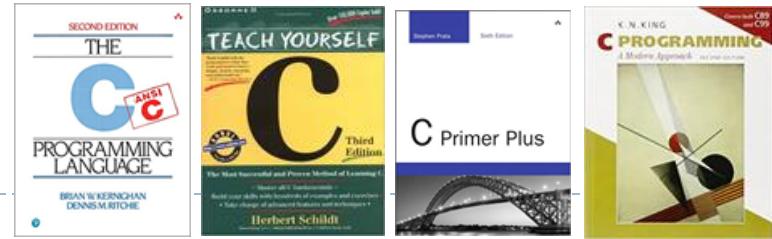
▶ **Egzamin pisemny (część wykładowa)**

- ▶ warunkiem dopuszczenia do egzaminu jest zaliczenie laboratorium oraz projektu (potwierdzone wpisem do EKS)

Ocena koncowa

*= 0,35 oceny z lab. + 0,35 oceny z wykładu
+ 0,1 oceny z e – learningu
+ 0,2 oceny z projektu*

Literatura



▶ Literatura podstawowa:

- ▶ Kernighan B.W., Ritchie D. M. : *Język ANSI C. Programowanie.* Helion, Gliwice, 2010
- ▶ Drozdek A. , Simon D. L.: *Struktury danych w języku C.* WNT, Warszawa, 1996

▶ Literatura uzupełniająca:

- ▶ Schildt H.: *Język C.* LTP, Warszawa, 2002
- ▶ Prata S.: *Szkoła programowania. Język C.* Wyd. IV, Helion, Gliwice, 2016
- ▶ King K.N.: *C Programming – A Modern Approach, 2nd Edition,* W. W. Norton & Company, 2009
- ▶ Wirth N. : *Algorytmy + Struktury Danych = Programy.* WNT, Warszawa, 2004

Dostęp do plików

- ▶ **Strumień** – poziom pośredni pomiędzy programistą a sprzętem, zapewniający programistie spójny interfejs we/wy niezależny od rzeczywistych urządzeń we/wy
- ▶ **Plik** – rzeczywiste urządzenie we/wy (może reprezentować wiele typów urządzeń)
- ▶ Najpopularniejszy typ pliku to **plik dyskowy**
- ▶ Strumień wiązany jest z plikiem za pomocą **operacji otwarcia** i odłączany od pliku za pomocą **operacji zamknięcia**

Dostęp do plików

- ▶ Funkcje pozwalające na dostęp do plików dostępne są w pliku nagłówkowym `<stdio.h>`
- ▶ Każdy plik zakończony jest specjalnym znakiem **EOF** (ang. *End of File*)
- ▶ **Wskaźnik pliku** (ang. *File Pointer*) to adres struktury, która zawiera informacje o pliku

```
FILE *f ;
```

- ▶ Bieżące położenie (pozycja) – miejsce, od którego zacznie się następna operacja dostępu.

1. Otworzyć plik w odpowiednim trybie

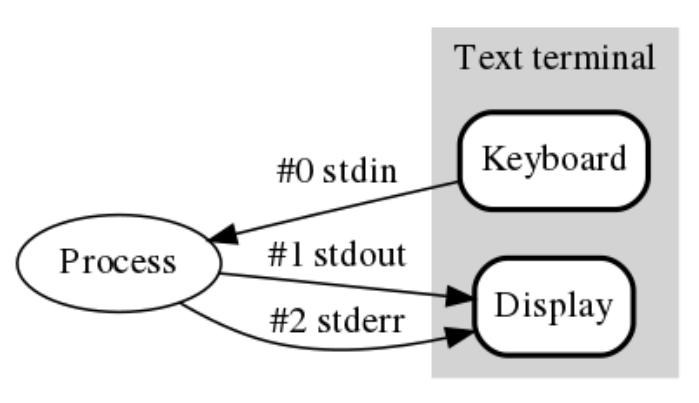
2. Wykonać operację na pliku

3. Zamknąć plik

Dostęp do plików

- ▶ **Trzy standardowe strumienie I/O** (gotowe do użycia, bez konieczności otwierania/zamykania)

Wskaźnik pliku	Strumień	Znaczenie
stdin	standardowy strumień wejścia	klawiatura
stdout	standardowy strumień wyjścia	monitor
stderr	standardowy strumień błędów	monitor



Pliki binarne vs. pliki tekstowe

- ▶ Dwa typy strumieni:
 - ▶ **tekstowe** – zawiera znaki kodu ASCII, zachodzi tłumaczenie
 - ▶ dane dzielone na linie, zakończone znakiem końca linii
 - ▶ **binarne** – brak tłumaczenia – dane w pliku odpowiadają danym w strumieniu
 - ▶ nie muszą zawierać danych tekstowych, ale np. struktury, liczby, etc.

liczba: 32767

00110011	00110010	00110111	00110110	00110111
'3'	'2'	'7'	'6'	'7'

w pliku tekstowym

00110011	00110010
32767	

w pliku binarnym
(lub w odwrotnej kolejności)

Podstawy systemu plików

▶ Otwieranie pliku

FILE *fopen(char* nazwa_pliku, char* tryb);

- ▶ zwracany jest wskaźnik do struktury związanej z plikiem
- ▶ nie można zmienić wskaźnika, ani obiektu na który wskazuje
- ▶ w przypadku niepowodzenia zwracany jest NULL

Tryb	Znaczenie
r	otwiera plik do czytania
r+	otwiera plik do czytania i nadpisywania (aktualizacja)
w	otwiera plik do nadpisywania (zamazuje starą treść)
w+	otwiera plik do nadpisywania i czytania
a	otwiera plik do dopisywania (jeśli nie istnieje, to jest tworzony)
a+	otwiera plik do dopisywania i odczytu (jeśli nie istnieje, jest tworzony)
t	otwiera plik w trybie tekstowym
b	otwiera plik w trybie binarnym

Podstawy systemu plików

▶ Otwieranie pliku

```
FILE *fp;  
fp = fopen("plik.txt", "r");  
  
if (fp==NULL)  
{  
    printf("Błąd otwarcia pliku\n");  
    exit(1); /*lub obsługa błędu*/  
}
```

▶ Ścieżka dostępu (Win. OS)

```
fp = fopen("D:\PP2\Wyklad\Nowy\TenSemestr.pptx", "rb");  
  
fp = fopen("D:\\PP2\\Wyklad\\Nowy\\TenSemestr.pptx", "rb");  
  
fp = fopen("D:/PP2/Wyklad/Nowy/TenSemestr.pptx", "rb");
```

Podstawy systemu plików

▶ Zamykanie pliku

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

- ▶ zamknięcie pliku związanego ze wskaźnikiem fp
- ▶ odłączenie strumienia od pliku
- ▶ w przypadku powodzenia zwracane jest 0
- ▶ w przypadku błędu zwracany jest EOF

Podstawy systemu plików

▶ Czytanie i pisanie bajtów (znaków)

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

- ▶ wczytanie kolejnego bajtu z pliku określonego przez *fp*
- ▶ znak zwracany jest w młodszych bajtach liczby
- ▶ w przypadku błędu lub napotkania końca pliku zwracany jest EOF

```
int fputc(int ch, FILE *fp);
```

- ▶ wpisanie młodszego bajta zmiennej *ch* do pliku związanego z *fp*

```
int ungetc(int ch, FILE *fp);
```

- ▶ Zwraca znak *ch* do strumienia związanego z *fp*

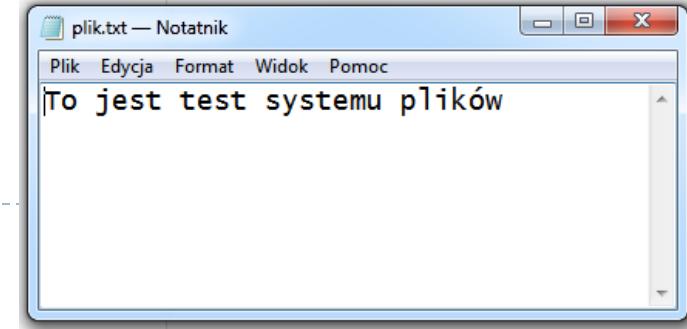
Przykład 1

```
#include <stdio.h>
#include <stdlib.h>
/*zapis do pliku*/
int main(void) {
    char nap[80] = "To jest test systemu plików\n";
    char *p = nap;
    FILE *fp;
    int i;

    if((fp=fopen("plik.txt","w"))==NULL) {
        printf("Błąd otwarcia pliku.\n");
        exit(1);
    }

    while(*p) {
        if(fputc(*p,fp)==EOF) {
            printf("Błąd zapisu pliku.\n");
            exit(1);
        }
        p++;
    }

    fclose(fp);
    return 0;
}
```



Przykład 2

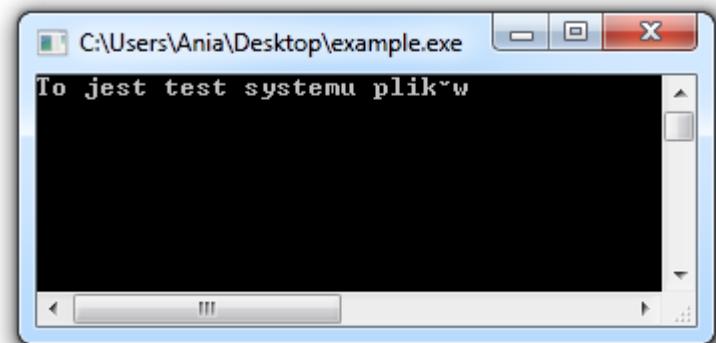
```
#include <stdio.h>
#include <stdlib.h>
/*odczyt pliku*/
int main(void) {
FILE *fp;
int i; /*lub char i*/

if((fp=fopen("plik.txt","r"))==NULL) {
printf("Błąd otwarcia pliku.\n");
exit(1);
}

for(;;) {
i=fgetc(fp);
if(i==EOF) break;
putchar(i);
}

fclose(fp);

return 0;
}
```



Przykład 3a

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>

int main (void)
{
    FILE* fp = fopen("plik.txt", "rb");
    int ch;

    if(!fp) {
        printf("error");
        exit(0);
    }

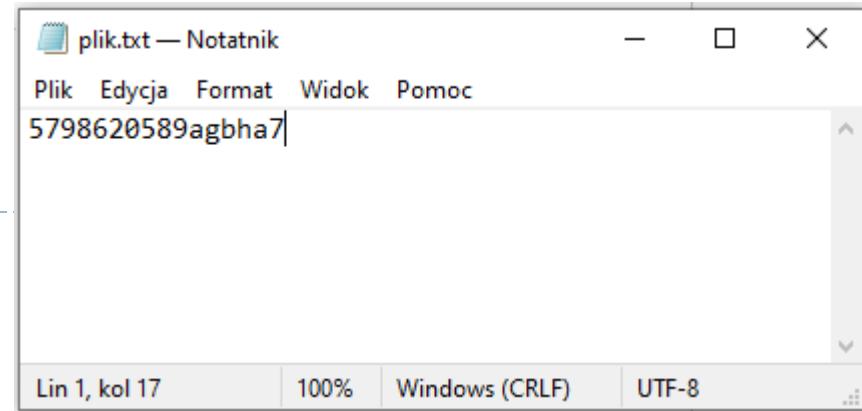
    while (isdigit(ch = fgetc(fp)))
        putchar(ch);

    ungetc(ch,fp); //oddaj znak do pliku
    printf("\n oddanie znaku do pliku\n");

    while (isalpha(ch = fgetc(fp)))
        putchar(ch);

    fclose(fp);

    return 0;
}
```



```
5798620589
oddanie znaku do pliku
agbha
Process finished with exit code 0
```

Przykład 3b

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>

int main (void)
{
    FILE* fp = fopen("plik.txt", "rb");
    int ch;

    if(!fp) {
        printf("error");
        exit(0);
    }

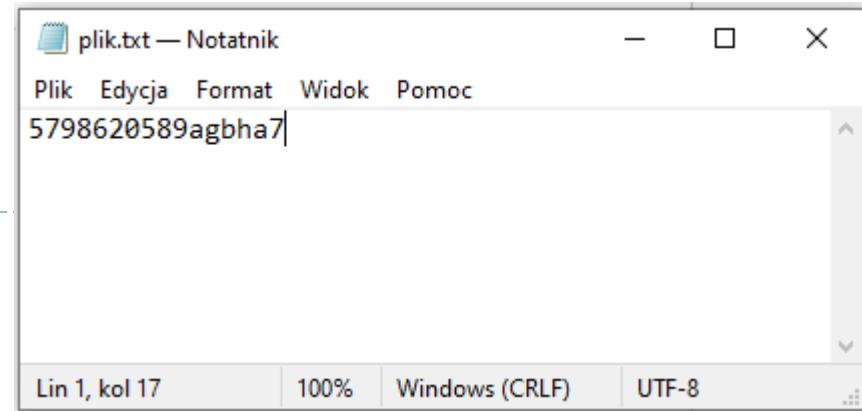
    while (isdigit(ch = fgetc(fp)))
        putchar(ch);

    printf("\n bez oddania znaku do pliku\n");

    while (isalpha(ch = fgetc(fp)))
        putchar(ch);

    fclose(fp);

    return 0;
}
```



A screenshot of a Windows Notepad window titled 'plik.txt — Notatnik'. The window contains the text '5798620589agbha7'. Below the window, a terminal window shows the output of the program: '5798620589', 'bez oddania znaku do pliku', 'gbha', and 'Process finished with exit code 0'. The terminal window has tabs at the bottom: 'Lin 1, kol 17', '100%', 'Windows (CRLF)', and 'UTF-8'.

Funkcje feof() oraz ferror()

▶ Wykrywanie końca pliku

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

- ▶ zwrócenie wartości niezerowej, gdy w pliku ze wskaźnikiem osiągnięty został koniec
- ▶ w przeciwnym przypadku zwracane jest 0

▶ Wykrywanie błędów

```
int ferror(FILE *fp);
```

- ▶ zwrócenie statusu systemu plików związanego z ostatnią operacją dostępu do pliku
- ▶ aby zapewnić pełne sprawdzanie błędów należy ją wywołać po każdej operacji na pliku

Przykład 4

```
FILE *skad, *dokad;
char *zrodlo = "plik.txt", *cel = "plik2.txt";
char ch;

if((skad=fopen(zrodlo,"rb"))==NULL){
    fprintf(stderr, "Błąd otwarcia pliku źródłowego.\n"); exit(1);
}

if((dokad=fopen(cel,"wb")) ==NULL){
    fprintf(stderr, „Błąd otwarcia pliku docelowego.\n"); exit(1);
}

while(!feof(skad)){
    ch = fgetc(skad);
    if(ferror(skad)){
        fprintf(stderr, "Błąd odczytu pliku źródłowego\n"); exit(1);
    }
    if(!feof(skad)) fputc(ch, dokad);

    if(ferror(dokad)){
        fprintf(stderr, "Błąd zapisu pliku docelowego\n"); exit(1);
    }
}

if(fclose(skad)==EOF){
    fprintf(stderr, "Błąd zamknięcia pliku źródłowego.\n"); exit(1);
}

if(fclose(dokad)==EOF){
    fprintf(stderr, "Błąd zamknięcia pliku docelowego.\n"); exit(1);
}
```

Funkcje tekstowe

▶ Zapis do pliku

```
int fputs(char *nap, FILE *fp);
```

- ▶ zapis ciągu znaków wskazywanego przez *nap* do pliku powiązanego z *fp*
- ▶ zwraca EOF przy niepowodzeniu oraz wartość dodatnią przy sukcesie

```
int fprintf(FILE *fp, char *napis_sterujący, ...);
```

- ▶ działanie analogiczne do *printf()* z tym, że pracuje na plikach
- ▶ dokonuje konwersji danych

Funkcje tekstowe

▶ Odczyt z pliku

```
char *fgets(char *nap, int ile, FILE *fp);
```

- ▶ wczytanie znaków z pliku powiązanego z *fp* do napisu wskazywanego przez *nap* do momentu wczytania *ile*-l znaków lub napotkania znaku nowej linii lub końca pliku

```
int fscanf(FILE *fp, char *napis_sterujący, ...);
```

- ▶ działanie analogiczne do *scanf()* z tym, że pracuje na plikach
- ▶ dokonuje konwersji danych

Przykład 5

```
#include <stdio.h>          /*program pobiera dane z klawiatury*/
#include <stdlib.h>          /*i zapisuje do pliku aż do napotkania pustej linii*/
#include <string.h>

int main(void) {
    FILE *fp;
    char nap[80], ch;

    if((fp=fopen("plik3.txt","w"))==NULL) {
        printf("Błąd otwarcia pliku.\n");
        exit(1);
    }

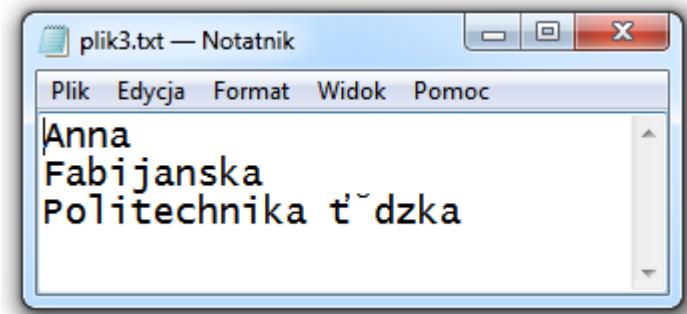
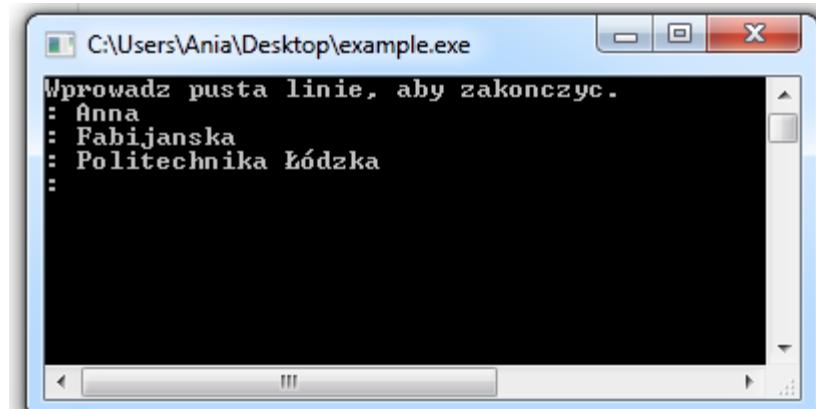
    printf("Wprowadź pustą linię, aby zakończyć.\n");

    do{
        printf(": ");
        fgets(nap,79,stdin);
        strcat(nap,"\n");           /*znak nowej linii*/

        if(*nap!='\n') fputs(nap,fp);

    }while(*nap!='\n');

    fclose(fp);
    return 0;
}
```



Przykład 6

```
#include <stdio.h>          /*program odczytuje kolejne linie z pliku*/
#include <stdlib.h>          /*i drukuje na ekranie*/
#include <string.h>

int main(void) {
    FILE *fp;
    char nap[80];

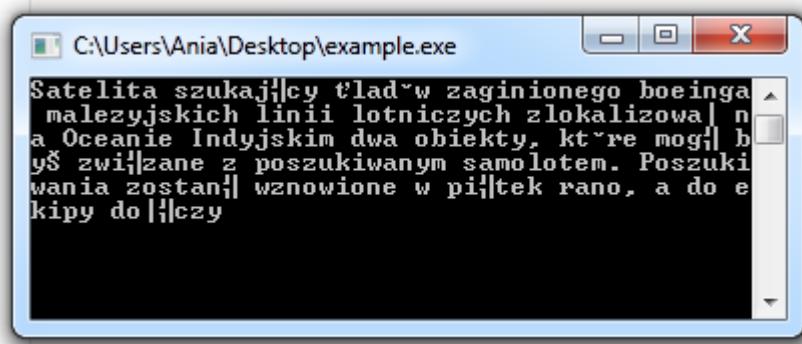
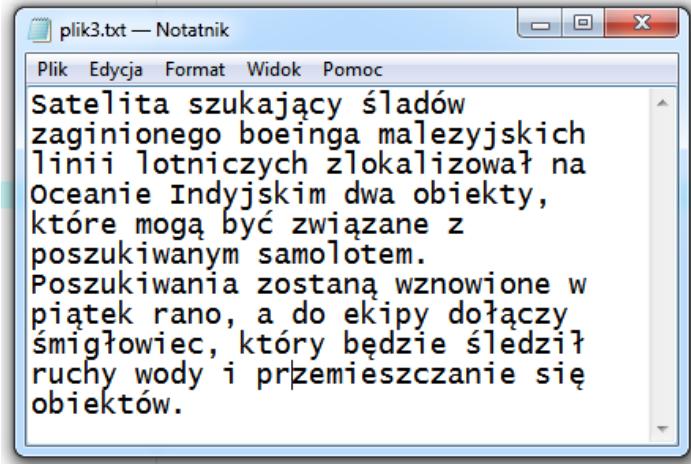
    if((fp=fopen("plik3.txt","r"))==NULL) {
        printf("Błąd otwarcia pliku.\n");
        exit(1);
    }

    do{
        fgets(nap, 80, fp);
        if(!feof(fp)) printf(nap);
    }while(!feof(fp));

    fclose(fp);

    return 0;
}
```

```
/*program odczytuje kolejne linie z pliku*/
/*i drukuje na ekranie*/
```



Przykład 7

```
#include <stdio.h>          /*program zapisuje dane do pliku*/
#include <stdlib.h>          /*w określonym formacie*/
#include <string.h>

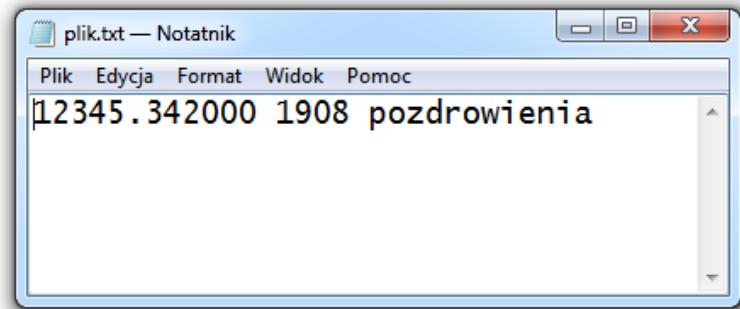
int main(void)
{
    FILE *fp;

    if((fp=fopen("plik.txt","w"))==NULL) {
        printf("Błąd otwarcia pliku.\n");
        exit(1);
    }

    fprintf(fp, "%lf %d %s", 12345.342, 1908, "pozdrowienia");

    fclose(fp);

    return 0;
}
```



Przykład 8

```
#include <stdio.h>          /*program odczytuje dane do pliku*/
#include <stdlib.h>          /*w określonym formacie*/
#include <string.h>

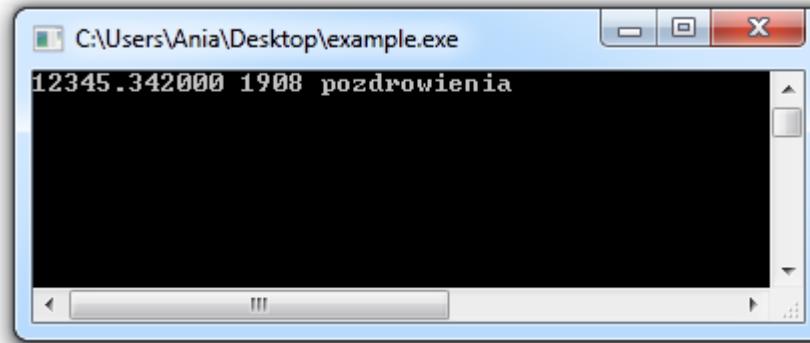
int main(void)
{
    FILE *fp;
    char nap[80];
    double ld;
    int d;

    if((fp=fopen("plik.txt","r"))==NULL) {
        printf("Błąd otwarcia pliku.\n");
        exit(1);
    }

    fscanf(fp, "%lf %d %s", &ld, &d, nap);
    printf("%lf %d %s", ld, d, nap);

    fclose(fp);

    return 0;
}
```



Przykład 9

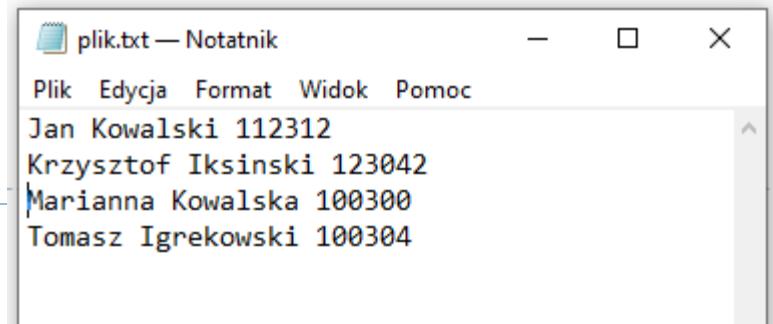
```
#include <stdio.h>

int main (void) {
    char imie[10], nazwisko[20];
    int album;

    FILE* fp = fopen("plik.txt","rt") ;

    if(fp) {
        while(!feof(fp)) {
            fscanf(fp, "%9s %19s %d\n", imie, nazwisko, &album);
            printf("%10s\t%20s\t%d\n", imie, nazwisko, album);
        }
        fclose(fp);
    }

    return 0;
}
```



The screenshot shows a Windows Notepad window titled "plik.txt — Notatnik". The file contains four lines of data: "Jan Kowalski 112312", "Krzysztof Iksinski 123042", "Marianna Kowalska 100300", and "Tomasz Igrekowski 100304". Below the notepad is a terminal window showing the output of the program. The terminal window has tabs at the bottom: "Lin 3, kol 1", "100%", "Windows (CRLF)", and "ANSI". The output consists of four lines of data, each with three columns: name, surname, and album number. The names and surnames are aligned under their respective column headers, while the album numbers are aligned under the third column header. At the bottom of the terminal window, the message "Process finished with exit code 0" is displayed.

Jan	Kowalski	112312
Krzysztof	Iksinski	123042
Marianna	Kowalska	100300
Tomasz	Igrekowski	100304

Process finished with exit code 0

Przykład 10

```
#include <stdio.h>

int main (){

    FILE* f = fopen("in.txt", "r");

    if(f){

        while(!feof(f)){

            int age, check;

            char line[100]={}, name[20]={}, surname[20]={};

            fgets(line, 99, f);

            check = sscanf(line, "Imie: %s\tNazwisko: %s\tWiek: %d\n", name, surname, &age);

            if (check!=3)

                continue;

            printf("Imie: %s\tNazwisko: %s\tWiek: %d\n", name, surname, age);

        }

        fclose(f);
    }

    return 0;
}
```

The screenshot shows a Notepad window titled '*in.txt — Notatnik'. The file contains the following text:

Imie:	Nazwisko:	Wiek:
Jan	Kowalski	19
Maria	Iksinska	22
jakas tam linia		
Tomasz	Malinowski	39

At the bottom of the window, there are status bars: 'Lin 4, kol 44', '100%', 'Windows (CRLF)', and 'UTF-8'.

The screenshot shows a terminal window titled 'C:\Users\an_fab\Desktop\test.exe'. The output of the program is:

```
Imie: Jan      Nazwisko: Kowalski      Wiek: 19
Imie: Maria    Nazwisko: Iksinska     Wiek: 22
Imie: Tomasz   Nazwisko: Malinowski   Wiek: 39
-----
Process exited after 0.01143 seconds with return value 0
Press any key to continue . . .
```

Operacje na danych binarnych

- ▶ Odczyt danych z wykorzystaniem ich reprezentacji binarnej

size_t fread (void *bufor, size_t rozmiar, size_t ilość, FILE *fp)

- ▶ wczytanie z pliku powiązanego z *fp* obiektów w liczbie *ilość*, gdzie każdy ma wielkość *rozmiar*, do bufora wskazywanego przez *bufor*
- ▶ zwrócenie liczby faktycznie przeczytanych obiektów (liczba mniejsza od *rozmiar* oznacza błąd lub wcześniejszy koniec pliku)
- ▶ plik musi być otwarty w trybie binarnym

wskaźnik void*

może pokazywać na dowolny typ danych

size_t (unsigned lub unsigned long)
typ pozwalający na przechowanie wielkości
równiej rozmiarowi największego obiektu
dopuszczanego przez kompilator

Wskaźnik void

- ▶ Wskaźnik do danych nieokreślonego typu.

```
void* wsk;
```

- ▶ Może pokazywać na dane dowolnego typu oraz można mu przypisać wskaźnik dowolnego typu.

```
void *wskv;  
int *wski;  
float *wskf  
  
wskv = wski; // operacja poprawna  
wskv = wskf; // operacja poprawna
```

- ▶ Aby przypisać wskaźnik void do wskaźnika innego typu, należy dokonać **rzutowania typów**

```
wski = wskv;           // BŁĄD !!  
wski = (int*)wskv;     // operacja poprawna  
wskf = (float*)wskv;   // operacja poprawna
```

Przykład 11

```
#include <stdio.h>

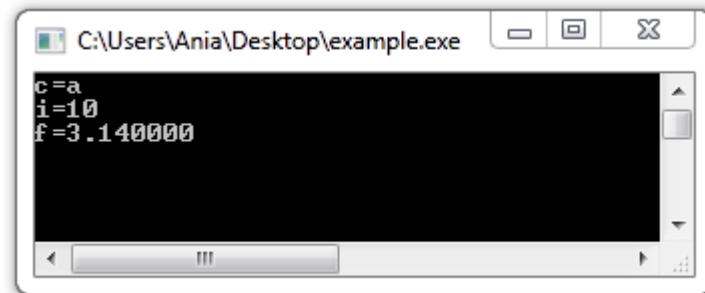
int main() {
    void *wskv;
    char c = 'a';
    int i=10;
    float f=3.14;

    wskv = &c;
    printf("c=%c\n", *(char*)wskv);

    wskv = &i;
    printf("i=%d\n", *(int*)wskv);

    wskv = &f;
    printf("f=%f\n", *(float*)wskv);

    return 0;
}
```



rzutowanie typów

wskv = &c;
printf("c=%c\n", *(char*)wskv);

wskv = &i;
printf("i=%d\n", *(int*)wskv);

wskv = &f;
printf("f=%f\n", *(float*)wskv);

Operacje na danych binarnych

- ▶ Zapis danych z wykorzystaniem ich reprezentacji binarnej

```
size_t fwrite (void *bufor, size_t rozmiar, size_t ilość, FILE *fp)
```

- ▶ zapisanie do pliku powiązanego z *fp* obiektów w liczbie *ilosc*, gdzie każdy ma wielkość *rozmiar*, do bufora wskazywanego przez *bufor*
- ▶ zwrócenie liczby faktycznie zapisanych obiektów (liczba mniejsza od *rozmiar* oznacza błąd)
- ▶ plik musi być otwarty w trybie binarnym

Przykład 12

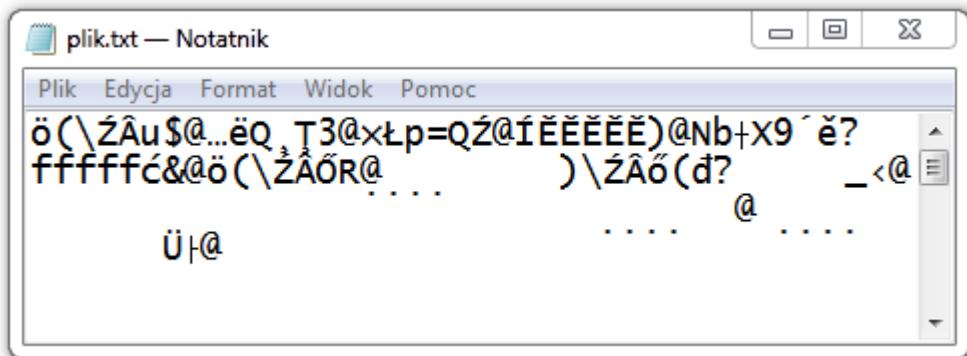
```
#include <stdio.h>
#include <stdlib.h>

int main(void) {
    int i;
    FILE *fp;
    double d[10] = {10.23, 19.87, 1002.13, 12.9, 0.897, 11.45, 75.34,
0.0, 1.01, 875.875};

    if((fp=fopen("plik.txt","wb"))==NULL) {
        printf("Błąd otwarcia pliku.\n");
        exit(1);
    }

    if(fwrite(d, sizeof d, 1, fp)!=1) {
        printf("Błąd zapisu");
        exit(1);
    }

    fclose(fp);
    return 0;
}
```



Przykład 13

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

int main(void){
    int i;
    FILE *fp;
    double d[10];

    if((fp=fopen("plik.txt","rb"))==NULL) {
        printf("Błąd otwarcia pliku.\n");
        exit(1);
    }

    if(fread(d, sizeof d, 1, fp)!=1) {
        printf("Błąd odczytu");
        exit(1);
    }

    fclose(fp);

    for (i=0; i<10; i++) {
        printf("%lf\t", *(d+i));
    }

    return 0;
}
```



Dostęp swobodny

▶ Odczyt dowolnego miejsca pliku w dowolnym momencie

int fseek (FILE *fp, long offset , int początek)

- ▶ przesunięcie wewnątrz pliku związanego ze wskaźnikiem *fp* o *offset* bajtów od miejsca *początek*
- ▶ wartości parametru *początek*:
 - ▶ SEEK_SET - początek pliku
 - ▶ SEEK_CUR - bieżąca pozycja
 - ▶ SEEK_END - koniec pliku
- ▶ wartość zwracana: 0 w przypadku powodzenia oraz wartość niezerowa w przypadku błędu
- ▶ można przesunąć bieżącą pozycję poza koniec pliku, lecz nigdy przed początek

Dostęp swobodny

- ▶ Określanie bieżącej pozycji w pliku

long ftell (FILE *fp)

- ▶ zwrócenie bieżącej pozycji w pliku związanym z *fp* lub -l w przypadku błędu

int fgetpos(FILE * pf, fpos_t * pos)

- ▶ pobranie aktualnej pozycji kurSORA odczytu/zapisu danych dla wskazanego strumienia do zmiennej wskazywanej poprzez drugi argument

int fsetpos(FILE * pf, const fpos_t * pos)

- ▶ ustawienie pozycji kurSORA odczytu/zapisu danych dla wskazanego strumienia na pozycję określoną przez drugi argument funkcji.

Przykład 14

```
#include <stdio.h> /*kopiowanie pliku w odwrotnej kolejności*/
#include <stdlib.h>

int main(void) {
    long polozenie;
    FILE *zrodlo, *cel;
    char ch;

    if((zrodlo=fopen("zrodlo.txt","rb"))==NULL) {
        printf("Błąd otwarcia pliku zrodłowego.\n"); exit(1);
    }

    if((cel=fopen("cel.txt","wb"))==NULL) {
        printf("Błąd otwarcia pliku docelowego.\n"); exit(1);
    }

    fseek(zrodlo, 0L, SEEK_END);
    polozenie = ftell(zrodlo);

    polozenie = polozenie - 1; /*pominięcie końca pliku*/

    while(polozenie >= 0L) {
        fseek(zrodlo, polozenie, SEEK_SET);
        ch = fgetc(zrodlo);
        fputc(ch, cel);
        polozenie--;
    }

    fclose(zrodlo);
    fclose(cel);

    return 0;
}
```

Inne funkcje systemu plików

- ▶ Zmiana nazwy pliku

```
int rename (char* stara_nazwa, char* nowa_nazwa);
```

- ▶ Usunięcie pliku

```
int remove (char* nazwa_pliku);
```

- ▶ Przesunięcie bieżącej pozycji na początek pliku

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

- ▶ Opróżnienie bufora dyskowego pliku

```
int fflush(FILE *fp);
```

Struktury

- ▶ Struktura jest agregatem (lub konglomeratem) typów danych i składa się z kilku związkanych zmiennych nazywanych składowymi (ew. polami lub elementami)
 - ▶ każda składowa może mieć własny typ, inny od typów innych składowych

```
struct nazwa-etykiety{  
    TYP składowa1;  
    TYP składowa2;  
    .  
    .  
    .  
    TYP składowaN;  
}lista-zmiennych;
```

- ▶ Jeden z elementów: *nazwa-etykiety* lub *lista-zmiennych* może zostać pominięty.

Struktury - deklaracja

```
nazwa typu struktury  
struct point{  
    int x;  
    int y;  
};  
  
struct complex{  
    float im;  
    float re;  
};  
  
struct panstwo{  
    char* stolica;  
    long ludnosc;  
    float powierzchnia;  
    char* kontynent;  
};
```

Deklaracja struktury określa jedynie nowy typ danych.
Obiekt tego typu nie istnieje do momentu zadeklarowania zmiennej.

Struktury – deklaracja

```
struct katalog{  
    char nazwisko[40]; /*nazwisko autora*/  
    char tytul[40]; /*tytuł*/  
    char wydawca[40]; /*wydawca*/  
    unsigned data; /*data wydania*/  
    unsigned char wyd; /*wydanie*/  
}karta;
```

nazwa typu struktury

nazwa zmiennej typu strukturalnego

Nazwy składowych struktury nie powodują konfliktów z innymi zmiennymi o takich samych nazwach.

Deklaracja zmiennych strukturalnych

```
struct katalog{  
    char nazwisko[40];  
    char tytul[40];  
    char wydawca[40];  
    unsigned data;  
    unsigned char wyd;  
}karta;
```

Deklaracja nazwa zmiennej typu strukturalnego na etapie deklaracji struktury.

```
struct katalog{  
    char nazwisko[40];  
    char tytul[40];  
    char wydawca[40];  
    unsigned data;  
    unsigned char wyd;  
};
```

```
struct katalog kartal;
```

Deklaracja nazwa zmiennej typu strukturalnego po uprzedniej deklaracji struktury.

Deklaracja zmiennych strukturalnych

- ▶ Deklaracja zmiennej typu strukturalnego:

```
struct nazwa-etykiety lista-zmiennych;
```

```
struct katalog zm1;  
struct katalog zm2, zm3;
```

- ▶ Każda instancja struktury zawiera własną kopię składowych struktury

```
zm1.tytul != zm2.tytul
```

Deklaracja zmiennych strukturalnych

- ▶ Odwołanie do składowej struktury wymaga podania nazwy struktury oraz nazwy składowej oddzielonych kropką

```
struct katalog{  
    char nazwisko[40];  
    char tytul[40];  
    char wydawca[40];  
    unsigned data;  
    unsigned char wyd;  
}karta;
```

Nazwa obiektu typu strukturalnego,
a nie struktury!

```
karta.data = 1776;  
printf("Data wydania: %u", karta.data);  
scanf("%c", &karta.wyd);  
gets(karta.nazwisko);  
printf("%c", karta.tytul[2]);
```

Przykład 15

```
#include <stdio.h>

struct typ_s{
    int i;
    char ch;
    double d;
    char nap[80];
}s;

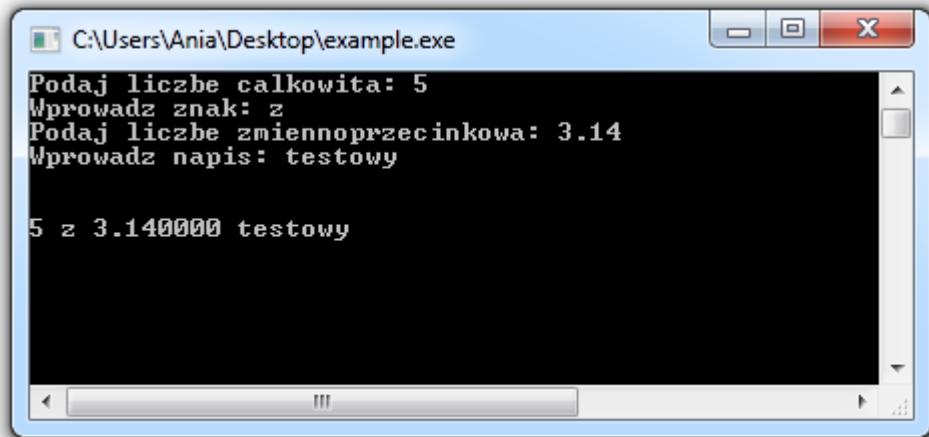
int main(void) {
    printf("Podaj liczbe calkowita: ");
    scanf("%d", &s.i);
    while(getchar() != '\n');

    printf("Wprowadz znak: ");
    scanf("%c", &s.ch);
    while(getchar() != '\n');

    printf("Podaj liczbe zmiennoprzecinkowa: ");
    scanf("%lf", &s.d);
    while(getchar() != '\n');

    printf("Wprowadz napis: ");
    scanf("%s", s.nap);
    while(getchar() != '\n');

    printf("\n\n%d %c %f %s", s.i, s.ch, s.d, s.nap);
    return 0;
}
```



Deklaracja zmiennych strukturalnych

- ▶ Jeżeli wiadomo, że jest potrzebna z góry ustalona liczba zmiennych danego typu strukturalnego, **można przy definicji typu pominąć nazwę etykiety**

```
struct{
    int a;
    char ch;
} zm1, zm2;
```

Deklaracja dwóch zmiennych strukturalnych.
Sama struktura pozostaje nienazwana;
w konsekwencji nie można utworzyć
kolejnych zmiennych tego typu.

Inicjalizacja pól struktury

- ▶ Poprzez bezpośrednie odwołanie do każdego z pól struktury

```
struct katalog kat1;
strcpy(kat1.nazwisko,"Sienkiewicz");
strcpy(kat1.tytul,"Potop");
strcpy(kat1.wydawca,"Literatura");
kat1.data = 2001;
kat1.wyd = 2;
```

- ▶ Inicjalizacja zbiorcza

```
struct katalog kat2 = {
    "Sienkiewicz",
    "Potop",
    "Literatura",
    2001,
    2
};
```

```
struct katalog{
    char nazwisko[40];
    char tytul[40];
    char wydawca[40];
    unsigned data;
    unsigned char wyd;
};
```

Przykład 16

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

struct katalog{
    char nazwisko[40];
    char tytul[40];
    unsigned data;
    unsigned char wyd;
};

int main(void){

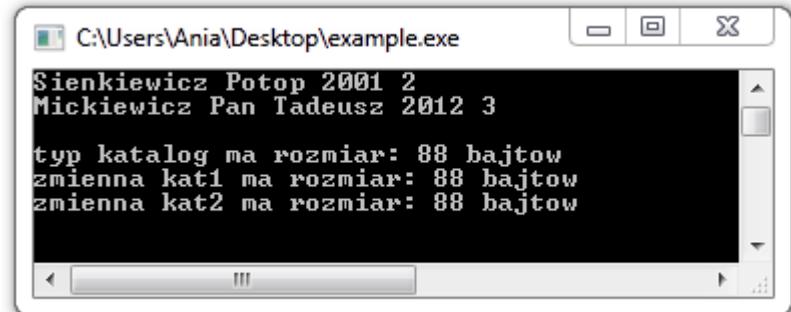
    struct katalog k1;
    strcpy(k1.nazwisko,"Sienkiewicz");
    strcpy(k1.tytul,"Potop");
    k1.data = 2001;
    k1.wyd = 2;

    struct katalog k2 = {"Mickiewicz","Pan Tadeusz",2012,3};

    printf("%s %s %d %d\n", k1.nazwisko, k1.tytul, k1.data, k1.wyd);
    printf("%s %s %d %d\n\n", k2.nazwisko, k2.tytul, k2.data, k2.wyd);

    printf("typ katalog ma rozmiar: %d bajtow\n", sizeof(struct katalog));
    printf("zmienna kat1 ma rozmiar: %d bajtow\n", sizeof(k1));
    printf("zmienna kat2 ma rozmiar: %d bajtow", sizeof(k2));

    return 0;
}
```



Tablice struktur

- ▶ Struktury mogą być umieszczane w tablicach w taki sam sposób, jak inne typy danych.

```
struct nazwa-etykiety nazwa-tablicy[ROZMIAR];
```

```
struct katalog{  
    char nazwisko[40];  
    char tytul[40];  
    char wydawca[40];  
    unsigned data;  
    unsigned char wyd;  
};  
  
struct katalog kat[100]; //tablica 100 struktur katalog  
  
kat[0]                      // pierwsza struktura w tablicy  
kat[33].wyd = 2;             // wypełnienie pola wyd w strukturze nr 33
```

Przykład 17

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

struct abonent{
    char imie[20];
    char nazwisko[50];
    char telefon[15];
};

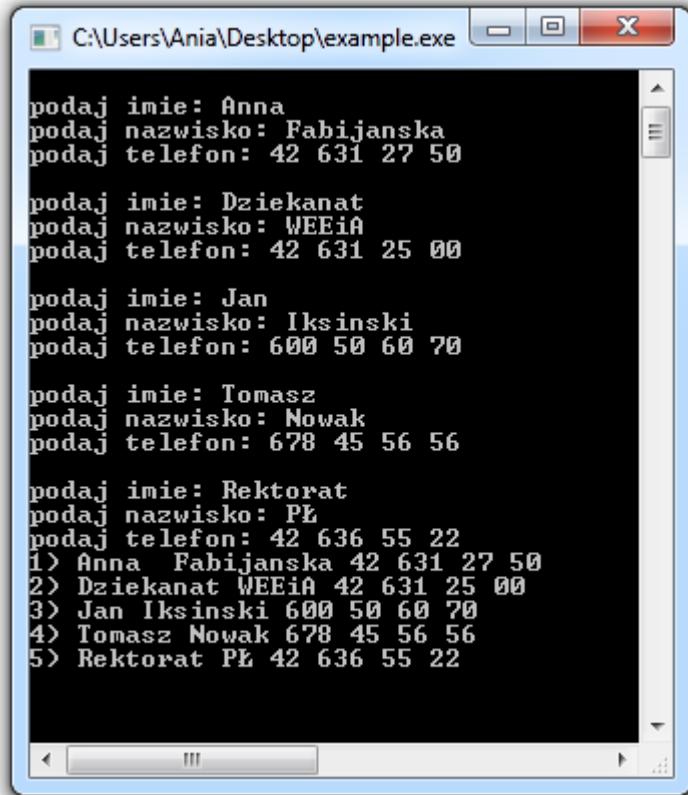
int main(void){

    int i;
    struct abonent spis[5];

    for (i=0; i<5; i++){
        printf("\npodaj imie: ");
        fgets(spis[i].imie, 19, stdin);
        printf("podaj nazwisko: ");
        fgets(spis[i].nazwisko, 49, stdin);
        printf("podaj telefon: ");
        gets(spis[i].telefon);
    }

    for (i=0; i<5; i++){
        printf("%d %s %s\n", i+1, spis[i].imie, spis[i].nazwisko, spis[i].telefon);
    }

    return 0;
}
```



```
C:\Users\Ania\Desktop\example.exe

podaj imie: Anna
podaj nazwisko: Fabijanska
podaj telefon: 42 631 27 50

podaj imie: Dziekanat
podaj nazwisko: WEEiA
podaj telefon: 42 631 25 00

podaj imie: Jan
podaj nazwisko: Iksinski
podaj telefon: 600 50 60 70

podaj imie: Tomasz
podaj nazwisko: Nowak
podaj telefon: 678 45 56 56

podaj imie: Rektorat
podaj nazwisko: PŁ
podaj telefon: 42 636 55 22
1> Anna Fabijanska 42 631 27 50
2> Dziekanat WEEiA 42 631 25 00
3> Jan Iksinski 600 50 60 70
4> Tomasz Nowak 678 45 56 56
5> Rektorat PŁ 42 636 55 22
```

Przykład 18

```
#include <stdio.h>

struct student_t{
    char imie[10];
    char nazwisko[20];
    int album;
    float oceny[5];
};

int main (void){

    struct student_t s = {"Jan", "Kowalski", 109778, {5.0, 4.5, 5.0, 4.0, 2.0}};

    FILE* fpb = fopen("C:/users/an_fab/Desktop/plik_b.txt", "wb");
    fwrite(&s, sizeof(struct student_t), 1, fpb);
    fclose(fpb);

    FILE* fpt = fopen("C:/users/an_fab/Desktop/plik_t.txt", "wt");
    fprintf(fpt, "%s %s %d\n", s.imie, s.nazwisko, s.album);
    for (int i=0; i<5; i++)
        fprintf(fpt, "%.1f ", s.oceny[i]);
    fclose(fpt);

    return 0;
}
```

The image shows two windows of the Windows Notepad application. The top window, titled 'plik_b.txt — Notatnik', contains the following text:

Jan	Kowalski	Ń-ł	@	@	@
€@	@				

The bottom window, titled 'plik_t.txt — Notatnik', contains the following text:

Jan Kowalski	109778				
5.0	4.5	5.0	4.0	2.0	

Both windows have status bars at the bottom showing 'Lin 1, kol 1', '100%', 'Windows (CRLF)', and either 'ANSI' or 'UTF-8' encoding.

Przykład 19

```
#include <stdio.h>

struct student_t{
    char imie[10];
    char nazwisko[20];
    int album;
    float oceny[5];
};

int main (void){
    struct student_t sb, st;

    FILE* fpb = fopen("plik_b.txt","rb");
    fread(&sb, sizeof(struct student_t), 1, fpb);
    fclose(fpb);

    FILE* fpt = fopen("plik_t.txt","rt");
    fscanf(fpt, "%s %s %d\n", st.imie, st.nazwisko, &st.album);
    for (int i=0; i<5; i++)
        fscanf(fpt,"%f ", &st.oceny[i]);
    fclose(fpt);

    printf("%s, %s, %d, %.1f\n", sb.imie, sb.nazwisko, sb.album, sb.oceny[3]);
    printf("%s, %s, %d, %.1f\n", st.imie, st.nazwisko, st.album, st.oceny[3]);

    return 0;
}
```

Jan, Kowalski, 109778, 4.0
Jan, Kowalski, 109778, 4.0
Process finished with exit code 0

Przykład 20

```
#include <stdio.h>

struct complex{
    float im, re;
}num;

int main (void){

    char id, res;
    struct complex data[5] = {{1,1},{2,2},{3,3},{4,4},{5,5}};

    FILE* fp = fopen("plik.txt","wb");
    fwrite(data, sizeof(struct complex), 5, fp);
    fclose(fp);

    fp = fopen("plik.txt", "rb");
    do {
        printf("input element id (1-5): ");
        scanf("%d", &id);
        fseek(fp, (id - 1) * sizeof(struct complex), SEEK_SET);
        fread(&num, sizeof(struct complex), 1, fp);
        printf("num: %.2f+j%.2f\n", num.im, num.re);
        printf("continue (press n to end)? ");
        scanf("%*c");
        res = getchar();
    }while(res!='n');
    fclose(fp);
}

52 return 0;
```

```
input element id (1-5): 5
num: 5.00+j5.00
continue (press n to end)? y
input element id (1-5): 1
num: 1.00+j1.00
continue (press n to end)? y
input element id (1-5): 3
num: 3.00+j3.00
continue (press n to end)? n
```



```

#include <stdio.h>

struct complex{
    float im, re;
};

int main (void){
    int id, res;
    struct complex data[5] = {{1,1},{2,2},{3,3},{4,4},{5,5}};
    struct complex c1 = {6, 6}, c2 = {7,7}, temp;

    FILE* fp = fopen("plik.txt","wb");
    fwrite(data, sizeof(struct complex),5, fp);
    fclose(fp);

    fp = fopen("plik.txt", "rb+");
    fseek(fp,0,SEEK_END);
    fwrite(&c1, sizeof(struct complex), 1, fp); //append c1 to file
    fwrite(&c2, sizeof(struct complex), 1, fp); //append c2 to file
    fclose(fp);

    fp = fopen("plik.txt", "rb");
    while(1){
        fread(&temp, sizeof(struct complex), 1, fp);
        if(feof(fp))
            break;
        printf("num: %.2f+j%.2f\n", temp.im, temp.re);
    }
    fclose(fp);
    return 0;
}

```

Przykład 21

```

num: 1.00+j1.00
num: 2.00+j2.00
num: 3.00+j3.00
num: 4.00+j4.00
num: 5.00+j5.00
num: 6.00+j6.00
num: 7.00+j7.00

```

Struktury jako parametry funkcji

- ▶ Struktury – tak jak inne typy danych, mogą być parametrami funkcji.

TYP nazwa-funkcji (struct nazwa-etykiety);

TYP nazwa-funkcji (struct nazwa-etykiety[ROZMIAR]);

```
struct abonent{  
    char imie[20];  
    char nazwisko[50];  
    char telefon[15];  
};  
  
void drukuj (struct abonent); //deklaracja funkcji  
...  
void drukuj (struct abonent dane) //definicja funkcji  
{  
printf("%s %s %s\n", dane.imie, dane.nazwisko, dane.telefon);  
}
```

typ danych a nie nazwa zmiennej

parametr formalny

Przykład 22 (1)

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

/*definicja struktury*/
struct abonent{
    char imie[20];
    char nazwisko[50];
    char telefon[15];
};

/*deklaracje funkcji z parametrami typu strukturalnego*/
int menu(void);
void wprowadz_dane(struct abonent[]);
void drukuj_dane(struct abonent[]);
void zapisz_do_pliku(struct abonent[]);
void wczytaj_z_pliku(struct abonent[]);
void wyszukaj_abonenta(char *nazwisko, char *imie);
void aktualizacja_danych(char* nazwisko, char* nowe_nazwisko);
```

Przykład 22 (2)

```
int main(void) {  
  
    int wybor;  
    struct abonent dane[5];  
  
    do{  
        wybor = menu();  
  
        switch(wybor) {  
            case 1:  
                wprowadz_dane(dane); break;  
            case 2:  
                drukuj_dane(dane); break;  
            case 3:  
                zapisz_do_pliku(dane); break;  
            case 4:  
                wczytaj_z_pliku(dane); break;  
            case 5:  
                wyszukaj_abonenta("Kowalski", "Jan"); break;  
            case 6:  
                aktualizacja_danych("Kowalski", "Vacavant"); break;  
        }  
  
    }while(wybor!=7);  
    return 0;  
}
```

Przykład 22 (3)

```
int menu(void) {  
  
    int i;  
    char nap[80];  
  
    printf("\n\nWybierz operacje:\n");  
    printf("1. Wprowadzenie danych\n");  
    printf("2. Wydruk danych\n");  
    printf("3. Zapis do pliku\n");  
    printf("4. Odczyt z pliku\n");  
    printf("5. Wyszukiwanie abonenta\n");  
    printf("6. Aktualizacja danych\n");  
    printf("7. Koniec\n");  
  
    do{  
        printf("Wybierz polecenie: ");  
        fgets(nap, 79, stdin);  
        i=atoi(nap);  
        printf("\n\n");  
    }while(i<1 || i> 7);  
  
    return i;  
}
```

Przykład 22 (4)

```
void wprowadz_dane(struct abonent spis[]){
    int i;
    for (i=0; i<5; i++){
        printf("\npodaj imie: ");
        fgets(spis[i].imie, 19, stdin);
        printf("podaj nazwisko: ");
        fgets(spis[i].nazwisko, 49, stdin);
        printf("podaj telefon: ");
        fgets(spis[i].telefon, 14, stdin);
    }
    return;
}

void drukuj_dane(struct abonent spis[]){
    int i;
    for (i=0; i<5; i++){
        printf("%d %s %s %s\n", i+1, spis[i].imie, spis[i].nazwisko,
               spis[i].telefon);
    }
    return;
}
```

Przykład 22 (5)

```
void zapisz_do_pliku(struct abonent spis[]){  
  
FILE *fp;  
  
if((fp = fopen("katalog","wb"))==NULL) {  
    printf("Blad otwarcia pliku katalogu\n");  
    exit(1);  
}  
  
if ((fwrite(spis, sizeof (struct abonent), 5, fp))!=5) {  
    printf("Blad zapisu danych do katalogu.\n");  
    exit(1);  
}  
  
fclose(fp);  
}
```

Przykład 22 (6)

```
void wczytaj_z_pliku(struct abonent spis[]){  
  
FILE *fp;  
  
if((fp = fopen("katalog","rb"))==NULL) {  
    printf("Blad otwarcia pliku katalogu\n");  
    exit(1);  
}  
  
if (fread(spis, sizeof (struct abonent), 5, fp)!=5){  
    printf("Blad zapisu danych do katalogu,\n");  
    exit(1);  
}  
  
fclose(fp);  
}
```

Przykład 22 (7)

```
void wyszukaj_abonenta(char *nazwisko, char *imie){
    FILE *fp;

    if((fp = fopen("D:/katalog", "rb"))==NULL) {
        printf("Blad otwarcia pliku katalogu\n");
        exit(1);
    }

    struct abonent t;

    while(1) {
        fread(&t, sizeof(struct abonent), 1, fp);

        if(feof(fp))
            break;

        if(!strcmp(nazwisko, t.nazwisko) && !strcmp(imie, t.imie))
            printf("%s %s %s\n", t.imie, t.nazwisko, t.telefon);
    }
    fclose(fp);
    return;
}
```

Przykład 22 (8)

```
void aktualizacja_danych(char* nazwisko, char* nowe_nazwisko){  
    FILE *fp;  
  
    if((fp = fopen("D:/katalog","rb+"))==NULL) {  
        printf("Blad otwarcia pliku katalogu\n");  
        exit(1);  
    }  
  
    struct abonent temp;  
  
    while(1){  
        fread(&temp, sizeof(struct abonent), 1, fp);  
  
        if(feof(fp))  
            break;  
  
        if(!strcmp(nazwisko, temp.nazwisko))  
            strcpy(temp.nazwisko, nowe_nazwisko);  
  
        fseek(fp, -sizeof(struct abonent), SEEK_CUR);  
        fwrite(&temp, sizeof(struct abonent), 1, fp);  
    }  
    fclose(fp);  
    return;
```

1) Jan Kowalski 42 719 44 44
2) Tomasz Maliniak 42 643 11 44
3) Genowefa Pigwa 42 336 77 77
4) Tomasz Kowalski 42 556 77 77
5) Anna Fabijanska 42 631 27 50

1) Jan Vacavant 42 719 44 44
2) Tomasz Maliniak 42 643 11 44
3) Genowefa Pigwa 42 336 77 77
4) Tomasz Vacavant 42 556 77 77
5) Anna Fabijanska 42 631 27 50

Wskaźniki do struktur

- ▶ Wskaźnik do struktury definiuje się tak, jak wskaźnik do innych typów danych.

```
zmienna strukturalna
                    ↓
struct typ_s{
    int i;
    char nap[80];
}s, *p
      ↑
p = &s;                                wskaźnik do struktury
                                         typu typ_s
                                         ↑
                                         ustawienie wskaźnika
                                         ↑
struct typ_s t, *q;
q = &t;
```

Wskaźniki do struktur

- Przy dostępie do pola struktury za pośrednictwem wskaźnika, zamiast kropki należy użyć „operatora strzałkowego” tj.: ->

```
struct nazwa-etykiety s, *pt;  
pt = &s;  
s. nazwa-pola ⇔ pt->nazwa-pola
```

```
struct typ_s{  
    int i;  
    char nap[80];  
}s, *p  
p = &s;
```

Dostęp bezpośredni

```
s.i = 10;  
gets(s.nap);
```

Dostęp poprzez wskaźnik

```
p->i = 10;  
gets(p->nap);
```

Przykład 23

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

struct typ_s{
    int i;
    char nap[80];
} s, *p;

int main(){
    p=&s;

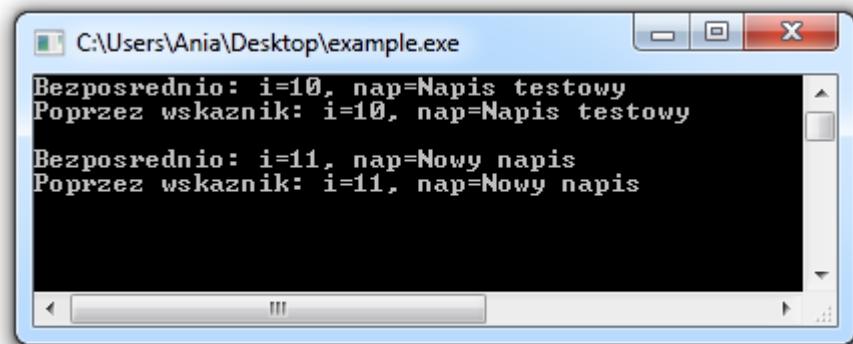
    s.i = 10;
    strcpy(s.nap,"Napis testowy");

    printf("Bezposrednio: i=%d, nap=%s\n", s.i, s.nap);
    printf("Poprzez wskaznik: i=%d, nap=%s\n", p->i, p->nap);

    p->i = 11;
    strcpy(p->nap, "Nowy napis");

    printf("\nBezposrednio: i=%d, nap=%s\n", s.i, s.nap);
    printf("Poprzez wskaznik: i=%d, nap=%s\n", p->i, p->nap);

    return 0;
}
```



Przykład 24a

```
#include <stdio.h>
#include <string.h>
/*****************/
struct typ_s{
    int i;
    char nap[80];
};

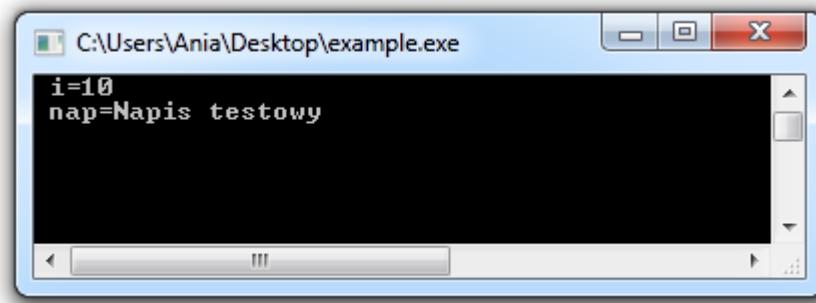
/*****************/
void wypelnij (struct typ_s*, int, char[]);
/*****************/
int main(){

    struct typ_s s;

    wypelnij(&s, 10, "Napis testowy");

    printf(" i=%d\n nap=%s\n", s.i, s.nap);

    return 0;
}
/*****************/
void wypelnij (struct typ_s *s, int i, char nap[]){
    s->i = i;
    strcpy(s->nap, nap);
}
```



Przykład 24b

```
#include <stdio.h>
#include <string.h>
/*****************/
struct typ_s{
    int i;
    char nap[80];
};

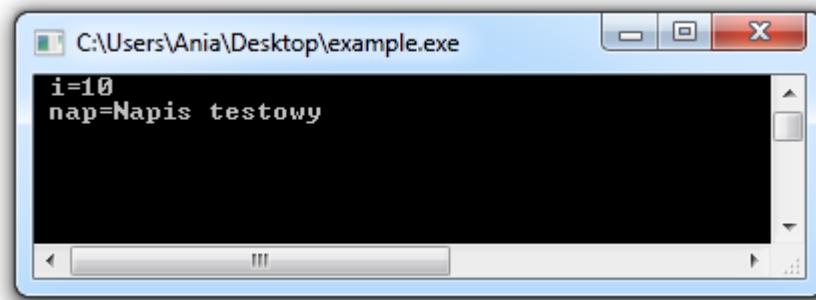
/*****************/
void wypelnij (struct typ_s*, int, char[]);
/*****************/
int main(){

    struct typ_s s, *p=&s;

    wypelnij(p, 10, "Napis testowy");

    printf(" i=%d\n nap=%s\n", s.i, s.nap);

    return 0;
}
/*****************/
void wypelnij (struct typ_s *s, int i, char nap[]){
    s->i = i;
    strcpy(s->nap, nap);
}
```



Struktury zagnieżdżone

- ▶ Elementami struktur mogą być inne struktury – wówczas mówi się o **strukturach zagnieżdżonych**

```
struct ksiazka_t typ{  
    unsigned data;  
    unsigned char wyd;  
    unsigned stron;  
}
```

```
struct katalog{  
    char nazwisko[80];  
    char tytul[80];  
    char wydawca[80];  
    struct ksiazka_t typ ksiazka;  
} rekord;
```

- ▶ Odwołanie do pola w strukturze będącej elementem innej:

```
rekord.ksiazka.data = 2001;  
rekord.ksiazka.wyd = 2;  
rekord.ksiazka.stron = 201;
```

Struktury zagnieżdżone

```
struct addressInfo {  
    char address[30];  
    char city[10];  
    char state[2];  
    long int zip;  
}
```

```
struct employees {  
    char empName[30];  
    struct addressInfo eAddress;  
    double salary;  
}
```

```
struct customers {  
    char custName[30];  
    struct addressInfo cAddress;  
    double balance;  
}
```

Tworzenie nowych nazw typów danych

- ▶ Język C dysponuje mechanizmem nazywanym **typedef**, który umożliwia nadawanie istniejącym typom danych nowych nazw

```
typedef TYP nowa-nazwa-typu;
```

```
typedef int Length; // utworzenie synonimu dla typu int  
Length zm1, zm2; // synonim typu int wykorzystany  
Length *len[100]; // podczas deklaracji zmiennych
```

- ▶ Użycie **typedef** nie unieważnia nazwy oryginalnej.
- ▶ Za pomocą **typedef** można nadać wiele różnych nazw jednemu typowi.

Przykład 25

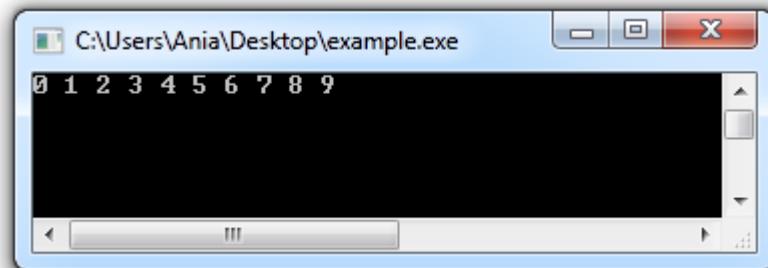
```
#include <stdio.h>
```

```
typedef signed char malicz;
```

```
int main(void)
{
    malicz i;

    for(i=0; i<10; i++)
        printf("%d ", i);

    return 0;
}
```



Tworzenie nowych nazw typów danych

- ▶ Nazwę utworzoną przez **typedef**, można wykorzystać w kolejnych definicjach typów

```
typedef int wysokosc;
typedef wysokosc dlugosc;
typedef dlugosc glebokosc;

glebokosc d; // d jest typu int
```

```
enum typ_w {jeden, dwa, trzy};
typedef enum typ_w mojelicz;

mojelicz num; // num jest typu typ_w
```

- ▶ **typedef** jest często stosowany z definicją struktury, w celu uniknięcia powtarzania słowa kluczowego **struct**

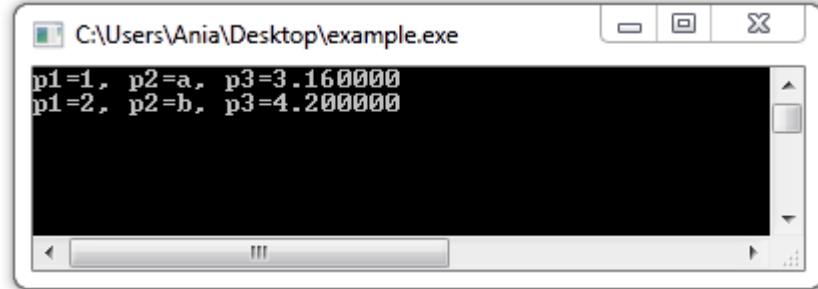
Przykład 26a

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

```
typedef struct strukturka{  
    int p1;  
    char p2;  
    double p3;  
} Struktura;
```

```
void drukuj (Struktura);  
  
int main(){  
    Struktura s1 = {1, 'a', 3.16};  
    struct strukturka s2 = {2, 'b', 4.20};  
  
    drukuj(s1);  
    drukuj(s2);  
  
    return 0;  
}
```

```
void drukuj (Struktura s){  
    printf("p1=%d, p2=%c, p3=%f\n", s.p1, s.p2, s.p3);  
    return;  
}
```



definicja typu strukturalnego

nowa nazwa dla typu strukturalnego

Przykład 26b

```
#include <stdio.h>

typedef struct{
    int p1;
    char p2;
    double p3;
} Struktura;

void drukuj (Struktura);

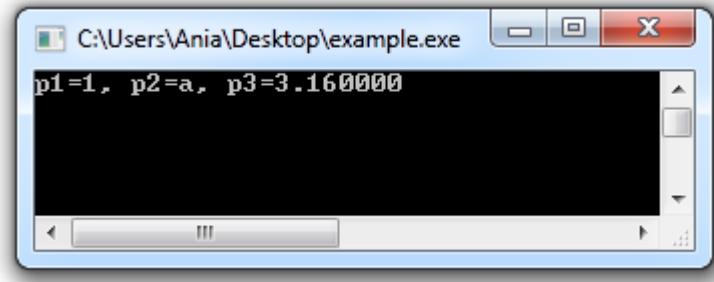
int main(){

    Struktura s1 = {1, 'a', 3.16};

    drukuj(s1);

    return 0;
}

void drukuj (Struktura s){
    printf("p1=%d, p2=%c, p3=%f\n", s.p1, s.p2, s.p3);
    return;
}
```



Pola bitowe

- ▶ Język C dopuszcza rodzaj składowej struktury nazwany polem bitowym
- ▶ Korzystając z pola bitowego, można sięgać do konkretnych bitów bajtu lub słowa

Nazwa-typu: rozmiar;

liczba bitów

- ▶ Nazwa-typu – `int` lub `unsigned`
w przypadku `int` bitem znaku jest najstarszy znak
- ▶ Cel stosowania – upakowywanie informacji do możliwie najmniejszego rozmiaru

Pola bitowe

- ▶ Przykład: przechowywanie informacji inwentaryzacyjnej
 - ▶ wykorzystanie jednego bajta, zamiast czterech

```
struct typ_b{  
    unsigned dzial: 3;          /* do 7 działów */  
    unsigned naskladzie: 1;     /* 1 jeśli tak, 0 jeśli nie */  
    unsigned zamowione: 1;      /* 1 jeśli tak, 0 jeśli nie */  
    unsigned wyprzedzenie: 3;   /* do 7 miesięcy */  
}inv[MAKS_ELEM];
```

- ▶ Odwołanie do pola bitowego

```
inv[9].dzial = 3;  
  
if(!inv[4].naskladzie)  
    printf("Brak na składzie");  
else  
    printf("Na składzie - dzial %d\n", inv[4].dzial);
```

Pola bitowe

- ▶ W strukturze, nie trzeba definiować wszystkich bitów do pełnego bajtu.

```
struct typ_b{  
    int a: 2;  
    int b: 3;  
}
```

- ▶ W strukturze, nie trzeba nazywać każdego bitu.

```
struct typ_b{  
    unsigned pierwszy: 1;           wykorzystanie pierwszego  
    int: 6;                      i ostatniego bitu w bajcie  
    unsigned ostatni: 1;  
}
```

Przykład 27(1)

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
```

```
struct telemetria{
    unsigned paliwo: 1;
    unsigned radio: 1;
    unsigned tv: 1;
    unsigned woda: 1;
    unsigned jedzenie: 1;
    unsigned odpady: 1;
}zapis_lotu;
/*****************/
void wyswietl(struct telemetria i);
void zapisz(struct telemetria i);
/*****************/
int main(void){
    int i;
    srand(time(NULL));

    for (i=0; i<10; i++){
        zapis_lotu.paliwo = rand()%2;
        zapis_lotu.radio = rand()%2;
        zapis_lotu.tv = rand()%2;
        zapis_lotu.woda = rand()%2;
        zapis_lotu.jedzenie = rand()%2;
        zapis_lotu.odpady = rand()%2;

        wyswietl(zapis_lotu);
        zapisz(zapis_lotu);
    }
    return 0;
}
```

Przykład 27₍₂₎

```
void wyswietl(struct telemetria i){  
  
    if(i.paliwo) printf("Paliwo OK\n");  
    else printf("Rezerwa paliwa\n");  
  
    if(i.radio) printf("Radio OK\n");  
    else printf("Awaria radia\n");  
  
    if(i.tv) printf("System telewizyjny OK\n");  
    else printf("Awaria systemu telewizyjnego\n");  
  
    if(i.woda) printf("Zapasy wody OK\n");  
    else printf("Rezerwa zapasow wody\n");  
  
    if(i.jedzenie) printf("Zapasy jedzenia OK\n");  
    else printf("Rezerwa zapasow jedzenia\n");  
  
    if(i.odpady) printf("Przechowywanie odpadow OK\n");  
    else printf("Awaria systemu przechowywania odpadow\n");  
}
```

Przykład 27(3)

```
void zapisz(struct telemetria i){  
  
    FILE *fp;  
    int ile;  
  
    if((fp=fopen("lot","wb"))==NULL) {  
  
        printf("Blad otwarcia pliku");  
        exit(1);  
    }  
  
    if ((ile=fwrite(&i, sizeof i, 1, fp))!=1){  
  
        printf("Blad zapisu");  
        exit(1);  
    }  
  
    fclose(fp);  
}
```

Unie

- ▶ W języku C unia jest pojedynczym obszarem pamięci, dzielonym przez kilka zmiennych (jednego lub różnego typu).
- ▶ Zmienne dzielące obszar mogą być różnych typów.
- ▶ Tylko jedna zmienna z unii może być wykorzystywana w danej chwili.

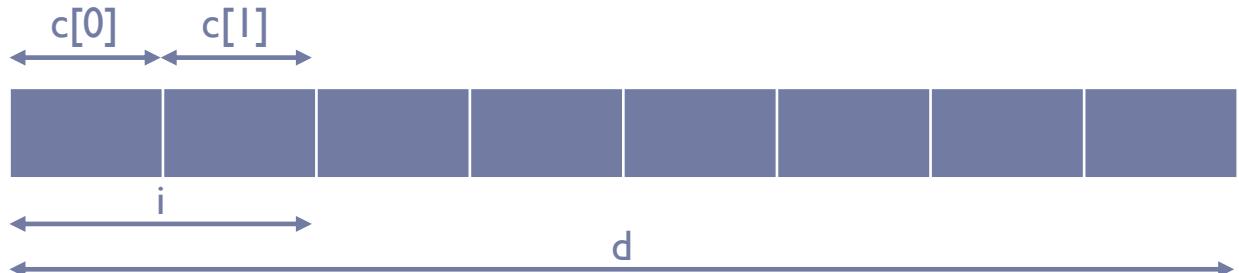
```
union nazwa-etykiety{  
    TYP składowa1;  
    TYP składowa2;  
    ...  
    TYP składowaN;  
}lista-zmiennych;
```

- ▶ Nazwa-etykiety lub lista-zmiennych mogą zostać pominięte.

Unie

- ▶ Sposób przechowywania unii w pamięci:

```
union typ_u{  
    int i;  
    char c[2];  
    double d;  
}probka;
```



- ▶ Rozmiar unii jest taki, jak rozmiar jej największego elementu.
- ▶ Typowe użycie:
 - ▶ tablica wartości, których typów nie można określić na etapie pisania kodu
 - ▶ tworzenie tablicy jednostek o jednakowej długości, z których każda może przechowywać dane innego typu.

Unie - deklaracja

▶ Deklaracja unii:

```
union magazyn{  
    int cyfra;  
    double duzf1;  
    char litera;  
};
```



rozmiar: 8 bajtów (64 bity)

▶ Deklaracja zmiennych:

```
union magazyn fit;      /* unia, 8 bajtów */  
union magazyn tab[10];  /* tablica 10 unii, 10 x 8 bajtów */  
union magazyn *wu;     /* wskaźnik do unii, 4 bajty */
```

Unie - wskaźniki

▶ Inicjalizacja

```
union magazyn{  
    int cyfra;  
    double duzfl;  
    char litera;  
};
```

```
union magazyn wartA, fit;  
wartA.litera = 'R';
```

```
union magazyn wartB = wartA; /*przypisanie innej unii*/  
union magazyn wartC = {88}; /*inicjalizacja składnika - tylko  
                           jednego - typ zgodny z typem pierwszej składowej! */  
fit.cyfra = 23; /*23 zapisane w fit; zajęte 2 bajty*/  
fit.duzfl = 2.0; /*23 usunięte, 2.0 zapisane, zajęte 8 bajtów*/  
fit.litera = 'h'; /*2.0 usunięte, h zapisane, zajęty 1 bajt*/
```

W danym momencie unia może przechowywać tylko jedną wartość

Unie – wskaźniki

- ▶ Deklaracja i inicjalizacja:

```
union typ_u{  
    int i;  
    char c[2];  
    double d;  
}probka, *wsk;  
wsk = &probka.
```

- ▶ Sieganie do elementu unii:
 - ▶ bezpośrednio: nazwa_unii.nazwa_pola:
 - ▶ **probka.i = 10;**
 - ▶ poprzez wskaźnik: nazwa_wskaźnika->nazwa.pola
 - ▶ **wsk->i = 10;**

Przykład 28a

```
#include <stdio.h>

struct punkt1 {
    int x, y;
};

union punkt2 {
    int x, y;
};

int main() {

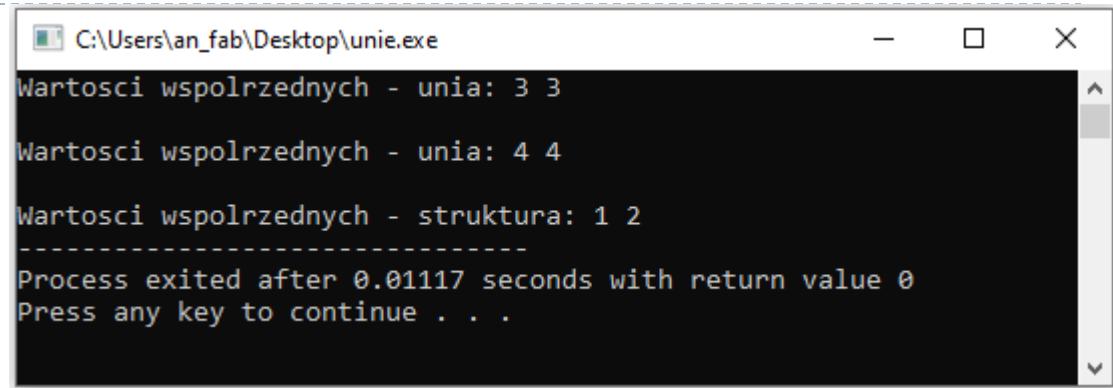
    struct punkt1 t1 = {1,2};
    union punkt2 t2;

    t2.x = 3;
    printf ("Wartosci wspolrzednych - unia: %d %d\n\n", t2.x, t2.y);

    t2.y = 4;
    printf ("Wartosci wspolrzednych - unia: %d %d\n\n", t2.x, t2.y);

    printf("Wartosci wspolrzednych - struktura: %d %d", t1.x,t1.y);

    return 0;
}
```



```
C:\Users\an_fab\Desktop\unie.exe
Wartosci wspolrzednych - unia: 3 3
Wartosci wspolrzednych - unia: 4 4
Wartosci wspolrzednych - struktura: 1 2
-----
Process exited after 0.01117 seconds with return value 0
Press any key to continue . . .
```

Przykład 28b

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

union typ_u{
    int i;
    char c[2];
    double d;
}probka, *wsk = & probka;

int main(void){

    probka.d = 19.11;

    printf("probka.d = %f\n", probka.d);
    printf("probka.c = %s\n", probka.c);
    printf("probka.i = %d\n\n", probka.i);

    wsk->i = 0;

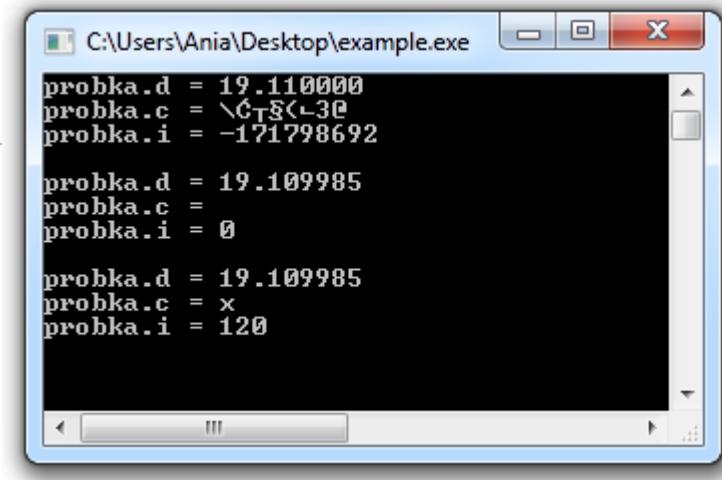
    printf("probka.d = %f\n", probka.d);
    printf("probka.c = %s\n", probka.c);
    printf("probka.i = %d\n\n", probka.i);

    strcpy(probka.c, "x");

    printf("probka.d = %f\n", probka.d);
    printf("probka.c = %s\n", probka.c);
    printf("probka.i = %d\n", probka.i);

    while(!kbhit());

    return 0;
}
```



```

#include<stdio.h>

struct student {
    union {
        char nazwisko[10];
        int album;
    };
    int ocena;
};

int main() {

    struct student stud;
    char wybor;

    printf("Mozesz prowadzic nazwisko lub numer albumu\n");
    printf("Czy chcesz wprowadzic nazwisko (t lub n): ");
    scanf("%c", &wybor);

    if(wybor=='t' || wybor=='T') {
        printf("Podaj nazwisko: ");
        scanf("%s", stud.nazwisko);
        printf("Name:%s", stud.nazwisko);
    }
    else {
        printf("Podaj numer albumu");
        scanf("%d", &stud.album);
        printf("Numer albumu:%d", stud.album);
    }

    printf("\nPodaj ocene:");
    scanf("%d", &stud.ocena);
    printf("Ocena:%d", stud.ocena);

    return 0;
}

```

unia w strukturze

```

C:\Users\an_fab\Desktop\unie.exe
Mozesz prowadzic nazwisko lub numer albumu
Czy chcesz wprowadzic nazwisko (t lub n): n
Podaj numer albumu19918
Numer albumu:19918
Podaj ocene:4
Ocena:4

C:\Users\an_fab\Desktop\unie.exe
Mozesz prowadzic nazwisko lub numer albumu
Czy chcesz wprowadzic nazwisko (t lub n): t
Podaj nazwisko: Kowalski
Name:Kowalski
Podaj ocene:5
Ocena:5

```

alternatywne wykorzystanie
pola w strukturze

Przykład 29

Przykład 30

```
#include<stdio.h>

struct job_t{

    union {
        char * company;
        char * school;
        char * project;
    };
    union {
        char * location;
        char * url;
    };
    union {
        char * title;
        char * program;
    };
    char * description;
};

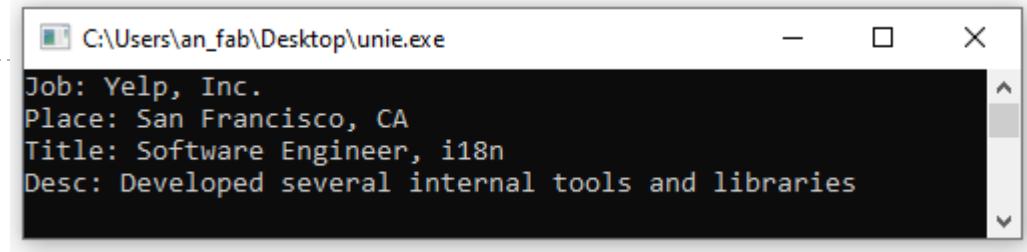
int main()
```

```
struct job_t yelp = {
    .company = "Yelp, Inc.",
    .location = "San Francisco, CA",
    .title = "Software Engineer, i18n",
    .description = "Developed several internal tools and libraries"};
```

```

    printf("Job: %s\n", yelp.project);
    printf("Place: %s\n", yelp.url);
    printf("Title: %s\n", yelp.program);
    printf("Desc: %s\n", yelp.description);

    return 0;
}
```



Trzy zagnieżdżone anonimowe unie

Inicjalizacja zbiorcza

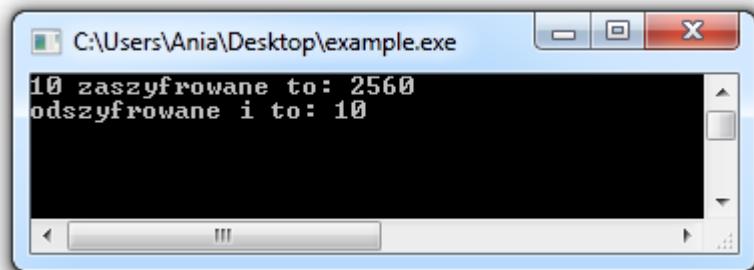
Szyfrowanie liczby całkowitej poprzez zamianę jej dwóch najmłodszych bajtów

Przykład 31

```
#include <stdio.h>

int szyfruj (int) ;

int main(void) {
    int i;
    i = szyfruj(10);
    printf("10 zaszyfrowane to: %d\n", i);
    i = szyfruj(i);
    printf("odszyfrowane i to: %d\n", i);
    return 0;
}
```



```
int szyfruj(int i)
{
    union typ_szyfr{
        int num;
        char c[2];
    }szyfr;
    unsigned char ch;
    szyfr.num=i;
    /*zamiana bajtów*/
    ch = szyfr.c[0];
    szyfr.c[0]=szyfr.c[1];
    szyfr.c[1]=ch;
    return szyfr.num;
}
```

Wyświetlanie binarnej reprezentacji znaku wprowadzonego z klawiatury

Przykład 32

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

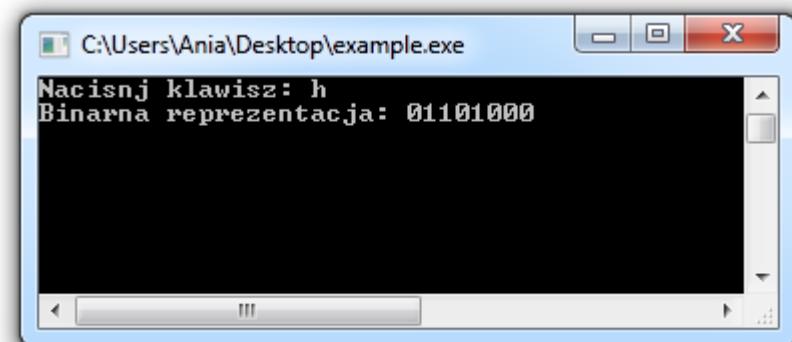
struct probka{
    unsigned a: 1;
    unsigned b: 1;
    unsigned c: 1;
    unsigned d: 1;
    unsigned e: 1;
    unsigned f: 1;
    unsigned g: 1;
    unsigned h: 1;
};

union typ_klawisz{
    char ch;
    struct probka bity;
} klawisz;
```

1.

```
int main(void) {
    printf("Nacisnij klawisz: ");
    klawisz.ch = getche();
    printf("\nBinarna reprezentacja: ");
    klawisz.bity.h ? printf("1") : printf("0");
    klawisz.bity.g ? printf("1") : printf("0");
    klawisz.bity.f ? printf("1") : printf("0");
    klawisz.bity.e ? printf("1") : printf("0");
    klawisz.bity.d ? printf("1") : printf("0");
    klawisz.bity.c ? printf("1") : printf("0");
    klawisz.bity.b ? printf("1") : printf("0");
    klawisz.bity.a ? printf("1") : printf("0");
    return 0;
}
```

2.



Operator „potrójny”

- ▶ Jedyny trójargumentowy operator języka C, określany często mianem **operatora warunkowego**

```
zmienna = warunek ? wyr1:wyr2;
```

- ▶ Pozwala na zastąpienie wyrażenia postaci:

```
if(warunek) zmienna = wyr1;  
else zmienna = wyr2;
```

- ▶ Dzięki użyciu operatora warunkowego, kompilator języka jest w stanie wygenerować wydajniejszy kod, niż w przypadku instrukcji **if-else**

Przykład 33

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>

int main(void) {

    int i;
    while(!kbhit()) rand();

    i=rand()%2 ? 1 : 0;

    if(i) printf ("Orzeł ");
    else printf("Reszka ");

    return 0;
}
```

Przykład 34

```
#include <stdio.h>
union tagname
{
    unsigned int a;
    unsigned char s[4];
} object;

int main (void){
    unsigned int a;
    object.a=0xAABBCCDD;
```

Address 0 1 2 3 4 5 6 7 8 9 a b c d e f Dump
00000000 dd cc bb aa
length : 4 lines : 1 Ln: 1 Col: 1 Pos: 1 Hex BigEndian INS ..
Integer number: 2864434397, hex: AABBCCDD
Individual bytes: AA BB CC DD
num: 2864434397
Process finished with exit code 0

Liczba całkowita zapisana w systemie HEX

```
FILE* fp = fopen("C:/users/an_fab/Desktop/plik.txt", "wb");

int ret = fwrite(&object, sizeof(union tagname), 1, fp);

printf("Integer number: %u, hex: %X\n", object.a, object.a);
printf("Individual bytes: ");

for(char i=3;i>=0;i--)
    printf("%02X ", object.s[i]);
fclose(fp);

fp = fopen("C:/users/an_fab/Desktop/plik.txt", "rb");
fread(&a, sizeof(unsigned int), 1, fp);
printf("\nnum: %u\n", a);
fclose(fp);
```

return 0;

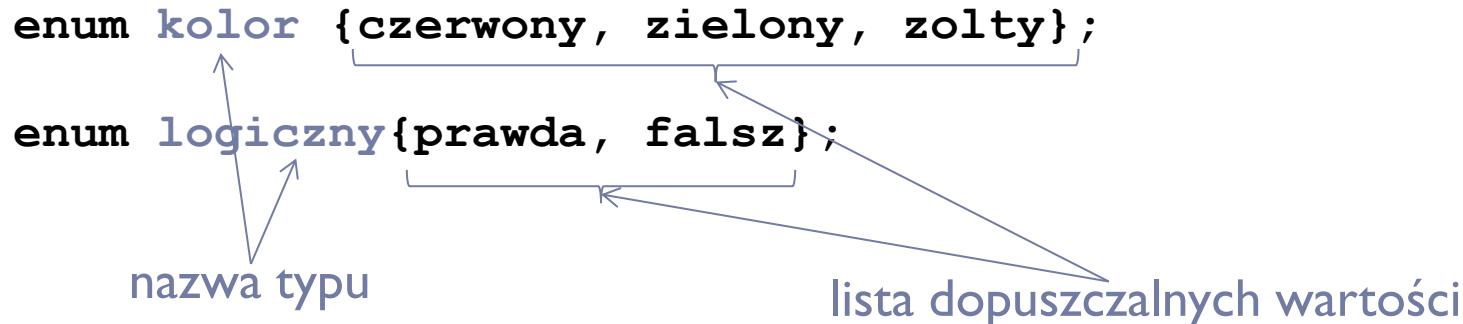
Typ wyliczeniowy

- ▶ Enumeracje (lub wyliczenia) to typy, których wartości wyznacza pewien zbiór nazwanych stałych **całkowitoliczbowych**.
- ▶ Stałe mogą być następnie wykorzystane w dowolnym miejscu programu, zamiast liczby całkowitej

```
enum nazwa-etykiety {lista-wyliczeniowa} lista-zmiennych;
```

- ▶ Jeden z elementów: *nazwa-etykiety* lub *lista-zmiennych* może zostać pominięty.

Typ wyliczeniowy



- O ile nie podane zostaną inne wartości, pierwsza nazwa w wyliczeniu ma wartość 0, druga 1 itd.
- Deklaracja zmiennej typu wyliczeniowego:

```
enum kolor zm;  
enum logiczny x, y, z;
```

- Stałe wyliczeniowe nie są napisami a nazwanymi liczbami całkowitymi

Przykład 35a

```
#include <stdio.h>

enum kolor {czerwony, zielony, zolty};
enum kolor k;

int main() {

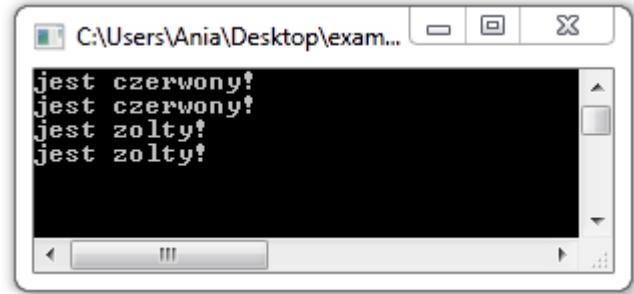
    k=czerwony;

    if (k==czerwony) printf("jest czerwony!\n");
    if (k==0) printf("jest czerwony!\n");

    k=zolty;

    if (k==zolty) printf("jest zolty!\n");
    if (k==2) printf("jest zolty!\n");

    return 0;
}
```



A screenshot of a terminal window titled 'C:\Users\Ania\Desktop\exam...'. The window contains the following text:
jest czerwony!
jest czerwony!
jest zolty!
jest zolty!

Przykład 35b

```
#include <stdio.h>

enum kolor {czerwony, zielony, zolty};
enum kolor k;

int main() {

    k=czerwony;

    if (k==czerwony) printf("jest czerwony!\n");
    if (k==0) printf("jest czerwony!\n");

    k=2; //BŁĄD

    if (k==zolty) printf("jest zolty!\n");
    if (k==2) printf("jest zolty!\n");

    return 0;
}
```

Typ wyliczeniowy

- ▶ Jeśli nie wszystkie wartości są określone, poszczególnym pozycjom przypisywane są rosnące wartości całkowite, począwszy od ostatniej wartości określonej jawnie.

```
enum dzien {PON = 1, WT, SR, CZW, PT, SOB, NIE};  
  
/* element PON ma wartość 1, WT ma wartość 2,  
 z kolei NIE ma wartość 7 */  
  
enum marka {VOLKSWAGEN, AUDI, SKODA = 9, BENTLEY,  
 BUGATTI = 32, LAMBORGHINI};  
  
/* VOLKSWAGEN = 0, AUDI = 1,  
 * SKODA = 9, BENTLEY = 10,  
 * BUGATTI = 32, LAMBORGHINI = 33 */
```

Przykład 36

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>

enum srodek_transportu{samochod, pociag, samolot, autobus};

int main(void) {
    printf("Nacisnij klawisz, aby wybrac srodek transportu: ");
    int wybor;
    while(!kbhit()) wybor = rand()%4;
    getch();
    if (wybor == samochod)
        printf("samochod \n");
    else if (wybor == pociag)
        printf("pociag \n");
    else if (wybor == samolot)
        printf("samolot \n");
    else if (wybor == autobus)
        printf("autobus \n");
    return 0;
}
```

Typ wyliczeniowy

- ▶ Nazwy wewnętrz enumeraacji nie mogą się powtarzać.
Mogą się natomiast powtarzać przypisane im wartości.

```
enum boolean {NO = 0, FALSE = 0, YES = 1, TRUE = 1};
```

/* albo krócej: */

```
enum boolean {NO, FALSE = 0, YES, TRUE = 1};
```

Przykład 37

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

typedef enum {TRUE = 1, FALSE = 0} bool;

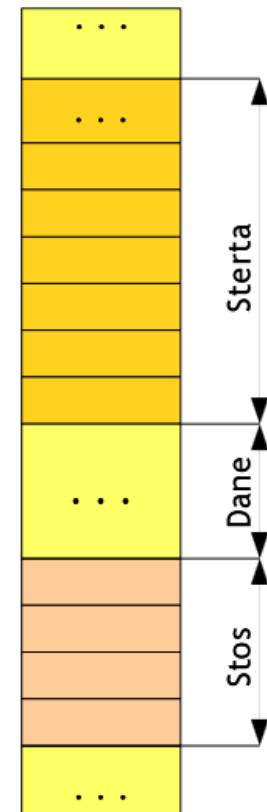
int main() {
    char c;
    bool dig;

    printf("Podaj znak z klawiatury, a powiem, czy to cyfra\n");
    scanf("%c", &c);
    dig = isdigit(c) == 0 ? FALSE : TRUE;
    if (dig == TRUE)
        printf("Podana zostala cyfra\n");
    else
        printf("Podany zostal znak niebedacy cyfra\n");

    return 0;
}
```

Dynamiczna alokacja pamięci

- ▶ Dynamiczna alokacja pamięci jest procesem, w którym pamięć jest przydzielana w trakcie działania programu, zgodnie z jego potrzebami.
- ▶ **Sterta**(ang. heap) wydzielony obszar pamięci wolnej:
 - ▶ przeznaczony do przechowywania danych dynamicznych,
 - ▶ kontrolowany przez programistę,
 - ▶ ograniczony pod względem rozmiaru,
 - ▶ przydzielany pasującymi fragmentami.
- ▶ **Stos** (ang. stack) wydzielony obszar pamięci roboczej:
 - ▶ przeznaczony do przechowywania danych automatycznych,
 - ▶ nie jest bezpośrednio kontrolowany przez programistę,
 - ▶ ograniczony pod względem rozmiaru,
 - ▶ przydzielany wg. zasady LIFO (ang. *last in, first out*).



Dynamiczna alokacja pamięci

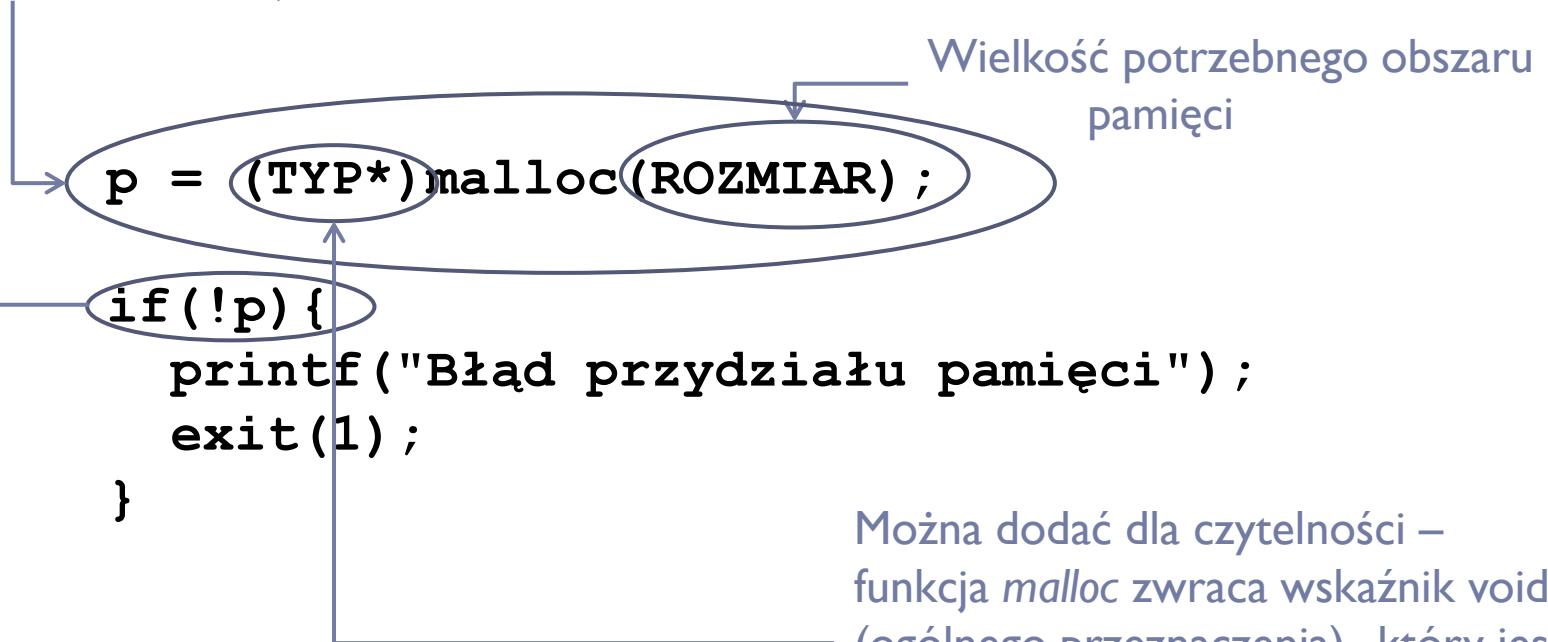
- ▶ Za dynamiczną alokację pamięci odpowiadają funkcje z pliku nagłówkowego `<stdlib.h>`

```
void *malloc(size_t size);
void *calloc(size_t nmeb, size_t size);
```

- ▶ `malloc()` przydziela pamięć o wielkości `size` bajtów
- ▶ `calloc()` przydziela pamięć dla `nmeb` elementów o rozmiarze `size` każdy i zeruje przydzieloną pamięć (przypisuje wszystkim bitom bloku pamięci wartość 0)
- ▶ wartość zwracana:
 - ▶ **w przypadku powodzenia:** wskaźnik do początku nowo przydzielonego bloku pamięci
 - ▶ **w przypadku niepowodzenia:** NULL i odpowiedni kod błędu wpisywany do zmiennej `errno`

Dynamiczna alokacja pamięci

Przydziął pamięci i zapamiętanie adresu początku tego obszaru w zmiennej wskaźnikowej



→ Sprawdzenie, czy przydział pamięci się powiodł

Dynamiczna alokacja pamięci

▶ Alternatywnie

```
TYP *p = malloc(ROZMIAR) ;
```



```
TYP *p = malloc (sizeof(TYP[ROZMIAR])) ;
```



```
TYP *p = malloc (ROZMIAR * sizeof *p) ;
```

Przykład 38

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

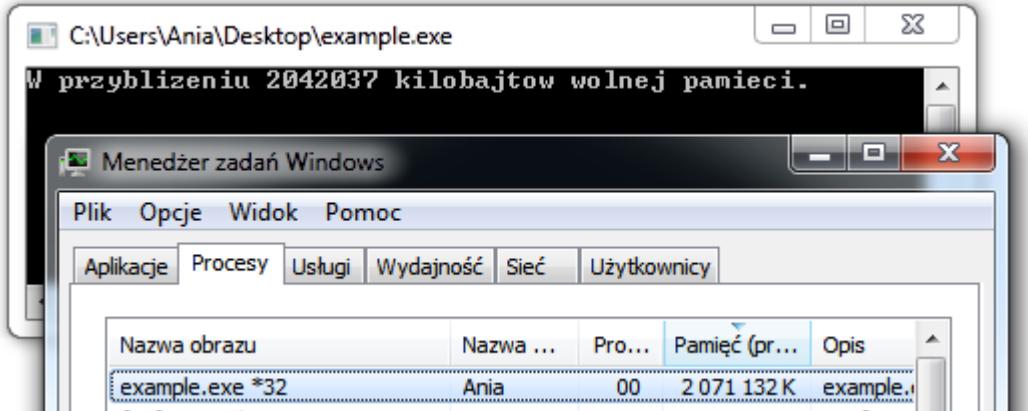
int main(void)
{
    char *p;
    long l;

    l=0;

    do{
        p = (char*)malloc(1024);
        if(p) l++;
    }while(p);

    printf("W przyblizeniu %ld kilobajtow wolnej pamieci.",l);

    return 0;
}
```



The screenshot shows two windows. The top window is titled 'C:\Users\Ania\Desktop\example.exe' and displays the text 'W przyblizeniu 2042037 kilobajtow wolnej pamieci.' (In the vicinity of 2042037 kilobytes of free memory). The bottom window is the Windows Task Manager titled 'Menedżer zadań Windows' (Windows Task Manager), showing the process 'example.exe *32' with details: Nazwa obrazu (Image Name) - example.exe, Nazwa ... (Name) - Ania, Pro... (Processor) - 00, Pamięć (pr... (Memory) - 2 071 132 K, Opis (Description) - example.exe. The 'Procesy' (Processes) tab is selected.

Dynamiczna alokacja pamięci

```
double *wsk1;           //alokacja pamięci dla 30 elementów typu double  
wsk1 = (double*) malloc(30*sizeof(double));
```

```
char *wsk2;           //alokacja pamięci dla n elementów typu char  
int n = 100;  
wsk2 = (char*) malloc(n*sizeof(char));
```

```
struct typ_s{           //alokacja pamięci dla zmiennej strukturalnej typ_s  
    int i;  
    char c;  
    double d;  
};  
  
struct typ_s *wsk3;  
wsk3 = (struct typ_s*) malloc(sizeof(struct typ_s));
```

Dynamiczna alokacja pamięci

```
int length = 10; //alokacja pamięci dla 10 elementów typu float
float * largeVec = malloc (sizeof(float[length]));
```

```
union sth_t{           //alokacja pamięci dla 5 unii sth_t
    char str[4];
    unsigned num;
};

union sth_t * largeVec;
largeVec = malloc (5 * sizeof *largeVec);
```

Zmiana rozmiaru zaalokowanej pamięci

- ▶ Za relokację (zmianę rozmiaru) pamięci wskazywanej przez wskaźnik odpowiada funkcja `realloc` z pliku nagłówkowego `<stdlib.h>`

```
void *realloc(void *ptr, size_t size);
```

- ▶ zmienia rozmiar przydzielonego wcześniej bloku pamięci wskazywanego przez `ptr` do `size` bajtów zaalokowanego przez `malloc()` lub `calloc()` (docelowo nowy blok może być pod innym adresem).
- ▶ pierwsze `n` bajtów bloku nie ulegnie zmianie gdzie `n` jest minimum z rozmiaru starego bloku `size`.
- ▶ Jeżeli `ptr` jest równy zero (tj. `NULL`), funkcja zachowuje się tak samo jako `malloc`.
- ▶ Wartość zwracana
 - ▶ **w przypadku powodzenia:** wskaźnik do początku nowo przydzielonego bloku pamięci
 - ▶ **w przypadku niepowodzenia:** `NULL` i odpowiedni kod błędu wpisywany do zmiennej `errno` (zawartość org. bufora jest zachowana)

Zwolnienie pamięci

- ▶ Za zwolnienie bloku pamięci przydzielonej dynamicznie odpowiada funkcja `free` z pliku nagłówkowego `<stdlib.h>`

```
void free(void *ptr);
```

- ▶ Funkcja `free()` zwalnia blok pamięci wskazywany przez `ptr` wcześniej przydzielony przez jedną z funkcji `malloc()`, `calloc()` lub `realloc()` lub `aligned_alloc()`.
- ▶ Jeżeli `ptr` ma wartość `NULL` funkcja nie robi nic.
- ▶ **Zwolnienie pamięci funkcją `free()` powinno towarzyszyć każdemu wywołaniu funkcji `malloc()` lub `calloc()`**

```
double *wsk1;  
wsk1 = (double*) malloc(30*sizeof(double));  
wsk1 = NULL; //BŁĄD - pamięć nie została zwolniona !
```

Przykład 39a

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

int main(){
    int *tab;
    int rozmiar;
    int i;

    printf("Podaj rozmiar tablicy: ");
    scanf("%d", &rozmiar);

    tab = (int*)malloc(rozmiar * sizeof(int));

    if(tab == NULL){
        printf("Przydziały pamięci się nie powiodł");
        exit(0);
    }

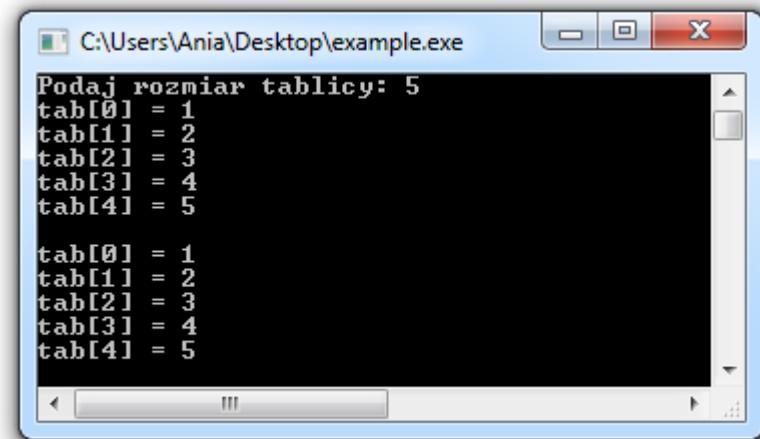
    for(i=0; i<rozmiar; i++){
        printf("tab[%d] = ", i);
        scanf("%d", &tab[i]);
    }

    printf("\n");

    for(i=0; i<rozmiar; i++)
        printf("tab[%d] = %d\n", i, tab[i]);

    free(tab);

    return 0;
}
```



Przykład 39b

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

int main(){
    int *tab;
    int rozmiar;
    int i;

    printf("Podaj rozmiar tablicy: ");
    scanf("%d", &rozmiar);

    tab = (int*)malloc(rozmiar * sizeof(int));

    if(tab == NULL){
        printf("Przydziały pamięci się nie powiodł");
        exit(0);
    }

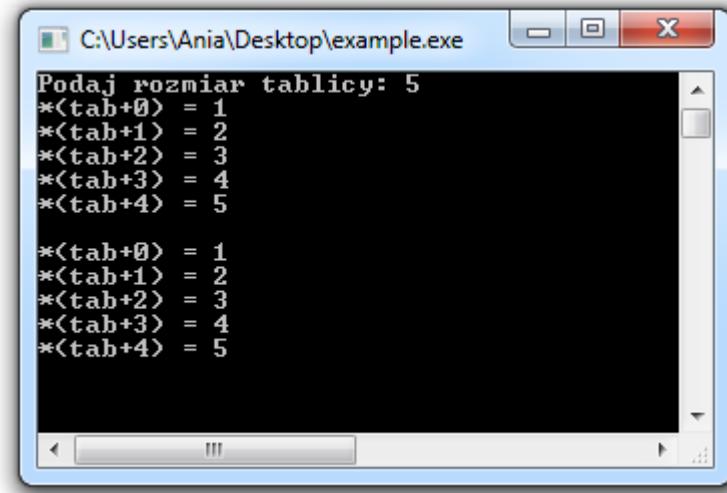
    for(i=0; i<rozmiar; i++){
        printf("* (tab+%d) = ", i);
        scanf("%d", (tab+i));
    }

    printf("\n");

    for(i=0; i<rozmiar; i++)
        printf("* (tab+%d) = %d\n", i, *(tab+i));

    free(tab);
}

return 0;
}
```



Znaczenie zwalniania pamięci

- ▶ Ilość pamięci statycznej jest znana na etapie komplikacji i nie zmienia się w czasie działania programu.
- ▶ Ilość pamięci poświęcanej zmiennym automatycznym zmienia się „samoczynnie” podczas działania programu.
- ▶ Ilość pamięci przydzielanej funkcją *malloc()* lub *calloc()* rośnie, jeśli nie jest zwalniana funkcją *free()*
 - ▶ **Brak zwalniania pamięci zaallokowanej dynamicznie powoduje tzw. wycieki pamięci (ang. memory leak).**

Przykład 40

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

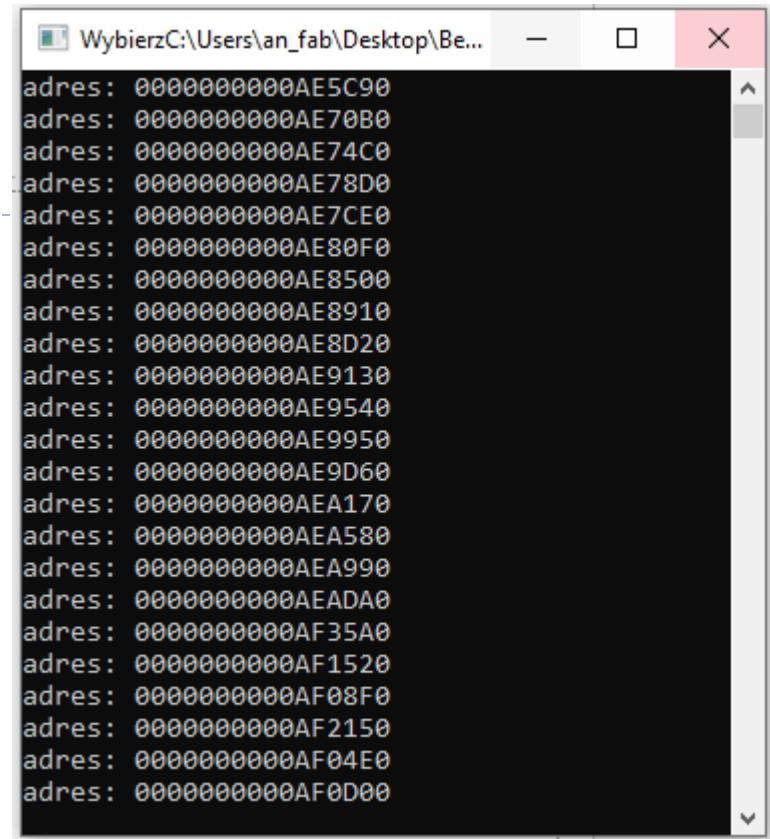
int main(void)
{
    char *p;
    long l;

    l=0;

    do{
        p = (char*)malloc(1024);
        if(p) l++;
        printf("adres: %p\n",p);
    }while(p);

    printf("W przyblizeniu %ld kilobajtow wolnej pamieci.",l);

    return 0;
}
```



The screenshot shows a Windows command-line interface window titled "WybierzC:\Users\an_fab\Desktop\Be...". It displays a list of memory addresses starting with "adres: 0000000000AE5C90" and continuing down to "adres: 0000000000AF0D00". The window has standard window controls (minimize, maximize, close) and a scroll bar on the right.

adres:
0000000000AE5C90
0000000000AE70B0
0000000000AE74C0
0000000000AE78D0
0000000000AE7CE0
0000000000AE80F0
0000000000AE8500
0000000000AE8910
0000000000AE8D20
0000000000AE9130
0000000000AE9540
0000000000AE9950
0000000000AE9D60
0000000000AEA170
0000000000AEA580
0000000000AEA990
0000000000AEADA0
0000000000AF35A0
0000000000AF1520
0000000000AF08F0
0000000000AF2150
0000000000AF04E0
0000000000AF0D00

Przykład 41a

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

void alokuj (int);

int main(void) {

    int i;

    for(i=0; i<1000; i++) {
        alokuj(2000);
    }

    return 0;
}

void alokuj (int n) {

    double *wsk;
    wsk = malloc(n*sizeof(double));
    //wyciek - pamięć alokowana, lecz nie zwalniana!
    //wskaźnik do bloku pamięci (zmienna automatyczna) jest niszczyony po
    //zakończeniu funkcji, ale pamięć pozostaje niezwolniona i brak do niej dostępu
```

Menedżer zadań Windows					
Aplikacje	Procesy	Usługi	Wydajność	Sieć	Użytkownicy
Nazwa obrazu	Nazwa ...	Pro...	Pamięć (pr...	Opis	
po 1 iteracji:					
example.exe *32	Ania	00	344 K	example.exe	
po 10 iteracjach:					
example.exe *32	Ania	00	412 K	example.exe	
po 50 iteracjach:					
example.exe *32	Ania	00	684 K	example.exe	
po 100 iteracjach:					
example.exe *32	Ania	00	1 084 K	example.exe	
po 500 iteracjach:					
example.exe *32	Ania	00	4 116 K	example.exe	
po 1000 iteracjach:					
example.exe *32	Ania	00	7 608 K	example.exe	

Przykład 41b

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

void alokuj (int);

int main(void) {

    int i;

    for(i=0; i<1000; i++) {
        alokuj(2000);
    }

    return 0;
}

void alokuj (int n) {

    double *wsk;
    wsk = malloc(n*sizeof(double));
    free(wsk);
}
```

Menedżer zadań Windows					
Aplikacje	Procesy	Usługi	Wydajność	Sieć	Użytkownicy
Nazwa obrazu	Nazwa ...	Pro...	Pamięć (pr...	Opis	
example.exe *32	Ania	00	344 K	example.exe	po 1 iteracji:
example.exe *32	Ania	00	344 K	example.exe	po 10 iteracjach:
example.exe *32	Ania	00	344 K	example.exe	po 50 iteracjach:
example.exe *32	Ania	00	344 K	example.exe	po 100 iteracjach:
example.exe *32	Ania	00	344 K	example.exe	po 500 iteracjach:
example.exe *32	Ania	00	344 K	example.exe	po 1000 iteracjach:

Częste błędy zarządzania pamięcią (1)

▶ Niezainicjalizowany wskaźnik (wild pointer)

```
int main(void)
{
    int *p;
    *p = 0;
}
```

Błąd!

```
int main(void)
{
    int *p = (int*)malloc(sizeof(int));
    *p = 0;
}
```

Poprawnie

▶ Błędnie zainicjalizowany wskaźnik (wild pointer)

```
int main(void)
{
    int *p, a;
    *p = (int*)0xABCDDEF01;
    p = 100;
    a = *p;
}
```

prawdopodobnie błędne adresy

Częste błędy zarządzania pamięcią (2)

▶ Dereferencja wskaźnika NULL

```
int main(void)
{
    int *p = NULL;
    printf("*p=%d", *p);

    return 0;
}
```

Błąd!

Niezdefiniowane zachowanie programu
Prawdopodobne przerwanie wykonania
programu oraz
segmentation fault/memory access violation

```
int main(void)
{
    int *p = NULL;

    if(p!=NULL)
        printf("*p=%d", *p);

    return 0;
}
```

Poprawnie

Częste błędy zarządzania pamięcią (3)

- ▶ Próba zapisu do pamięci „tylko do odczytu”

```
int main(void)
{
    char *s = "hello world";
    *s = 'H';
}
```

Błąd!

```
int main(void)
{
    char s[] = "hello world";
    *s = 'H';
}
```

Poprawnie

```
int main(void)
{
    char *s = (char*) malloc (20*sizeof(char));
    strcpy(s, "hello world");
    *s = 'H';
    free(s);
}
```

Poprawnie

Częste błędy zarządzania pamięcią (4)

- ▶ Dangling pointer – wskazywana zmienna przestaje istnieć

```
int main(void)
{
    int *p = NULL;
    {
        int a;
        p = &a;
    }
    *p = 0;
}
```

zmienna a tu już nie istnieje

Błąd!

Częste błędy zarządzania pamięcią (5)

- ▶ Dangling pointer – funkcja zwraca adres zmiennej lokalnej

```
int *func(void)
{
    int num = 1234;
    return &num;
}

...
printf("%d", *func());
```

Błąd!

```
int *func(void)
{
    static int num = 1234;
    return &num;
}

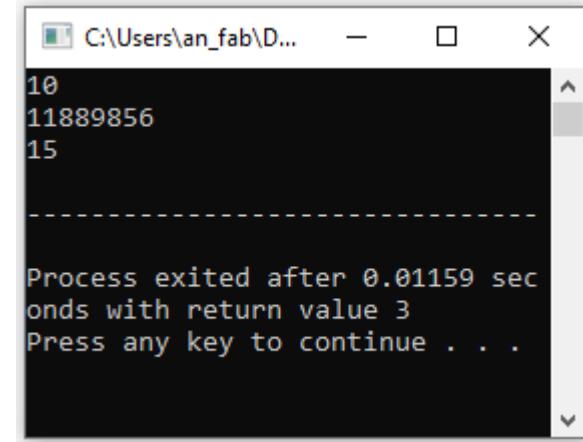
...
printf("%d", *func());
```

Poprawnie

Częste błędy zarządzania pamięcią (6)

- Dangling pointer – wskaźnik do zwolnionego bloku pamięci

```
int main() {  
    int *ptr = (int *)malloc(sizeof(int));  
    *ptr = 10;  
    printf("%d\n", *ptr);  
    free(ptr);  
    printf("%d\n", *ptr); //Błąd!  
    *ptr = 15;           //Błąd!  
    printf("%d\n", *ptr); //Błąd!  
    free(ptr);          //Błąd!  
  
    // No more a dangling pointer  
    ptr = NULL;  
  
    if (ptr!=NULL) {  
        *ptr = 15;  
        printf("%d\n", *ptr);  
    }  
}
```



Błąd !!!!
(program mimo to się wykonał...)

Częste błędy zarządzania pamięcią (7)

- Dangling pointer – kilka wskaźników na jeden obszar pamięci

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

int main() {
    int *ptr = malloc(sizeof(int));
    int *ptr2 = ptr, *ptr3 = ptr;
    *ptr = 10;
    printf("%d\n", *ptr);
    printf("%d\n", *ptr2);
    printf("%d\n", *ptr3);
    ptr = NULL;
    printf("*****\n");
    printf("%d\n", *ptr); //błąd
    printf("%d\n", *ptr2); //błąd
    printf("%d\n", *ptr3); // błąd
}
```

```
10
10
10
*****
-2144186136
-2144186136
-2144186136

Process finished with exit code 0
```

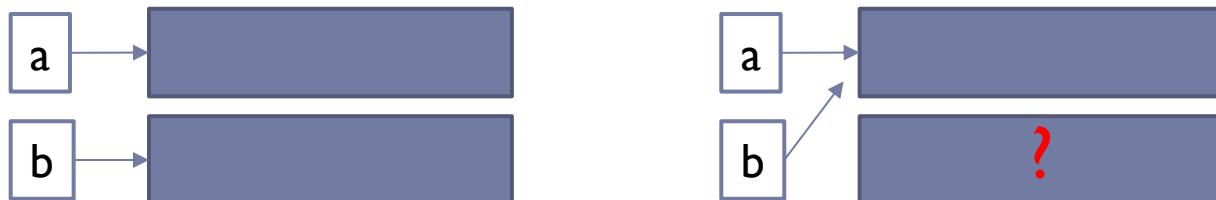
```
▶ clang-7 -pthread -lm -o main main.c
▶ ./main
10
10
10
*****
exited, segmentation fault
▶ █
```

Częste błędy zarządzania pamięcią (8)

- Dangling pointer – przestawienie wskaźnika

```
char *a = malloc(128*sizeof(char));  
char *b = malloc(128*sizeof(char));  
b = a;           // wyciek pamięci pod b  
free(a);  
free(b);         //próba ponownego zwolnienia pamięci pod a
```

Błąd!



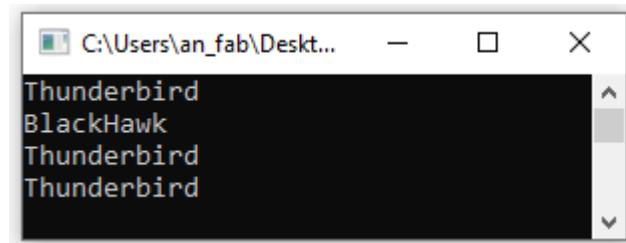
```
char *a = malloc(128*sizeof(char));  
char *b = malloc(128*sizeof(char));  
free(b);  
b = a;  
free(a); // lub free(b) bo wskazują to samo;
```

Poprawnie

Częste błędy zarządzania pamięcią (9)

▶ płytka kopia vs. głęboka kopia (shallow vs. deep copy)

```
int main() {
    Aircraft af1, af2, af3;
    af1.model = (char*)malloc(strlen("Thunderbird") + 1);
    strcpy(af1.model, "Thunderbird");
    af1.capacity = 320;
    af2 = af1; // płytka kopia
    printf("%s\n", af1.model);
    strcpy(af2.model, "BlackHawk");
    printf("%s\n", af1.model);
    af3.model = (char*)malloc(strlen("Thunderbird") + 1);
    strcpy(af3.model, af1.model); // głęboka kopia
    af3.capacity = af1.capacity; // głęboka kopia
    strcpy(af1.model, "Thunderbird");
    printf("%s\n", af1.model);
    strcpy(af3.model, "BlackHawk");
    printf("%s\n", af1.model);
    free(af1.model);
    free(af3.model);
    return 0;
}
```



```
typedef struct {
    char *model;
    int capacity;
} Aircraft;
```

Wskaźniki a tablice wielowymiarowe

- ▶ W języku C dwuwymiarowa tablica jest tablicą jednowymiarową, w której każdy element jest tablicą

```
int zippo[4][2]; /*tablica tablic typu int*/
```

$$\text{zippo} \Leftrightarrow \&\text{zippo}[0] \Leftrightarrow \&\text{zippo}[0][0]$$

- ▶ **zippo+1** przesunięcie do drugiego wiersza
- ▶ **zippo[0]+1** przesunięcie do drugiej komórki w pierwszym wierszu
- ▶ *** (zippo[0])** – wartość elementu **zippo[0][0]**
- ▶ ***zippo** $\Leftrightarrow \&zippo[0][0]$ stąd ***&zippo[0][0]** $\Leftrightarrow \ast\ast zippo$

$$\text{zippo}[2][1] \Leftrightarrow \ast(\ast(zippo+2)+1)$$

Wskaźniki a tablice wielowymiarowe

- ▶ Każdy element tablicy dwuwymiarowej jest tablicą, a tym samym adresem pierwszego elementu, stąd:

$$\text{zippo}[0] \Leftrightarrow \&\text{zippo}[0][0] \Leftrightarrow *\text{zippo}$$
$$\text{zippo}[i] \Leftrightarrow \&\text{zippo}[i][0] \Leftrightarrow *(\text{zippo}+i)$$

- ▶ Zastosowanie operatora `*` do ww. wyrażeń daje następujący wynik:

$$*\text{zippo}[0] \Leftrightarrow \text{zippo}[0][0] \Leftrightarrow **\text{zippo}$$
$$*\text{zippo}[i] \Leftrightarrow \text{zippo}[i][0] \Leftrightarrow *(*(\text{zippo}+i))$$
$$\text{zippo}[m][n] \Leftrightarrow *(*(\text{zippo}+m)+n)$$

Wskaźniki do tablic dwuwymiarowych

- ▶ Wskaźnik do tablicy dwuwymiarowej:

TYP (*nazwa)[rozmiar]

```
int tab[4][2] = {{1,2},{2,3},{3,4},{4,5}};  
int(*pz)[2];  
pz = tab;
```

- ▶ Tablica wskaźników:

TYP *nazwa[rozmiar]

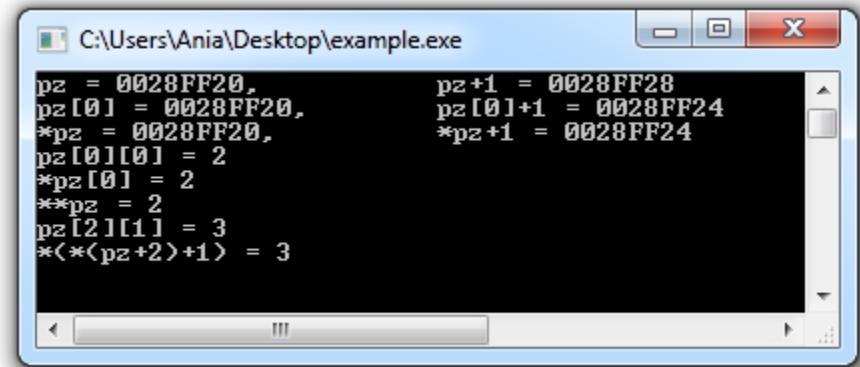
Przykład 42

```
#include <stdio.h>

int main(void)
{
    int zippo[4][2] = {{2, 4}, {6, 8}, {1, 3}, {5, 7}};
    int (*pz)[2];
    pz = zippo;

    printf("pz = %p, \t\t pz+1 = %p\n", pz, pz+1);
    printf("pz[0] = %p, \t pz[0]+1 = %p\n", pz[0], pz[0]+1);
    printf("*pz = %p, \t *pz+1 = %p\n", *pz, *pz+1);
    printf("pz[0][0] = %d\n", pz[0][0]);
    printf("*pz[0] = %d\n", *pz[0]);
    printf("**pz = %d\n", **pz);
    printf("pz[2][1] = %d\n", pz[2][1]);
    printf("*(*(pz+2)+1) = %d\n", *(*(pz+2)+1));

    return 0;
}
```



Wskaźniki a tablice wielowymiarowe

```
int a[10][20];  
//a[3][4] - ok
```



przygotowanych zostało 200 lokalizacji o rozmiarze **int**
stała długość wiersza

```
int *b[10];  
// b[3][4] - ok
```

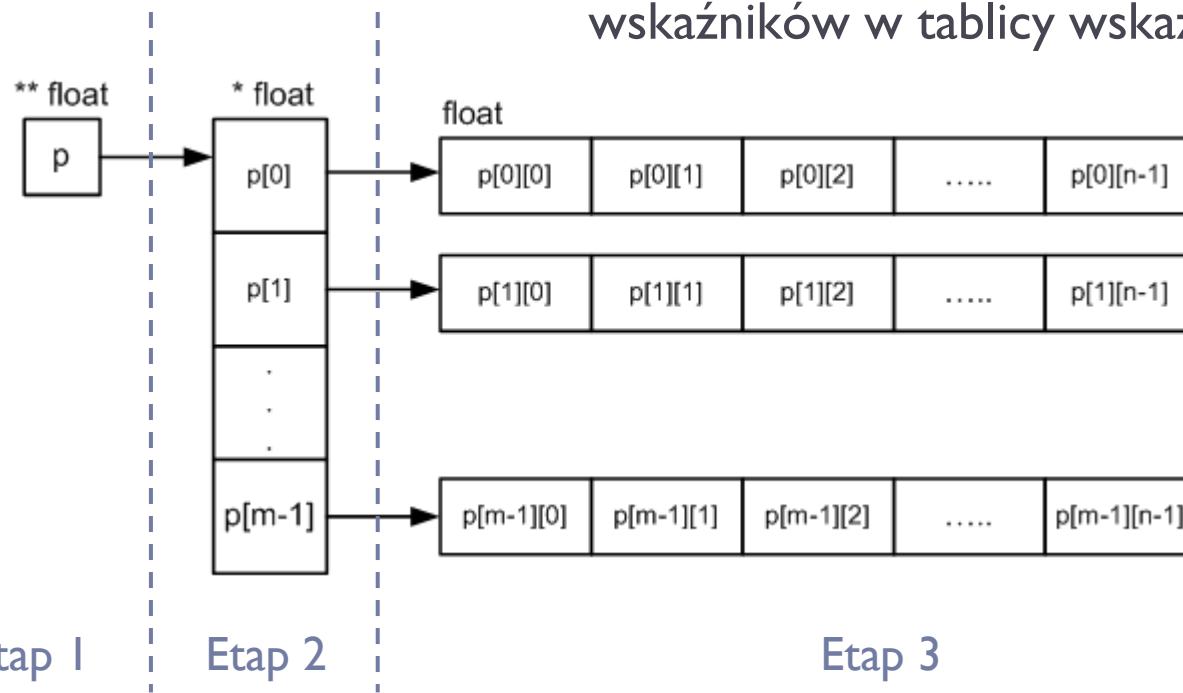


zaalokowano 10 wskaźników **int**
inicjalizacja musi nastąpić jawnie
wiersze mogą mieć różną długość

Dynamiczne tablice dwuwymiarowe

▶ Alokacja tablicy dwuwymiarowej

- ▶ **Etap 1:** deklaracja wskaźnika do tablicy wskaźników
- ▶ **Etap 2:** alokacja tablicy wskaźników (do poszczególnych wierszy)
- ▶ **Etap 3:** alokacja poszczególnych wierszy (i ich „dowiązanie” do wskaźników w tablicy wskaźników)



Dynamiczne tablice dwuwymiarowe

▶ Alokacja tablicy dwuwymiarowej

- ▶ **Etap 1:** deklaracja wskaźnika do tablicy wskaźników
- ▶ **Etap 2:** alokacja tablicy wskaźników (do poszczególnych wierszy)
- ▶ **Etap 3:** alokacja poszczególnych wierszy (i ich „dowiązanie” do wskaźników w tablicy wskaźników)

```
// Etap 1:
```

```
TYP ** tab = NULL;  
int w;
```

```
// Etap 2:
```

```
tab = (TYP **)calloc(liczba_wierszy, sizeof(TYP *));
```

```
// Etap 3:
```

```
for (w=0; w<liczba_wierszy; w++)  
*(tab+w) = (TYP*) calloc(liczba_kolumn, sizeof(TYP));
```

Dynamiczne tablice dwuwymiarowe

- ▶ **Zwolnienie pamięci zaalokowanej dla tablicy dwuwymiarowej:**
 - ▶ **Etap 1:** zwolnienie pamięci zaalokowanej dla poszczególnych wierszy
 - ▶ **Etap 2:** zwolnienie pamięci zaalokowanej dla tablicy wskaźników do poszczególnych wierszy

// Etap 1:

```
for (w=0; w<liczba_wierszy; w++)  
    free(*(tab+w));
```

// Etap 2:

```
free(tab);
```

```
tab = NULL;
```

Przykład 43₍₁₎

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

/***** alokuj *****/
float ** alokuj(int, int);
void wypelnij(float**, int, int);
void drukuj(float**, int, int);
void zwolnij(float ***, int);

/***** main *****/
int main(){

    int w = 40;      //liczba wierszy
    int k = 60;      //liczba kolumn

    float ** tab = alokuj(w, k);
    wypelnij(tab, w, k);
    drukuj(tab, w, k);
    zwolnij(&tab, w);

    return 0;
}
```

Funkcja zwracająca wskaźnik do tablicy 2D

Przykład 43(2)

```
float ** alokuj(int wiersze, int kolumny) {  
  
    float ** tablica = NULL;  
    int w, k;  
  
    tablica = (float **)calloc(wiersze, sizeof(float *));  
  
    for (w=0; w<wiersze; w++)  
        *(tablica+w) = (float*) calloc(kolumny, sizeof(float));  
  
    return tablica;  
}  
*****
```

```
void drukuj(float** tablica, int wiersze, int kolumny) {  
  
    int w, k;  
    for (w=0; w<wiersze; w++) {  
        for (k=0; k<kolumny; k++) {  
            printf("%.2f ", *(*(tablica+w)+k));  
        }  
        printf("\n");  
    }  
  
    return;  
}
```

Przykład 43(3)

Wskaźnik do wskaźnika do tablicy 2D
Dlaczego tak ?

```
void zwolnij(float ***tablica, int wiersze) {
    int w;

    for (w=0; w<wiersze; w++) {
        free(*((*tablica)+w));
    }

    free(*tablica);

    *tablica = NULL;

    return;
}
/********************************************/

void wypelnij(float** tablica, int wiersze, int kolumny) {
    int w, k;

    for (w=0; w<wiersze; w++) {
        for(k=0; k<kolumny; k++) {
            *(*(tablica+w)+k) = 1.0/(1+rand()%10);
        }
    }

    return;
}
```

Wskaźnik do wskaźnika

- ▶ Stosowane, np. gdy funkcja ma zmodyfikować wskaźnik

```
int alokuj1Dv1(float* tablica, int num){  
    tablica = calloc(num, sizeof(float));  
    printf("Adres w funkcji: %p",tablica);  
    return 1;  
}
```

```
int alokuj1Dv2(float** tablica, int num){  
    *tablica = calloc(num,sizeof(float));  
    printf("Adres w funkcji: %p",*tablica);  
    return 1;  
}
```

```
float *a=NULL, *b=NULL;  
  
printf(" --- Wskaźnik a ---\n");  
printf("Adres pod wskaźnikiem przed funkcją: %p",a);  
alokuj1Dv1(a, 10); //kopia a + wyciek pamięci  
printf("Adres pod wskaźnikiem po funkcji: %p", a);  
free(a);  
  
printf(" --- Wskaźnik b ---\n");  
printf("Adres pod wskaźnikiem przed funkcją: %p" b);  
alokuj1Dv2(&b, 10);  
printf("Adres pod wskaźnikiem po funkcji: %p", b);  
free(b);
```

```
--- Wskaźnik a ---  
Adres pod wskaźnikiem przed funkcją: 0x0  
Adres w funkcji: 0x8000389a0  
Adres pod wskaźnikiem po funkcji: 0x0  
--- Wskaźnik b ---  
Adres pod wskaźnikiem przed funkcją: 0x0  
Adres w funkcji: 0x8000389d0  
Adres pod wskaźnikiem po funkcji: 0x8000389d0
```

Przykład 44(1)

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

#define W 3
#define K 3
```

```
float* alokuje1D(int w, int k); //alokuje pamięć 1D
float** alokuje2D(int w, int k); //alokuje pamięć 2D (por. przykład 25)

void drukuj2D_ver1(float(*)[K]); //drukuję tablice (statyczną) 2D
void drukuj2D_ver2(float** tab); //drukuję tablice (dynamiczną) 2D

void drukuj1D(float* tab); //drukuję tablicę 1D
```

```
float * alokuje1D(int w, int k) {
    float * tablica = NULL;
    tablica=(float *)calloc(w*k,sizeof(float));
    return tablica;
}
```

Przykład 44(2)

```
int main() {
    int x, y, i;
    float* tab1 = alokuje1D(W, K);
    float** tab2 = alokuje2D(W, K);
    float tab11 [W*K] = {1,2,3,4,5,6,7,8,9};
    float tab22 [W][K] = {{1,2,3},{4,5,6},{7,8,9}};

    for(y=0; y<W; y++) {
        for(x=0; x<K; x++) {
            i = x + K * y;
            *(tab1+i) = *(*(tab22+y)+x);
        }
    }

    for(i=0; i<W*K; i++) {
        x = i % W;
        y = i / W;
        *(*(tab2+y)+x) = *(tab11+i);
    }
}
```

2D jak 1D

ID jak 2D

```
** drukuj2D_ver1 **
1 2 3
4 5 6
7 8 9

** drukuj2D_ver2 **
1 2 3
4 5 6
7 8 9

** drukuj1D **
1 2 3
4 5 6
7 8 9

** drukuj1D **
1 2 3
4 5 6
7 8 9
```

```
drukuj2D_ver1(tab22);
//drukuj2D_ver1(tab2); //błąd
//drukuj2D_ver2(tab11); //błąd
drukuj2D_ver2(tab2);
drukuj1D(tab1);
drukuj1D(tab11);
drukuj1D((float*)tab22);
//drukuj1D((float*)tab2); //błąd
```

+ zwolnienie pamięci

Przykład 44(3)

```
void drukuj2D_ver1(float(*tab) [K]) {
    printf("\n** drukuj2D_ver1 **\n");
    int i, j;
    for(i=0; i<W; i++) {
        for(j=0; j<K; j++) {
            printf("%.0f ", tab[i][j]);
        }
        printf("\n");
    }
}
```

```
void drukuj2D_ver2(float**tab) {
    printf("\n** drukuj2D_ver2 **\n");
    int i, j;
    for(i=0; i<W; i++) {
        for(j=0; j<K; j++) {
            printf("%.0f ", tab[i][j]);
        }
        printf("\n");
    }
}
```

```
void drukuj1D(float* tab) {
    printf("\n** drukuj1D **\n");
    int i, x, y;

    for(i=0; i<W*K; i++) {
        x = i % W;
        y = i / W;
        printf("%.0f ", tab[i]);
        if (x == K-1)
            printf("\n");
    }
}
```

Alokacja pamięci

- ▶ Łatwiejsze użycie tablic 2D (podejście mieszane)

```
int ar2[5][6];
int (* p2)[6];
p2 = (int (*)[6]) malloc(n*6*sizeof(int)); //n*tablic 6-elem.
ar2[1][2] = p2[1][2] = 12;
free(p2)
```

p2 pokazuje na tablicę 6 elementów typu int
p2[i] jest tablicą 6 elementów typu int
p2[i][j] jest wartością typu int

Dynamiczne struktury

- ▶ Język C pozwala na zaalokowanie pamięci dla dowolnego typu danych; w programach można więc w dynamiczny sposób alokować również struktury.

```
struct typ_s{           //alokacja pamięci dla zmiennej strukturalnej typ_s
    int i;
    char c;
    double d;
};

struct typ_s *wsk;
wsk = (struct typ_s*) malloc(sizeof(struct typ_s ));
```

- ▶ Ww. mechanizm jest przydatny w przypadku funkcji, modyfikujących struktury

Dynamiczne unie

- ▶ Język C pozwala na zaalokowanie pamięci dla dowolnego typu danych; w programach można więc w dynamiczny sposób alokować również unie.

```
union sth_t{ //alokacja pamięci dla unii sth_t  
    char str[4];  
    unsigned num;  
};  
  
union sth_t * largeVec  
largeVec = (union sth_t)malloc(5*sizeof(union sth_t));
```

Przykład 45a(1)

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

/***********************/

struct typ_s{
    int i;
    char c;
    double d;
};

/***********************/

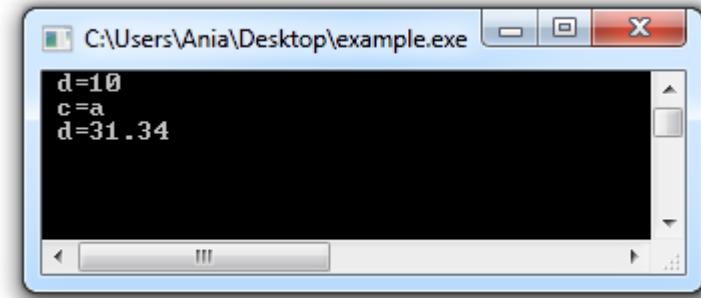
struct typ_s* wypełnij(int i, char c, double d);
void drukuj (struct typ_s s);

/***********************/

int main(){

    struct typ_s* wsk = wypełnij(10, 'a', 31.34);
    drukuj (*wsk);
    free(wsk);

    return 0;
}
```



Przykład 45a₍₂₎

```
*****  
struct typ_s* wypełnij(int i, char c, double d){  
  
    struct typ_s *wsk;  
    wsk = (struct typ_s*) malloc(sizeof(struct typ_s ));  
    wsk->i = i;  
    wsk->c = c;  
    wsk->d = d;  
  
    return wsk;  
}  
  
*****  
void drukuj (struct typ_s s){  
  
    printf(" d=%i\n c=%c\n d=% .2lf\n", s.i, s.c, s.d);  
    return;  
}  
  
*****
```

Przykład 45b(1)

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

/***********************/

struct typ_s{
    int i;
    char c;
    double d;
};

/***********************/

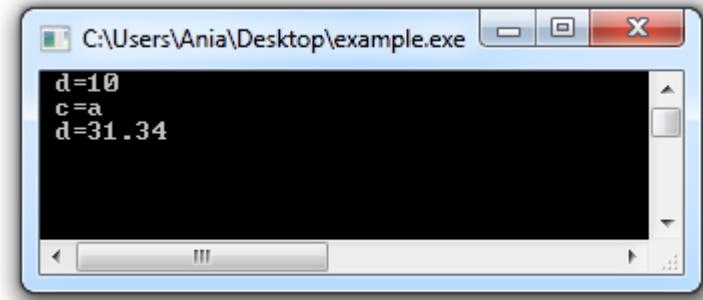
struct typ_s wypełnij(int i, char c, double d);
void drukuj (struct typ_s s);

/***********************/

int main(){

    struct typ_s wsk = wypełnij(10, 'a', 31.34);
    drukuj (wsk);

    return 0;
}
```



Przykład 45b₍₂₎

```
*****  
struct typ_s wypełnij(int i, char c, double d) {  
  
    struct typ_s wsk;  
    wsk.i = i;  
    wsk.c = c;  
    wsk.d = d;  
  
    return wsk;  
}  
*****
```

Nieefektywne

```
void drukuj (struct typ_s s){  
  
    printf(" d=%i\n c=%c\n d=%.2lf\n", s.i, s.c, s.d);  
    return;  
}  
*****
```

Przykład 46

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

int main() { //alokacja struktury
    struct namect *ptr = (struct namect*) malloc (sizeof(struct namect));
    if(!ptr)
        exit(EXIT_FAILURE);

    //alokacja pola wewnętrz struktury
    ptr->name = (char*)malloc(20*sizeof(char));
    if(!ptr->name) {
        free(ptr);
        exit(EXIT_FAILURE);
    }

    //kopiowanie do zaalokowanego bloku pamięci
    strcpy(ptr->name,"Jan Kowalski"); ptr->id = 101214;
    printf("name: %s, id: %d\n", ptr->name, ptr->id);

    //ważna kolejność, aby uniknąć wycieku pamięci
    free(ptr->name);
    free(ptr);
}
```

```
struct namect {
    char *name; // wskaźnik zamiast tablicy
    int id;
};
```

Przykład 47

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

union sth_t{
    char str[4];
    unsigned num;
};

int main(void) {
    union sth_t * largeVec = malloc (5 * sizeof * largeVec); //definicja unii

    for(int i = 0; i< 5; i++)
    {
        (largeVec+i)->num = 6513249+i; //zmiana pierwszego znaku
        printf("%s\n", (largeVec+i)->str);
    }

    free(largeVec);
    return 0;
}
```

```
abc
bbc
cbc
dbc
ebc

Process finished with exit code 0
```

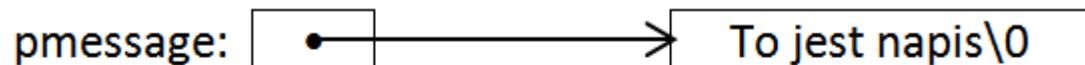
Wskaźniki znakowe

- ▶ Wskaźnik znakowy – wskaźnik wskazujący na stałą tekstową / tablicę znaków.
- ▶ Stała tekstowa: "**To jest napis**"

- ▶ tablica znaków zakończona znakiem pustym '\0,

```
char amessage[] = "To jest napis "; /*tablica*/
char *pmassage = "To jest napis" ; /*wskaźnik*/
```

- ▶ Przypisanie do wskaźnika stałej znakowej **nie jest** kopiowaniem ciągu znakowego ani nie powoduje alokacji pamięci
pmassage – tylko do odczytu!



Przykład 48

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

void mojaputs(char *p);

int main(void) {
    char napis1[] = "napis testowy 1";
    char *napis2 = "napis testowy 2";
    char *napis3 = malloc(50);
    //napis3 = "napis testowy 3;" // błąd - wyciek pamięci!
    strcpy(napis3, "napis testowy 3");

    mojaputs("to tylko test");
    mojaputs(napis1);
    mojaputs(napis2);
    mojaputs(napis3);
    free(napis3);

    napis2 = "napis testowy 4"; //OK, przestawienie wskaźnika
    mojaputs(napis2);

    return 0;
}
```

```
void mojaputs(char *p) {
    while(*p) { //aż do napotkania NULL'a
        printf("%c", *p);
        p++; //przejdz do następnego znaku
    }
    printf("\n");
}
```

```
to tylko test
napis testowy 1
napis testowy 2
napis testowy 3
napis testowy 4
```

```
Process finished with exit code 0
```

Przykład 49

▶ Szukanie końca napisu / zliczanie znaków

```
size_t strlen(const char *s){  
    size_t n = 0;  
    for(; *s != '\0'; s++)  
        n++;  
    return n;  
}
```

```
size_t strlen(const char *s){  
    size_t n = 0;  
    for(; *s++;)  
        n++;  
    return n;  
}
```

```
size_t strlen(const char *s){  
    const char *p = s;  
    while(*s)  
        s++;  
    return s-p;  
}
```

```
size_t strlen(const char *s){  
    size_t n = 0;  
    for(; *s; s++)  
        n++;  
    return n;  
}
```

```
size_t strlen(const char *s){  
    size_t n = 0;  
    while(*s++)  
        n++;  
    return n;  
}
```

idiom: `while(*s)
 s++;`

idiom: `while(*s++) ;`

Przykład 50

▶ Kopiowanie napisu **t** do napisu **s**

Wersja 1 - tablicowa

```
void strcpy(char *s, char *t)
{
    int i = 0;

    while((s[i] = t[i]) != '\0')
        i++;
}
```

Wersja 2 - wskaźnikowa

```
void strcpy(char *s, char *t)
{
    while((*s = *t) != '\0')
    {
        s++;
        t++;
    }
}
```

Wersja 3 - wskaźnikowa

```
void strcpy(char *s, char *t)
{
    while(*s++ = *t++) != '\0';
}
```

Wersja 4 - wskaźnikowa

```
void strcpy(char *s, char *t)
{
    while(*s++ = *t++);
}
```

Wskaźniki i łańcuchy znakowe

▶ Kopiowanie/przypisywanie łańcuchów znakowych

```
#include <string.h>
...
char str1[10], str2[10];
char *ptr1, *ptr2;

str1 = "abc"; /*** BŁĄD - próba zmiany adresu ***/
str2 = str1; /*** BŁĄD - próba zmiany adresu ***/

ptr1 = "abc"; /*** OK - ustawienie wskaźnika na napis ***/
ptr1 = str1; /*** OK - ustawienie wskaźnika na tablicę znaków ***/

strcpy(str1, "abc"); /*** OK - kopowanie znak po znaku ***/
strcpy(str2, str1); /*** OK - kopowanie znak po znaku **/

strcpy(ptr2, "abc"); /*** BŁĄD - ptr2 na nic nie pokazuje ***/
ptr2 = (char*)malloc(10*sizeof(char)); /*** Ustawienie ptr2 ***/
strcpy(ptr2, "abc"); /*** OK - kopowanie napisu do bloku pamięci **/

free(ptr2);
```

Wskaźniki i łańcuchy znakowe

▶ Porównywanie łańcuchów znakowych

```
char str1[10]={"bbc"}, str2[10>{"cbb"};
char *ptr1, *ptr2;

ptr1 = "abc"; /** OK - ustawienie wskaźnika na napis */
ptr2 = str1; /** OK - ustawienie wskaźnika na tablicę znaków */

if(str1 == str2)
    printf("Porównanie adresów tablic, nie ich zawartości\n");

if(ptr2 == str1)
    printf("prawda, bo ptr2 pokazuje na str1\n");

if(!strcmp(str1,str2)){
    printf("OK - porównanie zawartości tablic\n");
    printf("ale fałsz, bo zawartości tablic są różne");
}

if(!strcmp(ptr1,"abc")) //potrzebne string.h
    printf("prawda, bo ptr1 ustawione na \"abc\"\n");
```

prawda, bo ptr2 pokazuje na str1
prawda, bo ptr1 ustawione na "abc"

Process finished with exit code 0

<string.h> funkcje operujące na napisach i blokach pamięci

int strcmp(const char* s1, const char* s2)

- ▶ porównuje napisy s1 oraz s2, zwraca wartość <0 , gdy s1 < s2, wartość >0 , gdy s1 > s2 i 0 dla identycznych tekstów,

int strncmp(const char* s1, char* s2, size_t n)

- ▶ działa analogicznie do strcmp() z tym, że porównuje nie więcej niż n znaków z podanych tekstów,

char* strcpy(char* s1, char* s2)

- ▶ kopiuje tekst z s2 do s1, dodatkowo zwracany jest wskaźnik do tekstu s1,

char* strncpy(char* s1, char* s2, size_t n)

- ▶ funkcja działa analogicznie do strcpy() z tym, że kopiuje co najwyżej n znaków z s2 do s1,

char* strcat(char* s1, char* s2)

- ▶ dołącza na końcu tekstu reprezentowanego przez s1 kopię tekstu s2, zwraca wskaźnik do połączonego ciągu znaków (s1),

char* strncat(char* s1, char* s2, size_t n)

- ▶ działa analogicznie do strcat() z tym, że dołącza do tekstu s1 co najwyżej n znaków z tekstu s2,

Przykład 51a

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

int main(void) {
    char * mesg = "Don't worry!";
    char * copy;
    copy = mesg;
    printf("%s\n", copy);
    printf("mesg = %s; &mesg = %p; value = %p\n", mesg, &mesg, mesg);
    printf("copy = %s; &copy = %p; value = %p\n", copy, &copy, copy);

    mesg = "Be happy!";
    printf("%s\n", copy);
    printf("mesg = %s; &mesg = %p; value = %p\n", mesg, &mesg, mesg);
    printf("copy = %s; &copy = %p; value = %p\n", copy, &copy, copy);

    return 0;
}
```

```
Don't worry!
mesg = Don't worry!; &mesg = 0xfffffcc38; value = 0x100403000
copy = Don't worry!; &copy = 0xfffffcc30; value = 0x100403000
Don't worry!
mesg = Be happy!; &mesg = 0xfffffcc38; value = 0x10040305b
copy = Don't worry!; &copy = 0xfffffcc30; value = 0x100403000

Process finished with exit code 0
```

Przykład 51b

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

int main(void) {
    char mesg[50] = {"Don't worry!"};
    char * copy;
    copy = mesg;
    printf("%s\n", copy);
    printf("mesg = %s; &mesg = %p; value = %p\n", mesg, &mesg, mesg);
    printf("copy = %s; &copy = %p; value = %p\n", copy, &copy, copy);

    strcpy(mesg, "Be happy!");
    printf("%s\n", copy);
    printf("mesg = %s; &mesg = %p; value = %p\n", mesg, &mesg, mesg);
    printf("copy = %s; &copy = %p; value = %p\n", copy, &copy, copy);

    return 0;
}
```

```
Don't worry!
mesg = Don't worry!; &mesg = 0xfffffcc00; value = 0xfffffcc00
copy = Don't worry!; &copy = 0xffffcbf8; value = 0xfffffcc00
Be happy!
mesg = Be happy!; &mesg = 0xfffffcc00; value = 0xfffffcc00
copy = Be happy!; &copy = 0xffffcbf8; value = 0xfffffcc00

Process finished with exit code 0
```

Tablice wskaźników do napisów

`char *tab[ROZMIAR];`

Tablica napisów

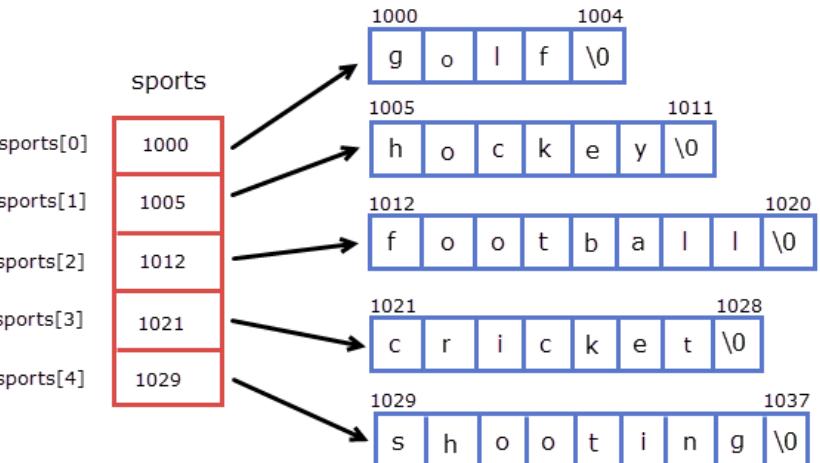
```
char sports[5][15] = {"golf",
                      "hockey",
                      "football",
                      "cricket",
                      "shooting"
};
```

1000	g	o	l	f	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0
1015	h	o	c	k	e	y	\0	\0	\0	\0	\0	\0	\0	\0	\0
1031	f	o	o	t	b	a	i	i	\0	\0	\0	\0	\0	\0	\0
1047	c	r	i	c	k	e	t	\0	\0	\0	\0	\0	\0	\0	\0
1063	s	h	o	o	t	i	n	g	\0	\0	\0	\0	\0	\0	\0
1079															

Vs.

Tablica wskaźników napisów

```
char *sports[5] = {"golf",
                    "hockey",
                    "football",
                    "cricket",
                    "shooting"
};
```



Przykład 52

```
#include <stdio.h>
int main(void) {
    const char *mytalents[5] = {
        "Adding numbers swiftly",
        "Multiplying accurately", "Stashing data",
        "Following instructions to the letter",
        "Understanding the C language"};
    char yourtalents[5][40] = {
        "Walking in a straight line",
        "Sleeping", "Watching television",
        "Mailing letters", "Reading email"};
    int i;
    puts("Let's compare talents.");
    printf ("%-36s %-25s\n", "My Talents", "Your Talents");
    for (i = 0; i < 5; i++)
        printf("%-36s %-25s\n", mytalents[i], yourtalents[i]);
    printf("\nsizeof mytalents: %zd, sizeof yourtalents: %zd\n",
          sizeof(mytalents), sizeof(yourtalents));
    return 0;
}
```

Let's compare talents.	Your Talents
My Talents	Walking in a straight line
Adding numbers swiftly	Sleeping
Multiplying accurately	Watching television
Stashing data	
Following instructions to the letter	Mailing letters
Understanding the C language	Reading email

sizeof mytalents: 40, sizeof yourtalents: 200

5 wskaźników każdy po 8 bajtów
vs.
5 napisów każdy po 40 bajtów

Tablice wskaźników do napisów

▶ Częste zastosowanie – tablice napisów

```
const char *p[] = {  
    "Wejście przekracza długość pola",  
    "Poza zakresem",  
    "Drukarka wyłączona",  
    "Brak papieru",  
    "Przepelenie dysku",  
    "Błąd zapisu na dysku"  
};
```

...

```
void blad(int num_bledu)  
{  
    printf(p[num_bledu]);  
}
```

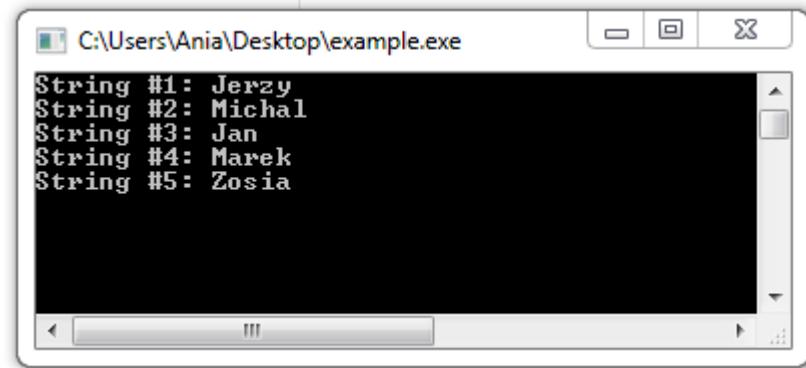
Przykład 53

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

```
int main()
{
    char *name[5]={
        "Jerzy",
        "Michał",
        "Jan",
        "Marek",
        "Zosia"};
    int ctr;

    for(ctr=0; ctr<5; ctr++)
    {
        printf("String #%d: %s\n", (ctr+1), *(name+ctr));
    }

    return 0;
}
```



Inicjalizacja zbiorcza

[0]	J	e	r	z	y	\0	
[1]	M	i	c	h	a	I	\0
[2]	J	a	n	\0			
[3]	M	a	r	e	k	\0	
[4]	Z	o	s	i	a	\0	

ctr[0][4] – OK
ctr[2][4] – nie istnieje

Przykład 54

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

char *p[][2] = {
    "Czerwone wybrane", "czervony",
    "Zlociste wybrane", "zolty",
    "Winesap", "czervony",
    "Gala", "czervonawopomaranczowy",
    "Lodi", "zielony",
    "", ""
};
```

Inicjalizacja zbiorcza +
Napisy NULL na końcu tablicy

```
int main(void) {

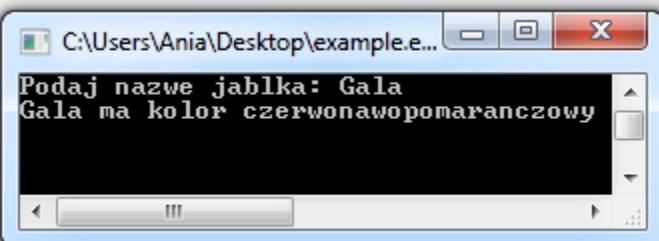
    int i, flaga = 0;
    char jablko[80];

    printf("Podaj nazwe jablka: ");
    fgets(jablko, 79, stdin);

    for(i=0; *p[i][0]; i++) {
        if(!strcmp(jablko, p[i][0])) {
            printf("%s ma kolor %s\n", jablko, p[i][1]);
            flaga = 1;
        }
    }

    if(!flaga)
        printf("Nieznane jablko\n");

    return 0;
}
```



<string.h> funkcje operujące na napisach i blokach pamięci

- ▶ **void *memchr(const void *s, int c, size_t n);**
 - ▶ skanuje pierwsze n bajtów obszaru pamięci wskazywanego przez s w poszukiwaniu znaku c ; pierwszy bajt pasujący do c przerywa szukanie;
- ▶ **void *memrchr(const void *s, int c, size_t n);**
 - ▶ skanuje ostatnie n bajtów obszaru pamięci wskazywanego przez s w poszukiwaniu znaku c ; pierwszy bajt pasujący do c przerywa szukanie;
- ▶ **int memcmp (const void *s1, const void *s2, size_t size);**
 - ▶ porównuje $size$ pierwszych bajtów z bloków pamięci $s1$ i $s2$;
- ▶ **void *memcpy (void* dest, const void* src, size_t size);**
 - ▶ kopiuje $size$ bajtów z obiektu source do obiektu dest;
- ▶ **void *memmove (void* dest, const void* src, size_t size);**
 - ▶ kopiuje $size$ bajtów z obiektu source do obiektu dest; różni się od *memcpy()* tym, że obszary obiektów source oraz dest mogą się częściowo pokrywać;
- ▶ **void * memset (void * buffer, int c, size_t num);**
 - ▶ wypełnia kolejne bajty w pamięci ustaloną wartością;

Przykład 55

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

int main(int argc, char *argv[]) {

    char nap1[80] = {"Hello "}, nap2[80] = {"World!"};

    printf(" napis1: %s\n napis2: %s\n", nap1, nap2);

    strcat(nap2, nap1);

    printf(" sklejony: %s\n", nap2);

    return 0;
}
```

```
napis1: Hello
napis2: World
sklejony: WorldHello
```

Przykład 56₍₁₎

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#define ROZMIAR 10

void pokaz_tablice(const int tab[], int n);

int main() {

    int zrodlo[ROZMIAR] = {1,2,3,4,5,6,7,8,9,10};
    int cel[ROZMIAR];
    double ciekawostka[ROZMIAR/2] = {1.0, 2.0, 3.0, 4.0, 5.0};

    puts("uzyto memcpy(): ");
    puts("zrodlo (dane oryginalne): ");
    pokaz_tablice(zrodlo, ROZMIAR);
    memcpy(cel, zrodlo, ROZMIAR*(sizeof(int)));
    puts("cel (koplowanie zrodla: ");
    pokaz_tablice(cel, ROZMIAR);

    puts("\nUzycie funkcji memmove() z nakladajacymi sie obszarami: ");
    memmove(zrodlo+2, zrodlo, 5*sizeof(int));
    puts("zrodlo -- elementy 0-5 kopowane do 2-7: ");
    pokaz_tablice(zrodlo, ROZMIAR);

    puts("\nUzycie memcpy() do skopiowania double do int: ");
    memcpy(cel, ciekawostka, (ROZMIAR/2) * sizeof(double));
    puts("cel -- 5 double 10 int: ");

    pokaz_tablice(cel, ROZMIAR);

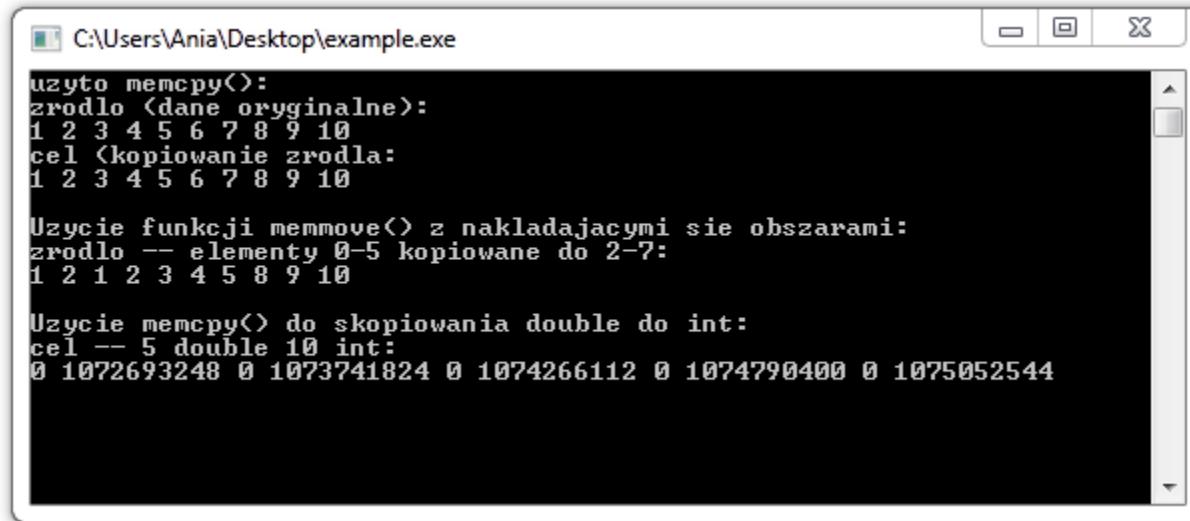
    return 0;
}
```

Przykład 56(2)

```
void pokaz_tablice(const int tab[], int n)
{
    int i;

    for(i=0; i<n; i++)
        printf("%d ", tab[i]);

    putchar('\n');
}
```



The screenshot shows a Windows command-line interface window titled "C:\Users\Ania\Desktop\example.exe". The window displays the following text:

```
uzyto memcpy():
zrodlo <dane oryginalne>:
1 2 3 4 5 6 7 8 9 10
cel <kopiowanie zrodla>:
1 2 3 4 5 6 7 8 9 10

Uzycie funkcji memmove() z nakladajacymi sie obszarami:
zrodlo -- elementy 0-5 kopiowane do 2-7:
1 2 1 2 3 4 5 8 9 10

Uzycie memcpy() do skopiowania double do int:
cel -- 5 double 10 int:
0 1072693248 0 1073741824 0 1074266112 0 1074790400 0 1075052544
```

Przykład 56

```
#include <stdio.h>
#include <string.h>
#define ROZMIAR 5

struct complex_t{
    float im, re;
};

void pokaz_tablice(const struct complex_t[], int n);

int main() {

    struct complex_t zrodlo[ROZMIAR] = {{1,1},{2,2},{3,3},{4,4},{5,5}};
    struct complex_t cel[ROZMIAR] ={{0,0}};

    puts("zrodlo (dane oryginalne): ");
    pokaz_tablice(zrodlo, ROZMIAR);
    puts("cel (dane oryginalne): ");
    pokaz_tablice(cel, ROZMIAR);
    memcpy(cel, zrodlo, ROZMIAR*(sizeof(struct complex_t)));
    puts("cel (kopiowanie zrodla: )");
    pokaz_tablice(cel, ROZMIAR);

    return 0;
}

void pokaz_tablice(const struct complex_t tab[], int n){

    for(int i=0; i<n; i++)
        printf("%.2f + j%.2f\n", (tab+i)->im, (tab+i)->re);
    printf("\n");
}
```

```
zrodlo (dane oryginalne):
1.00 + j1.00
2.00 + j2.00
3.00 + j3.00
4.00 + j4.00
5.00 + j5.00

cel (dane oryginalne):
0.00 + j0.00

cel (kopiowanie zrodla: )
1.00 + j1.00
2.00 + j2.00
3.00 + j3.00
4.00 + j4.00
5.00 + j5.00
```

```

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

#define ROZMIAR 5

struct sth_t{
    char x, y;
};

void pokaz_tablice(const struct sth_t[], int n);

int main() {
    //zgodne rozmiary źródła i celu
    char zrodlo [ROZMIAR*2] = {'A','B','C','D','E','F','G','H','I','J'};
    struct sth_t cel[ROZMIAR]= {{0,0}};
    //kopiowanie różnych typów
    memcpy(cel, zrodlo, 2*ROZMIAR*(sizeof(char)));
    pokaz_tablice(cel, ROZMIAR);

    return 0;
}

void pokaz_tablice(const struct sth_t tab[], int n){

    for(int i=0; i<n; i++)
        printf("%c / %c\n", (tab+i)->x, (tab+i)->y);
    printf("\n");
}

```

Przykład 57

```

A / B
C / D
E / F
G / H
I / J

```

Process finished with exit code 0

```

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

#define ROZMIAR 5

struct sth_t{
    char x, y;
};

void pokaz_tablice(const struct sth_t[], int n);

int main() {
    struct sth_t cel[ROZMIAR]=
{{'A','B'},{'C','D'},{'E','F'},{'G','H'},{'I','J'}};
    pokaz_tablice(cel, ROZMIAR);
    memmove(cel+2, cel, ROZMIAR*(sizeof(struct sth_t)));
    pokaz_tablice(cel, ROZMIAR);
    return 0;
}

void pokaz_tablice(const struct sth_t tab[], int n){

    for(int i=0; i<n; i++)
        printf("%c / %c\n", (tab+i)->x, (tab+i)->y);
    printf("\n");
}

```

Przykład 58

A	/	B
C	/	D
E	/	F
G	/	H
I	/	J
A	/	B
C	/	D
A	/	B
C	/	D
E	/	F

Przykład 59a

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

#define BUF_SIZE 20

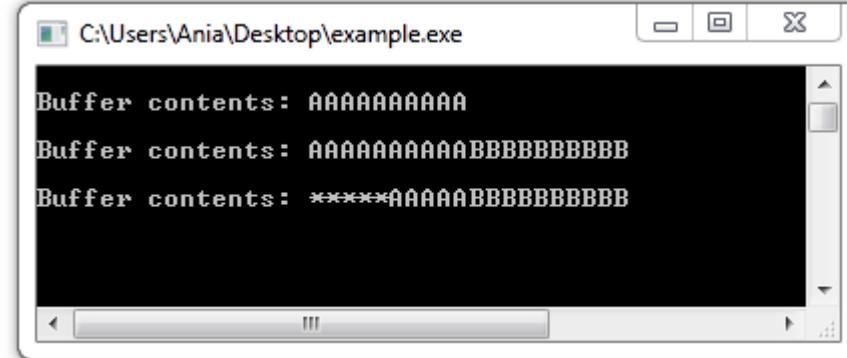
int main(void)
{
    char buffer[BUF_SIZE + 1];
    char *string;

    memset(buffer, 0, sizeof(buffer));

    string = (char *) memset(buffer, 'A', 10);

    printf("\nBuffer contents: %s\n", string);
    memset(buffer+10, 'B', 10);
    printf("\nBuffer contents: %s\n", buffer);
    memset(buffer, '*', 5);
    printf("\nBuffer contents: %s\n", buffer);

    return 0;
}
```



Przykład 59b

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

#define ROZMIAR 5

struct sth_t{
    char x, y;
};

void pokaz_tablice(const struct sth_t[], int n);

int main()
{
    struct sth_t cel[ROZMIAR] = {{'A', 'B'}, {'C', 'D'}, {'E', 'F'}, {'G', 'H'}, {'I', 'J'}};
    pokaz_tablice(cel, ROZMIAR);
    memset(cel, 'X', ROZMIAR*(sizeof(struct sth_t)));
    pokaz_tablice(cel, ROZMIAR);
    return 0;
}

void pokaz_tablice(const struct sth_t tab[], int n)
{
    for(int i=0; i<n; i++)
        printf("%c / %c\n", (tab+i)->x, (tab+i)->y);
    printf("\n");
}
```

```
A / B
C / D
E / F
G / H
I / J

X / X
X / X
X / X
X / X
X / X
```

Przykład 60

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

int main () {
    const char str[] = "Podstawy Programowania 2";
    char ch;
    char *ret;

    printf("Podaj znak: ");
    ch = getchar();

    ret = memchr(str, ch, strlen(str));

    if (ret)
        printf("Napis po znaku |%c| to - |%s|\n", ch, ret);
    else
        printf("Nie znaleziono znaku");

    return(0);
}
```

```
Podaj znak: y
Napis po znaku |y| to - |y Programowania 2|
```

```
Podaj znak: x
Nie znaleziono znaku
```

<string.h> funkcje operujące na napisach i blokach pamięci

size_t strlen(const char* s)

- ▶ zwraca ilość znaków wchodzących w skład tekstu s, znak terminujący '\0' nie jest wliczany,

char* strchr(const char* s, char c)

- ▶ zwraca wskaźnik do **pierwszego** wystąpienia litery c w tekście s, jeśli litera nie występuje zwraca NULL,

char* strrchr(const char* s, char c)

- ▶ zwraca wskaźnik do **ostatniego** wystąpienia litery c w tekście s, jeśli litera nie występuje zwraca NULL,

char* strstr(const char* s1, const char* s2)

- ▶ zwraca wskaźnik do pierwszego wystąpienia ciągu s2 w tekście s1, jeśli ciąg nie występuje zwraca NULL, jeśli s2 jest ciągiem pustym zwracany jest wskaźnik s1,

char* strdup(char* s)

- ▶ zwraca wskaźnik do nowego ciągu znaków będącego kopią tekstu s, alokując pamięć na nowy ciąg znaków, jeśli nie można zaalokować pamięci zwraca NULL.

char* strtok(char *s1, const char* s2)

- ▶ zwraca wskaźnik do następnego leksemu w napisie s1, znaki tworzące s2 są znakami rozdzielającymi leksemy, jeżeli nie ma leksemu zwracana jest wartość **NULL**

Przykład 61

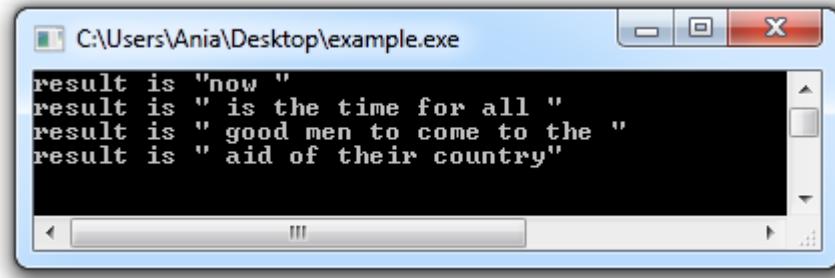
```
#include <stdio.h>
#include <string.h>

int main()
{
    char str[] = "now # is the time for all @ good men to come to the #
                  aid of their country";
    char delims[] = "#@";
    char *result = NULL;

    result = strtok(str, delims);

    while( result != NULL ) {
        printf( "result is \"%s\"\n", result );
        result = strtok(NULL, delims);
    }

    getchar();
    return 0;
}
```



Wskaźniki do funkcji

- ▶ Wskaźnik do funkcji jest zmienną zawierającą adres punktu wejścia do funkcji
- ▶ Jest to miejsce, do którego przekazane jest sterowanie w momencie wywołania funkcji (adres pod którym zaczyna się kod maszynowy funkcji)
- ▶ Mając wskaźnik do funkcji można następnie wywołać funkcję za pomocą tego wskaźnika.
- ▶ Aby utworzyć wskaźnik do funkcji, należy zadeklarować wskaźnik o typie takim samym, jak typ wartości zwracanej przez funkcję i podać typy parametrów funkcji



Wskaźniki do funkcji

- ▶ Wskaźnik do funkcji zwracającej wartość całkowitą (*int*) i przyjmującej dwa parametry typu całkowitego (*int*):

```
int (*p) (int, int);
```

- ▶ Wskaźnik do funkcji zwracającej wartość zmienoprzecinkową (*double*) oraz przyjmującej jeden parametr typu znakowego (*char*):

```
double (*wskFun) (char);
```

- ▶ Wskaźnik do funkcji zwracającej typ pusty (*void*) oraz przyjmującej wskaźnik do łańcucha znakowego (*char**) i parametr typu całkowitego (*int*)

```
void (*wsk) (char*, int);
```

Wskaźniki do funkcji

▶ Ustawienie wskaźnika do funkcji:

wskaźnik = nazwa-funkcji;

```
int suma(int, int);  
int(*p) (int, int);  
p = suma; // ustawienie wskaźnika
```

▶ Wywołanie funkcji poprzez wskaźnik:

(wskaźnik*)(lista-argumentów);

```
(*p)(10,11); //wywołanie funkcji suma
```

Wskaźniki do funkcji

- ▶ Po utworzeniu wskaźnika do funkcji, może on otrzymać adresy funkcji odpowiedniego typu

```
void(*wf)(char*);  
void Duzelit(char*);  
void MaleLit(char*);  
int zaokr(double);  
wf = Duzelit;      //OK  
wf = MaleLit;     //OK  
wf = zaokr;        //BŁĄD - nie ten typ funkcji  
wf = MaleLit();    //BŁĄD - wywołanie a nie adres
```

Wskaźniki do funkcji

Przy deklaracjach wskaźników do funkcji należy pamiętać o priorytetach operatorów

- ▶ **char* flip()**

- ▶ funkcja zwracająca wskaźnik do char

- ▶ **char (*flap) ()**

- ▶ wskaźnik do funkcji, która zwraca wartość typu char

- ▶ **char (*flop[3]) ()**

- ▶ tablica trzech wskaźników do funkcji zwracających wartość char

Przykład 62

```
#include <stdio.h>

int suma (int a, int b);

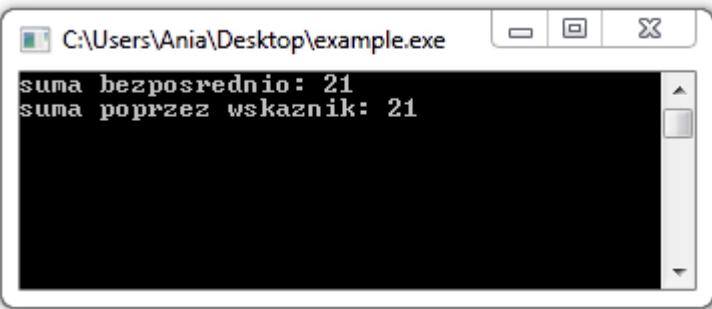
int main(void) {

    int wynik;
    int (*p) (int, int);
    wynik = suma(10,11);
    printf("suma bezpośrednio: %d\n", wynik);

    p = suma;
    wynik = (*p)(10,11);
    printf("suma poprzez wskaznik: %d\n", wynik);

    return 0;
}

int suma (int a, int b) {
    return a+b;
}
```



Deklaracja wskaźnika do funkcji

Ustawienie wskaźnika

Wywołanie funkcji poprzez wskaźnik

Przykład 63(1)

```
#include <stdio.h>

int dodawanie(int, int);
int odejmowanie(int, int);
int mnozenie(int, int);
int dzielenie(int, int);

int main(void){
    int wynik;
    int i, j, op;

    int(*p[4])(int, int);
    p[0] = dodawanie;
    p[1] = odejmowanie;
    p[2] = mnozenie;
    p[3] = dzielenie;

    printf("Wprowadz dwie liczby: ");
    scanf("%d %d", &i, &j);
    while(getchar() != '\n');

    printf(" 0: Dodawanie\n 1: Odejmowanie\n 2: Mnozenie\n 3: Dzielenie\n");

    do{
        printf("Wprowadz numer dzialania: ");
        scanf("%d", &op);
        while(getchar() != '\n');
    }while(op<0 || op>3);

    wynik = (*p[op])(i, j);

    printf("%d", wynik);

    return 0;
}
```

Alternatywnie:

```
int(*p[4])(int, int)={  
    dodawanie,  
    odejmowanie,  
    mnozenie,  
    dzielenie};
```

Deklaracja tablicy wskaźników do funkcji
oraz jej wypełnienie adresami funkcji do
wywołania

Wywołanie odpowiedniej funkcji poprzez
wskaźnik w tablicy

Przykład 63₍₂₎

```
int dodawanie (int a, int b) {
    return a+b;
}
/*********************************/
int odejmowanie (int a, int b) {
    return a-b;
}
/*********************************/
int mnozenie (int a, int b) {
    return a*b;
}
/*********************************/
int dzielenie (int a, int b) {
    if(b) return a/b;
    else return 0;
}
```

Wskaźniki do funkcji

- ▶ Podobnie jak wskaźniki do danych, wskaźniki do funkcji mogą być wykorzystywane w roli argumentów

```
void pokaz(void(*fw)(char*), char *lan);
```

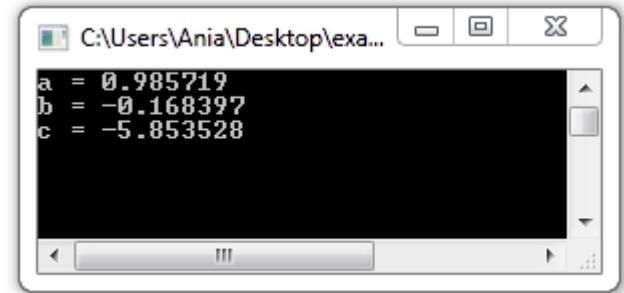
Parametr 1:
wskaźnik do funkcji

Parametr 2:
wskaźnik do char

```
void pokaz(void(*fw)(char*), char *lan){  
    (*fw)(lan); /*stosuje (*fw) do lan*/  
    puts(lan); /*wyswietla wynik*/  
}
```

Przykład 64

```
#include <stdio.h>
#include <math.h>
//****************************************************************
double funkcja(double (*wskFun) (double), double x);
//****************************************************************
int main() {
    double a, b, c, d;
    a = funkcja(sin, 1.74);
    b = funkcja(cos, 1.74);
    c = funkcja(tan, 1.74);
    printf("a = %lf\n", a);
    printf("b = %lf\n", b);
    printf("c = %lf\n", c);
    return 0;
}
//****************************************************************
double funkcja(double (*wskFun) (double), double x) {
    double wynik;
    wynik = (*wskFun)(x);
    return (wynik);
}
```



```
a = 0.985719
b = -0.168397
c = -5.853528
```

Użycie nazwy funkcji
oznacza pobranie jej adresu

Przykład 65₍₁₎

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

char pokazmenu(void) ;
void usunwiersz(void) ; /*usuwa dane z konca wiersza*/
void pokaz(void (*fp)(char*) , char* str) ;
void DuzeLit(char* ) ; /*zmienia litery male na duze*/
void MaleLit(char* ) ; /*zmienia litery duze na male*/
void Odwroc(char* ) ; /*zmienia: male->duze, duze->male*/
void Atrapa(char* ) ; /*pozostawia lancuch bez zmian*/
```

Przykład 65(2)

```
int main(void) {  
  
    char wiersz[81];  
    char kopia[81];  
    char wybor;  
    void (*wfun)(char*) ; /*wskaznik do funkcji*/  
  
    puts("Podaj lancuch (pusty wiersz konczy program)") ;  
  
    while(gets(wiersz) != NULL && wiersz[0] != '\0') {  
        while((wybor = pokazmenu()) != 'n') {  
            switch(wybor) {  
                case 'd' : wfun = DuzeLit; break;  
                case 'm' : wfun = MaleLit; break;  
                case 'o' : wfun = Odwroc; break;  
                case 'b' : wfun = Atrapa; break;  
            }  
            strcpy(kopia, wiersz); /*tworzy kopie dla funkcji pokaz*/  
            pokaz(wfun, kopia); /*korzysta z wybranej funkcji */  
        }  
        puts("Podaj lancuch (pusty wiersz konczy program)") ;  
    }  
    return 0;  
}
```

Przykład 65(3)

```
char pokazmenu(void) {
    char odp;
    puts("Wybierz jedna opcję: ");
    puts("d) duże litery\t\t m) małe litery");
    puts("o) odwracanie liter\t b) bez zmian");
    puts("n) następny lancuch");

    odp = getchar();
    odp = tolower(odp);
    usunwiersz();
    while(strchr("dmobn", odp)==NULL) {
        puts("Wpisz d, m, o, b lub n: ");
        odp = tolower(getchar());
        usunwiersz();
    }

    return odp;
}
```

char *strchr(const char *s, int c);
zwraca wskaźnik do pierwszego
wystąpienia znaku c w łańcuchu
znaków s, lub NULL kiedy znaku
nie znaleziono

Przykład 65(4)

```
void DuzeLit(char* lan){  
    while(*lan != '\0') {  
        *lan = toupper(*lan);  
        lan++;  
    }  
}  
/********************************************/  
void MaleLit(char* lan){  
    while(*lan != '\0') {  
        *lan = tolower(*lan);  
        lan++;  
    }  
}  
/********************************************/  
void Odwroc(char* lan){  
    while(*lan != '\0') {  
        if(islower(*lan))  
            *lan = toupper(*lan);  
        else if(isupper(*lan))  
            *lan = tolower(*lan);  
        lan++;  
    }  
}  
/********************************************/  
void Atrapa(char* lan){  
    /*funkcja nic nie robi*/  
}
```

Przykład 65(5)

```
void usunwiersz(void) {
    while(getchar() != '\n')
        continue;
}
```

The screenshot shows a window titled 'C:\Users\Ania\Desktop\example.exe'. Inside, a menu loop is displayed:

- Podaj lancuch <pusty wiersz konczy program>
- Napis Testowy
- Wybierz jedna opcje:
 - d> duze litery m> male litery
 - o> odwracanie liter b> bez zmian
 - n> nastepny lancuch
 - o
- nAPIS tESTOWY
- Wybierz jedna opcje:
 - d> duze litery m> male litery
 - o> odwracanie liter b> bez zmian
 - n> nastepny lancuch
 - d
- NAPIS TESTOWY
- Wybierz jedna opcje:
 - d> duze litery m> male litery
 - o> odwracanie liter b> bez zmian
 - n> nastepny lancuch
 - m
- napis testowy
- Wybierz jedna opcje:
 - d> duze litery m> male litery
 - o> odwracanie liter b> bez zmian
 - n> nastepny lancuch
 - b
- Napis Testowy
- Wybierz jedna opcje:
 - d> duze litery m> male litery
 - o> odwracanie liter b> bez zmian
 - n> nastepny lancuch

```
void pokaz(void (*fw)(char*) , char* lan) {
    (*fw)(lan);
    puts(lan);
}
```

Przykład 66

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <time.h>

struct complex_t{
    float im, re;
    void (*print)(struct complex_t*);
};

void p1 (struct complex_t* );
void p2 (struct complex_t* );
void p3 (struct complex_t* );

int main(){
    srand (time(NULL));
    void (*p[3]) (struct complex_t*) = {p1, p2, p3};

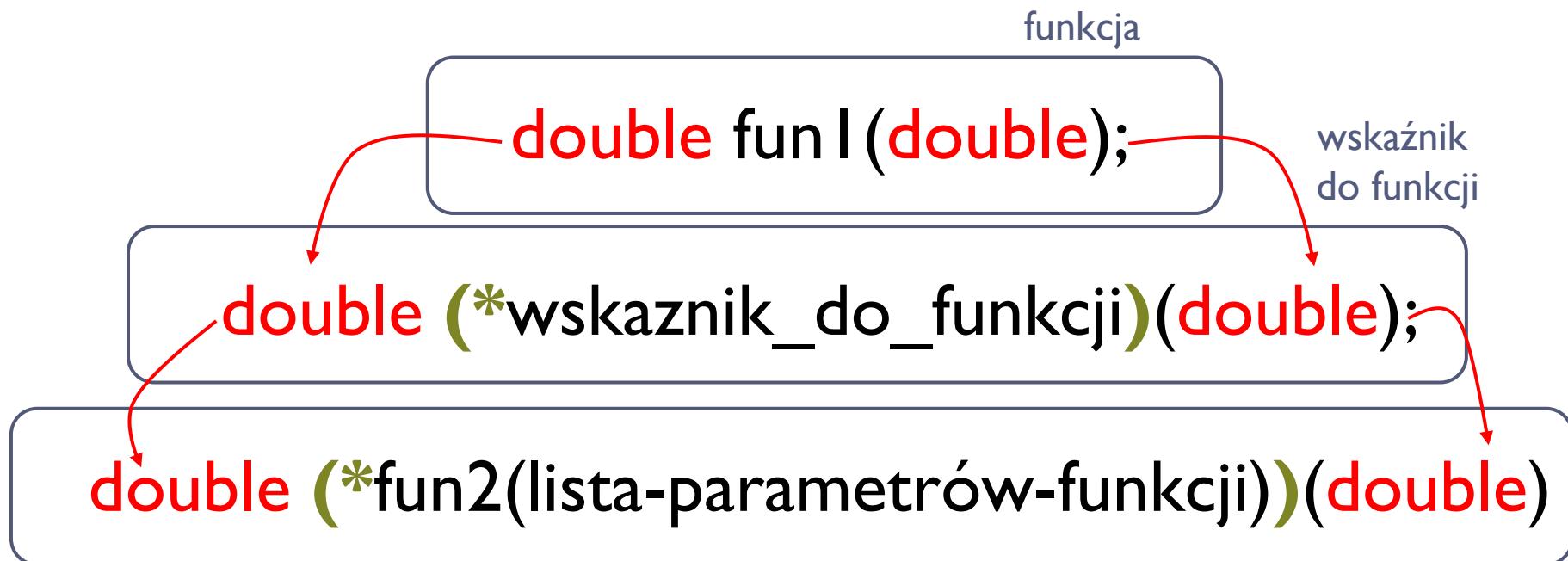
    for(int i = 0; i<5; i++){
        struct complex_t c = {rand()%9, rand()%9, p[rand()%3]};
        c.print(&c); // lub (*c.print)(&c)
    }
    return 0;
}

void p1 (struct complex_t* s){
    printf("printing - fun1: %.2f + j%.2f\n", s->im, s->re);
}
void p2 (struct complex_t* s){
    printf("printing - fun2: im: %.2f, re: %.2f\n", s->im, s->re);
}
void p3 (struct complex_t* s){
    printf("printing - fun3: |%f|\n", sqrt(pow(s->im, 2) + pow(s->re, 2)));
}
```

```
printing - fun1: 7.00 + j8.00
printing - fun1: 7.00 + j0.00
printing - fun1: 2.00 + j8.00
printing - fun3: |8.062258|
printing - fun1: 0.00 + j4.00
printing - fun2: im: 6.00, re: 2.00
printing - fun1: 5.00 + j8.00
printing - fun3: |6.403124|
printing - fun3: |4.242641|
printing - fun2: im: 2.00, re: 1.00
```

Wskaźniki do funkcji

- ▶ Wskaźników do funkcji **nie dotyczą** zasady arytmetyki wskaźników
- ▶ Wskaźniki do funkcji mogą być zwracane przez funkcje



Przykład 67a

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

double (* funmax(double (*[5])(double), double))(double);
double fun0(double x) {return log(x);}
double fun1(double x) {return x*x;}
double fun2(double x) {return exp(x);}
double fun3(double x) {return sin(x);}
double fun4(double x) {return cos(x);}

int main(void) {
    int i;
    double (*tabfun[5])(double) = {fun0, fun1, fun2, fun3, fun4};
    double (*fun)(double) = funmax(tabfun,1);

    for (i = 0; i < 5; ++i)
        if (fun == tabfun[i]) break;

    printf("Najwieksza wartosc dla x=1 ma funkcja nr %d\n", i);
    printf("Wartosc ta wynosi: %lf\n", fun(1));

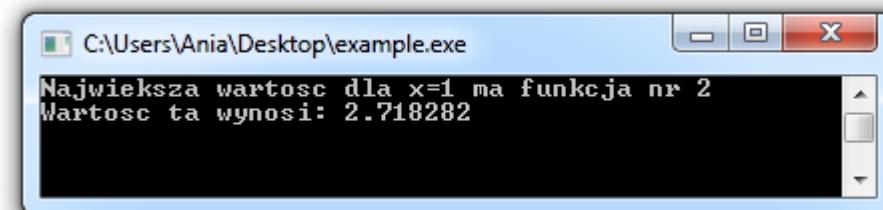
    return 0;
}

double (* funmax(double (*f[5])(double), double x))(double) {
    double z, m = f[0](x);
    int k = 0, i;

    for (i = 1; i < 5; i++) {
        if (z = f[i](x) > m) {
            m = z;
            k = i;
        }
    }
    return f[k];
}
```

nazwa funkcji

argumenty funkcji



```

#include <stdio.h>
#include <math.h>

typedef double (*FUNDtoD)(double);
typedef FUNDtoD TABFUN[];

FUNDtoD funmax(TABFUN, double);

double fun0(double x) {return log(x);}
double fun1(double x) {return x*x; }
double fun2(double x) {return exp(x);}
double fun3(double x) {return sin(x);}
double fun4(double x) {return cos(x);}

int main(void){

    int i;
    TABFUN tabfun = {fun0, fun1, fun2, fun3, fun4};
    FUNDtoD fun = funmax(tabfun,1);

    for (i = 0; i < 5; ++i)
        if (fun == tabfun[i]) break;

    printf("Najwieksza wartosc dla x=1 ma funkcja nr %d\n", i);
    printf("Wartosc ta wynosi: %lf\n", fun(1));

    return 0;
}

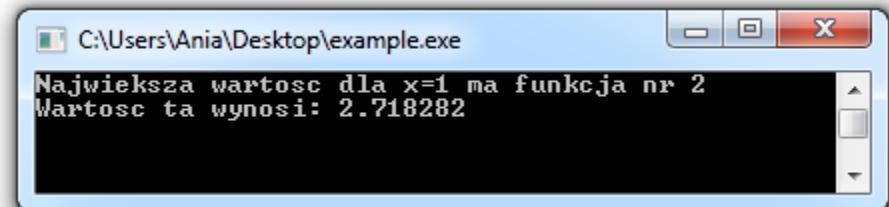
FUNDtoD funmax(TABFUN f, double x){

    double z, m = f[0](x);
    int k = 0, i;

    for (i = 1; i < 5; i++) {
        if (z = f[i](x) > m) {
            m = z;
            k = i;
        }
    }
    return f[k];
}

```

Przykład 67b



<stdlib.h>

Sortowanie szybkie qsort()

```
void * qsort(void *tab, size_t liczbaElementow, size_t roz,  
            int( * porownaj)( const void *, const void * ));
```

- ▶ funkcja *qsort* sortuje tablicę wskazywaną przez *tab* korzystając z algorytmu sortowania **quicksort**, liczba elementów tablicy jest określona przez *liczbaElementow*, a wielkość każdego elementu (w bajtach) przez *roz*
- ▶ funkcja wskazywana przez *porownaj* służy do porównywania elementów tablicy, jej postać jest następująca:

```
int nazwa-funkcji (const void *arg1, const void *arg2);
```

- ▶ funkcja musi zwracać wartość mniejszą niż 0, gdy *arg1* jest większe od *arg2*, 0 gdy *arg1* oraz *arg2* są równe oraz wartość większą od 0 jeżeli *arg1* jest większe od *arg2*

Przykład 68

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

int por(const void *, const void *);

int main(void){

    int i;
    int num[10] = {1,3,6,5,8,7,9,6,2,0};

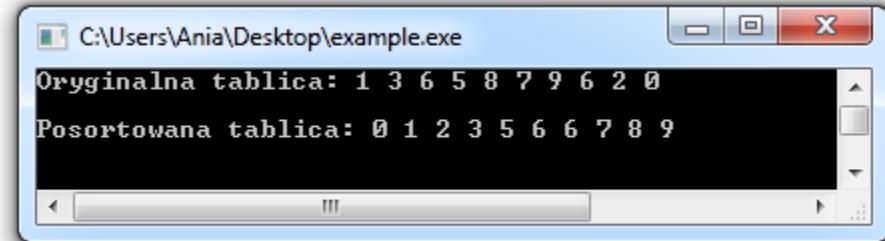
    printf("Oryginalna tablica: ");
    for(i=0; i<10; i++)
        printf("%d ", num[i]);

    qsort(num, 10, sizeof(int), por);

    printf("\n\nPosortowana tablica: ");
    for(i=0; i<10; i++)
        printf("%d ", num[i]);

    return 0;
}

int por(const void *ch, const void *s){
    return *(int *)ch - *(int *)s;
}
```



Sortowanie tablicy liczb

Funkcja porównująca elementy

Przykład 69

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

int compare (const void * a, const void * b ) {
    return strcmp(* (char **)a, *(char **)b);
}

int main(){

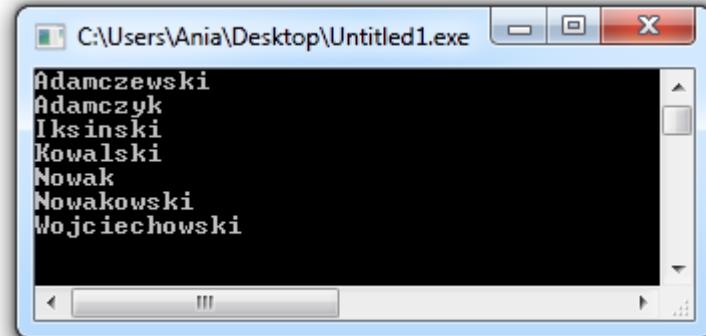
    int i, n;

    char* imiona[] = {
        "Kowalski",
        "Iksinski",
        "Adamczyk",
        "Adamczewski",
        "Nowak",
        "Nowakowski",
        "Wojciechowski"};
    n = sizeof(imiona) / sizeof(char *);

    qsort (imiona, n, sizeof (char*), compare);

    for(i=0; i<n; i++)
        puts(*(imiona+i));

    return 0;
}
```



Funkcja porównująca elementy

Sortowanie tablicy napisów

```

#include <stdio.h>
#include <math.h>
#include <stdlib.h>

struct complex_t{
    float im, re;
};

float module (struct complex_t *c){
    return sqrt(pow(c->im,2) + pow(c->re,2));
}

int compare(const void * a, const void * b ) {

    float mod1 = module((struct complex_t*)a);
    float mod2 = module((struct complex_t*)b);
    float diff = mod1 - mod2;

    if (diff < 0) return -1;
    else if (diff > 0) return 1;
    else return 0;
}

void print (struct complex_t* s, int size){
    for (int i=0; i<size; i++)
        printf("(%.2f + j%.2f) ", (s+i)->im, (s+i)->re);
    printf("\n****\n");
}

int main(){
    struct complex_t data[5] = {{2,2},{5,5},{1,1},{3,3},{4,4}};
    print(data,5);
    qsort(data,5,sizeof(struct complex_t),compare);
    print(data,5);
    return 0;
}

```

```

(2.00 + j2.00) (5.00 + j5.00) (1.00 + j1.00) (3.00 + j3.00) (4.00 + j4.00)
****
(1.00 + j1.00) (2.00 + j2.00) (3.00 + j3.00) (4.00 + j4.00) (5.00 + j5.00)
****

```

Przykład 70

Funkcja porównująca struktury względem modułu

Sortowanie tablicy struktur

Funkcje ze zmienią liczbą argumentów

- ▶ Niektóre funkcje, np. **printf()**, pozwalają na przekazanie im zmiennej liczby argumentów
- ▶ Plik nagłówkowy **stdarg.h** udostępnia narzędzia umożliwiające definiowanie przez użytkownika funkcji o zmiennej liczbie argumentów
- ▶ Deklaracja (prototyp) tego typu funkcji powinna posiadać listę parametrów o przynajmniej jednym parametrze poprzedzającym wielokropek

Parametry ustalone

```
void f1(int n, ...); //OK
int f2(const char* s, int k, ...); //OK
char f3(char c1, ..., char c2); //BŁĄD ... nie jest ostatni
double f4(...); //BŁĄD - brak parametrów
```

Funkcje ze zmienią liczbą argumentów

Plik nagłówkowy **stdarg.h** deklaruje typ **va_list** i definiuje trzy makra, iterujące poprzez listę argumentów, których liczba i typy nie są znane wywołanej funkcji.

```
void va_start(va_list ap, last);
```

▶ Makro **va_start**:

- ▶ inicjuje **ap** do dalszego użytku przez **va_arg** i **va_end**,
- ▶ musi być wywołane jako pierwsze.
- ▶ parametr **last** jest nazwą ostatniego parametru, którego typ był funkcji znany
 - ▶ ponieważ adres tego parametru jest używany w makrzu **va_start**, nie powinien on być deklarowany jako zmienna rejestrowa, funkcja czy typ tablicowy.

Funkcje ze zmienią liczbą argumentów

```
type va_arg(va_list ap, type);
```

▶ Makro **va_arg**:

- ▶ rozwija się do wyrażenia, które ma typ i wartość następnego argumentu w wywołaniu
- ▶ parametr **va_list ap** - zainicjowany przez **va_start**
- ▶ każde wywołanie **va_arg** zwraca kolejny argument typu **type** z listy argumentów nieokreślonych

```
void va_end(va_list ap);
```

▶ Makro **va_end**:

- ▶ każdemu wywołaniu **va_start** musi odpowiadać wywołanie **va_end** w obrębie tej samej funkcji
- ▶ po wywołaniu **va_end(ap)** wartość **ap** będzie nieokreślona.

```
void va_copy(va_list dest, va_list src);
```

▶ Makro **va_copy**

- ▶ tworzy kopię listy argumentów (C99)

Funkcje ze zmienią liczbą argumentów

Etapy tworzenia funkcji ze zmienią liczbą argumentów:

- ▶ Deklaracja funkcji ze znakiem wielokropka
- ▶ Utworzenie w definicji zmiennej typu **va_list** oraz jej inicjalizacja za pomocą makra **va_start()**
- ▶ Uzyskanie dostępu do listy argumentów za pomocą makra **va_arg()**
- ▶ Zwolnienie zasobów makrem **va_end()**

```
double suma(int lim, ...)  
{  
    va_list ap;      //ap - obiekt przechowujący argumenty  
    va_start(ap, lim); //zainicjalizowanie ap listą argumentów  
    double tik;  
    int tak;  
    ...  
    tik = va_arg(ap, double); // pobranie pierwszego argumentu  
    tak = va_arg(ap, int);   // pobranie drugiego argumentu  
    ...  
    va_end(ap);            // czyszczenie  
}
```

Przykład 71

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

int mnoz (int, ...);

int main(){
    int wynik;

    wynik = mnoz(5, 1, 2, 3, 4, 0);

    printf("%d\n", wynik);

    getchar();

    return 0;
}

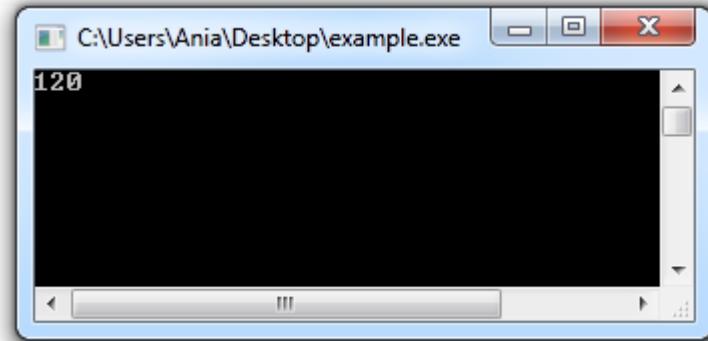
int mnoz (int pierwszy, ...){

    va_list arg;
    int iloczyn = 1, t;
    va_start (arg, pierwszy);

    for (t = pierwszy; t; t = va_arg(arg, int)) {
        iloczyn *= t;
    }

    va_end (arg);

    return iloczyn;
}
```



Przykład 72

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

double sumuj(int, ...);

int main(void) {
    double s, t;

    s = sumuj(3, 1.1, 2.5, 13.3);
    t = sumuj(6, 1.1, 2.5, 13.1, 4.1, 5.1, 6.1);

    printf("Zwroc wartosc:\n suma(1.1, 2.5, 13.3): %g\n\n", s);
    printf("Zwroc wartosc:\n suma(1.1, 2.5, 13.1, 4.1, 5.1, 6.1): %g\n", t);

    return 0;
}

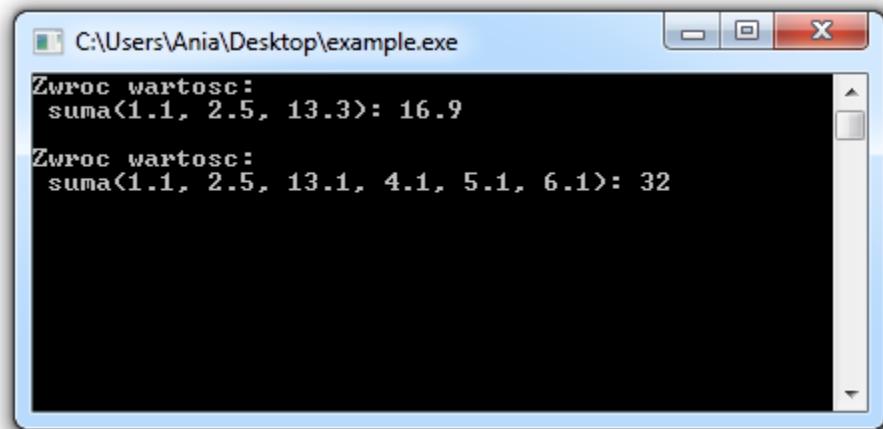
double sumuj(int lim, ...) {
    int i;
    double suma = 0;

    va_list ap;
    va_start(ap, lim);

    for(i=0; i<lim; i++)
        suma += va_arg(ap, double);

    va_end(ap);

    return suma;
}
```



```
Zwroc wartosc:
suma(1.1, 2.5, 13.3): 16.9

Zwroc wartosc:
suma(1.1, 2.5, 13.1, 4.1, 5.1, 6.1): 32
```

Przykład 73

```
#include <stdarg.h>
#include <math.h>

double wielomian(double x, int st, ...){

    double wartoscWiel = 0;
    va_list ap;
    va_start (ap,st);

    for ( ; st; --st )
        wartoscWiel += va_arg(ap, double) * pow(x,st);

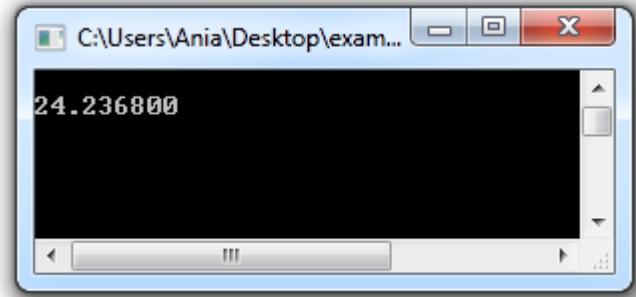
    wartoscWiel += va_arg (ap, double);
    va_end (ap);

    return wartoscWiel;
}

int main(void) {

// wartość wielomianu f(x) stopnia 3 w punkcie 2.2
// f(x) = 1.6*x^3 + 2.0*x^2 - 3.4*x + 5.0

    printf("\n%lf", wielomian(2.2, 3, 1.6, 2.0, -3.4, 5.0) );
    return 0;
}
```



```

#include <stdio.h>
#include <stdarg.h>

void minprintf(char* fmt, ...) {
    va_list ap;
    char*p, *sval;
    int ival;
    double dval;

    va_start(ap, fmt);
    for(p = fmt; *p; p++){
        if(*p != '%'){
            putchar(*p);
            continue;
        }
        switch(*++p){
            case 'd':
                ival = va_arg(ap, int);
                printf("%d", ival);
                break;
            case 'f':
                dval = va_arg(ap, double);
                printf("%f", dval);
                break;
            case 's':
                for(sval = va_arg(ap, char*); *sval; sval++)
                    putchar(*sval);
                break;
            default:
                putchar(*p);
                break;
        }
    }
    va_end(ap);
}

```

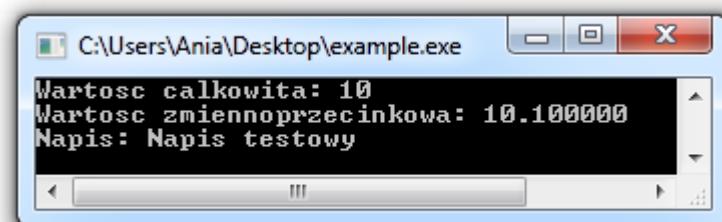
```

int main(void){

    minprintf("Wartosc calkowita: %d\n", 10);
    minprintf("Wartosc zmiennoprzecinkowa: %f\n", 10.10);
    minprintf("Napis: %s\n", "Napis testowy");

    getchar();
    return 0;
}

```



Przykład 74

```

#include <stdio.h>
#include <stdarg.h>

void typy(const char typ[], ...) {
    int i = 0, integ;
    char c, *strin;
    double doubl;
    va_list ap;
    va_start(ap,typ);
    while ( (c = typ[i++]) != '\0') {
        switch (c) {
            case 'i': case 'I':
                integ = va_arg(ap,int);
                printf("Liczba int : %d\n", integ);
                break;
            case 'd': case 'D':
                doubl = va_arg(ap,double);
                printf("Liczba double: %lf\n", doubl);
                break;
            case 'n': case 'N':
                strin = va_arg(ap,char*);
                printf("Napis : %s\n", strin);
                break;
            default:
                printf("Nielegalny kod typu!!!!\n");
        }
    }
    va_end(ap);
}

```

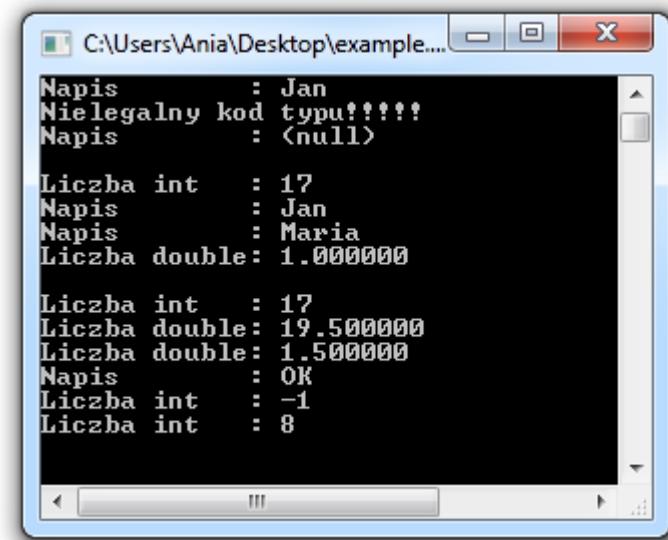
Przykład 75

```

void typy(const char typ[], ...);

int main() {
    typy("NxN","Jan",0,"Maria");
    typy("innD",17,"Jan","Maria",1.);
    typy("iDdniI",17,19.5,1.5,"OK",-1,8);
    return 0;
}

```



Argumenty funkcji main()

- ▶ Programy pisane w języku C mogą wykorzystywać argumenty przekazywane w wierszu poleceń
- ▶ Argumenty te są przekazywane do programu przez dwa argumenty funkcji **main()**:

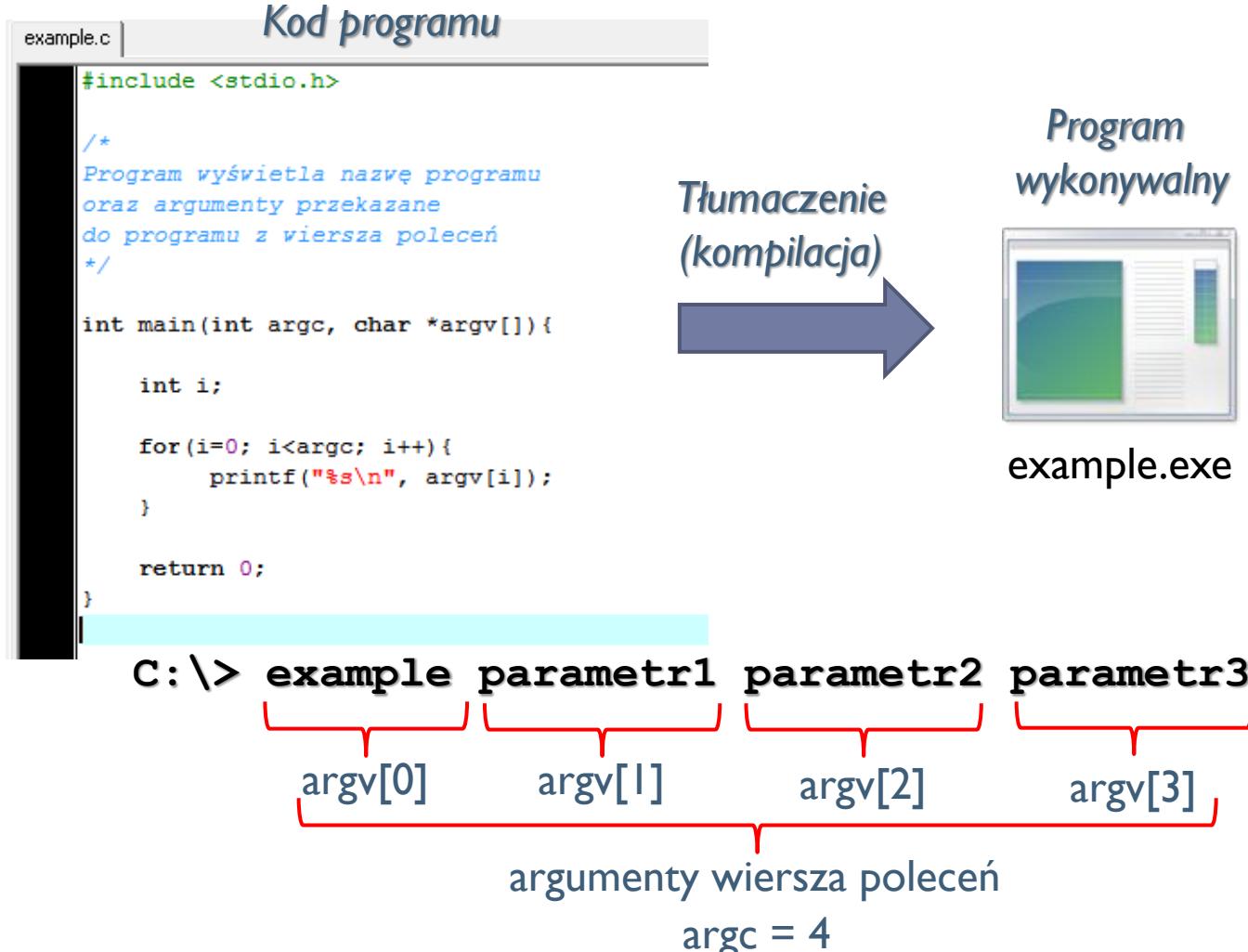
```
int main(int argc, char *argv[])
```

- ▶ **argc** (ang. *argument count*) liczba całkowita odpowiadająca liczbie argumentów występujących w wierszu poleceń, ma wartość nie mniejszą niż 1, gdyż nazwa programu kwalifikowana jest jako pierwszy argument
- ▶ **argv** (ang. *argument values*) – tablica wskaźników do napisów (wszystkie argumenty do funkcji main() przekazywane są jako napisy)

Argumenty funkcji main()

- ▶ Każdy argument wiersza poleceń musi być oddzielony znakiem spacji lub tabulacji
 - To tylko test** – interpretowane jako trzy parametry
 - To,tylko,test** – interpretowane jako jeden parametr
- ▶ Jeżeli trzeba do programu przekazać argument zawierający znaki spacji lub tabulacji należy go umieścić w cudzysłowie prostym
 - "To tylko test"** – interpretowane jako jeden parametr

Argumenty funkcji main()



Przykład 76

```
#include <stdio.h>

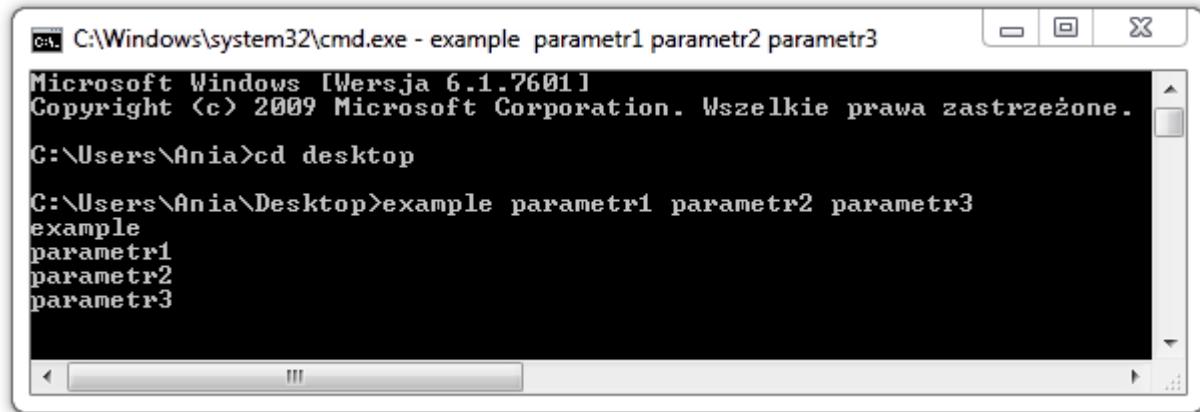
/*
Program wyświetla nazwę programu
oraz argumenty przekazane
do programu z wiersza poleceń
*/

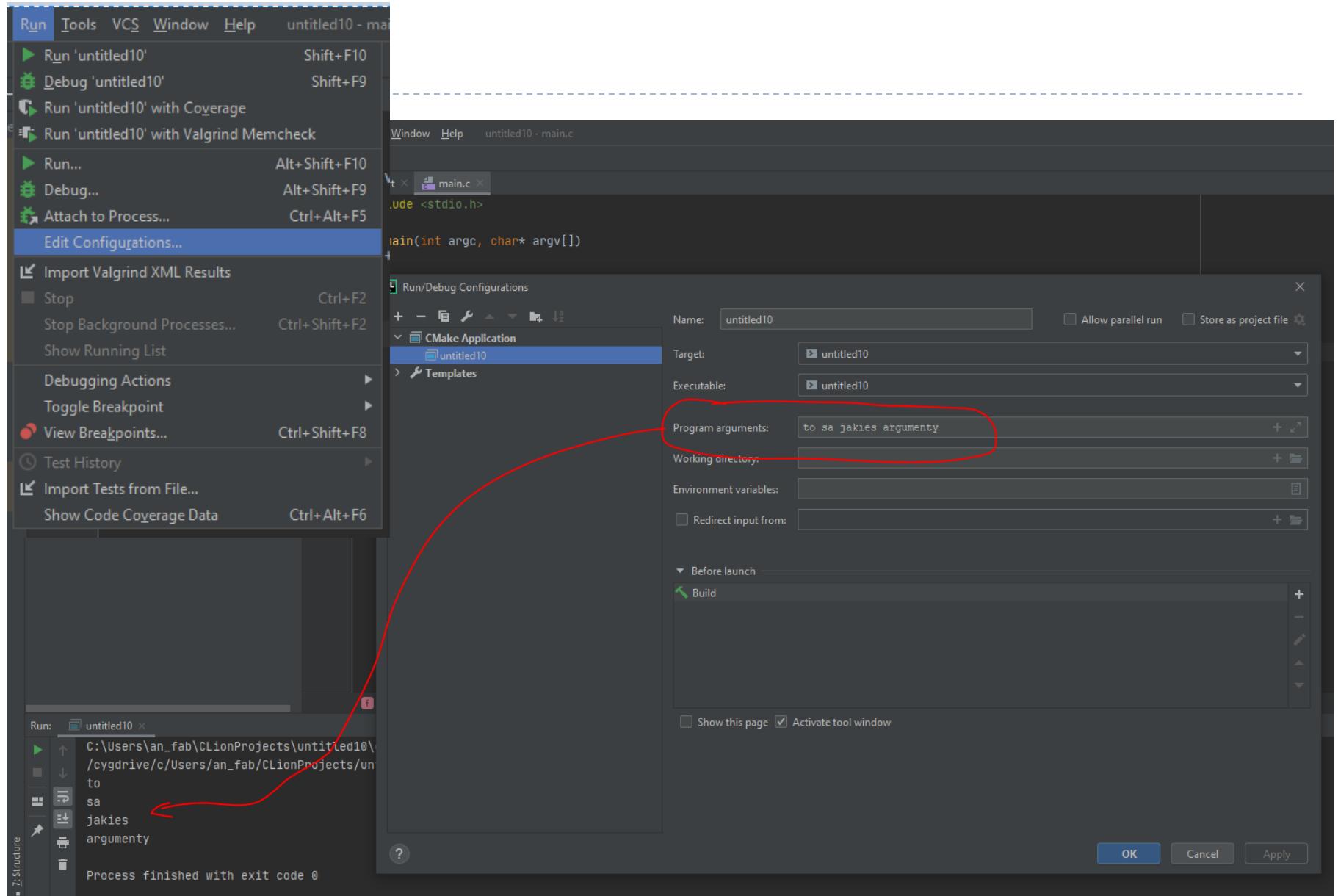
int main(int argc, char *argv[]) {

    int i;

    for(i=0; i<argc; i++) {
        printf("%s\n", argv[i]);
    }

    return 0;
}
```





Przykład 77

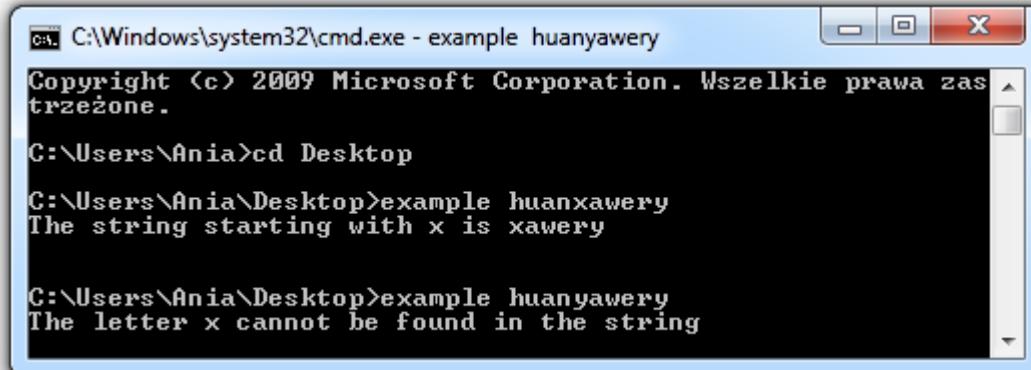
```
#include <stdio.h>
#include <string.h>

int main(int argc, char ** argv){

    char * result;

    if (argc != 2)
        printf("Usage: %s string\n", argv[0]);
    else {
        if ((result = (char *)memchr(argv[1], 'x', strlen(argv[1]))) != NULL)
            printf("The string starting with x is %s\n", result);
        else
            printf("The letter x cannot be found in the string\n");
    }

    getchar();
    return 0;
}
```



```
C:\Windows\system32\cmd.exe - example huanyawery
Copyright (c) 2009 Microsoft Corporation. Wszelkie prawa za-
trzezone.

C:\Users\Ania>cd Desktop
C:\Users\Ania\Desktop>example huanyawery
The string starting with x is xawery

C:\Users\Ania\Desktop>example huanxawery
The letter x cannot be found in the string
```

Przykład 78

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

int main(int argc, char *argv[])

{
    if(argc != 3) {
        printf("Bledne wywolanie!\nUzycie: example napis1 napis2\n\n");
        exit(1);
    }

    char nap1[80], nap2[80];

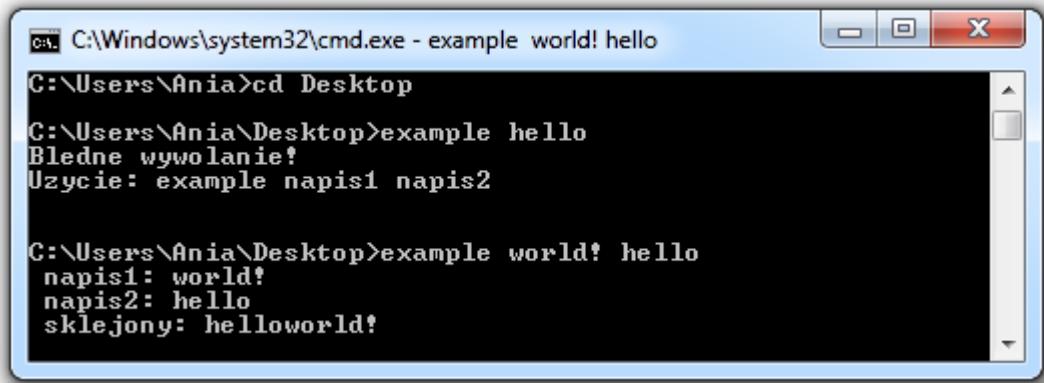
    strcpy(nap1, argv[1]);
    strcpy(nap2, argv[2]);

    printf(" napis1: %s\n napis2: %s\n", nap1, nap2);

    strcat(nap2, nap1);

    printf(" sklejony: %s\n", nap2);

    return 0;
}
```



```
C:\Windows\system32\cmd.exe - example world! hello
C:\Users\Ania>cd Desktop
C:\Users\Ania\Desktop>example hello
Bledne wywolanie!
Uzycie: example napis1 napis2

C:\Users\Ania\Desktop>example world! hello
napis1: world!
napis2: hello
sklejony: helloworld!
```

Przykład 79

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

int main(int argc, char *argv[])
{
    if(argc != 3){
        printf("Bledne wywolanie!\nUzycie: example napis1 znak\n\n");
        exit(1);
    }

    if(strlen(argv[2])!=1){
        printf("Podaj znak jako drugi argument!\n");
        printf("Uzycie: example napis1 znak\n\n");
        exit(1);
    }

    char nap1[80], zn;
    char *p;

    strcpy(nap1, argv[1]);
    zn = argv[2][0];

    p = strchr(nap1, zn);
    printf(p);

    getchar();
    return 0;
}
```

```
C:\Windows\system32\cmd.exe - example "To tylko test..." Microsoft Windows [Wersja 6.1.7601] Copyright <c> 2009 Microsoft Corporation. Wszelkie prawa zastrzezone.

C:\Users\Ania>cd Desktop
C:\Users\Ania\Desktop>example "To tylko test"
Bledne wywolanie!
Uzycie: example napis1 znak

C:\Users\Ania\Desktop>example "To tylko test" abc
Podaj znak jako drugi argument!
Uzycie: example napis1 znak

C:\Users\Ania\Desktop>example "To tylko test" y
yloko test
```

Przykład 80

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

int main(int argc, char *argv[])
{
    if(argc != 3){
        printf("Bledne wywolanie!\nUzycie: example napis1 znak\n\n");
        exit(1);
    }

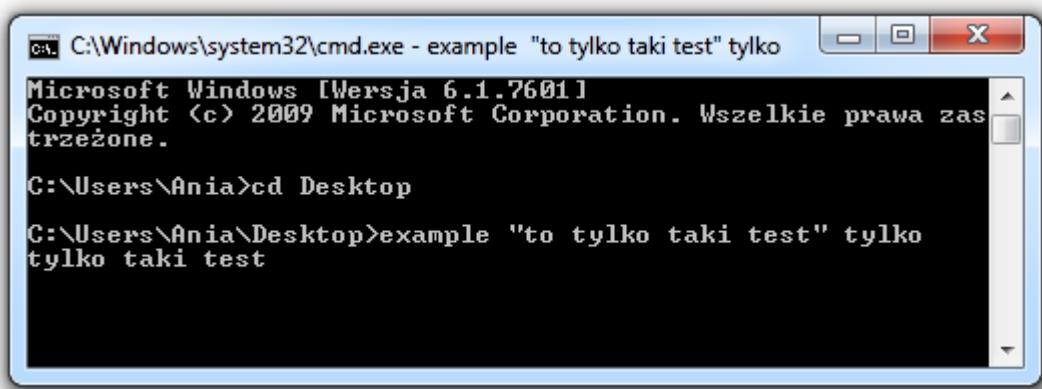
    char nap1[80], nap2[80];
    char *p=NULL;

    strcpy(nap1, argv[1]);
    strcpy(nap2, argv[2]);

    p = strstr(nap1, nap2);

    puts(p);

    getchar();
    return 0;
}
```



Konwersja łańcuchów do liczb

Funkcje z pliku nagłówkowego **<stdlib.h>**:

- ▶ **int atoi(const char* nptr)** – dokonuje konwersji łańcucha znakowego *nptr* na liczbę całkowitą *int*;
- ▶ **double atof(const char* nptr)** – dokonuje konwersji łańcucha znakowego *nptr* na liczbę zmiennoprzecinkową *double*;
- ▶ **long atol(const char* nptr)** – dokonuje konwersji łańcucha znakowego *nptr* na liczbę całkowitą *long*;
- ▶ **long strtol(const char * nptr, char ** endptr, int base)** – przetwarza łańcuch *nptr* na wartość typu *long* zapisaną w systemie o podstawie *base* oraz zgłasza adres pierwszego znaku nie należącego do liczby w *endptr*;
- ▶ **unsigned long strtoul(const char * nptr, char ** endptr, int base)** – przetwarza łańcuch *nptr* na wartość typu *unsigned long* zapisaną w systemie o podstawie *base* oraz zgłasza adres pierwszego znaku nie należącego do liczby w *endptr*;
- ▶ **double strtod(const char * nptr, char ** endptr)** – przetwarza łańcuch *nptr* na wartość typu *double* oraz zgłasza adres pierwszego znaku nie należącego do liczby w *endptr*;

Przykład 81

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

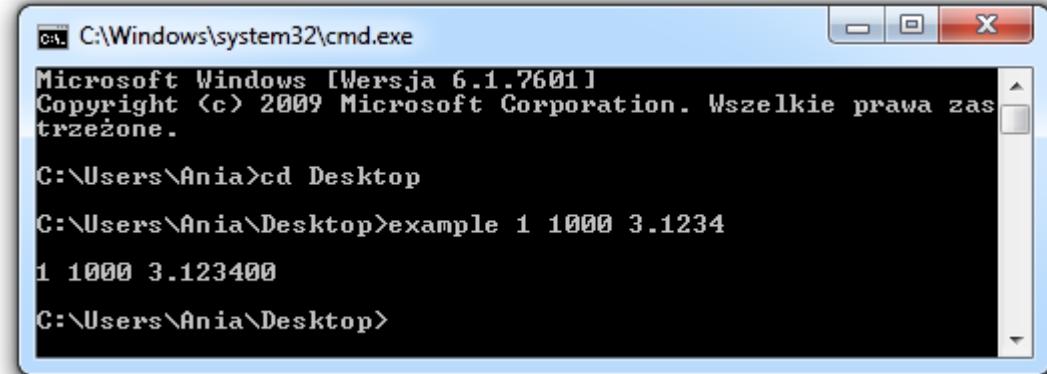
```
int main(int argc, char *argv[]) {
```

```
    int i;
    double d;
    long l;
```

```
    i = atoi(argv[1]);
    l = atol(argv[2]);
    d = atof(argv[3]);
```

```
    printf("\n%d %ld %f\n", i, l, d);
```

```
    return 0;
}
```



The screenshot shows a Windows Command Prompt window titled 'C:\Windows\system32\cmd.exe'. The window displays the following text:

```
Microsoft Windows [Wersja 6.1.7601]
Copyright © 2009 Microsoft Corporation. Wszelkie prawa zastrzeżone.

C:\Users\Ania>cd Desktop
C:\Users\Ania\Desktop>example 1 1000 3.1234
1 1000 3.123400
C:\Users\Ania\Desktop>
```

Przykład 82

```
#include <stdio.h>

int main(int argc, char *argv[]){
/*
    argv[1] - liczba konwertowana
    argv[2] - baza systemu liczbowego
*/
    char *koniec;
    long wartosc;

    wartosc = strtol(argv[1], &koniec, atoi(argv[2]));

    printf("wartosc: %ld, poprzedza %s (%c)\n",
           wartosc, koniec, *koniec);

    return 0;
}
```

```
C:\Windows\system32\cmd.exe
C:\Users\Ania>cd desktop
C:\Users\Ania\Desktop>example 10 10
wartosc: 10, poprzedza < >
C:\Users\Ania\Desktop>example 100 10
wartosc: 100, poprzedza < >
C:\Users\Ania\Desktop>example 100 16
wartosc: 256, poprzedza < >
C:\Users\Ania\Desktop>example 100abc 10
wartosc: 100, poprzedza abc <a>
C:\Users\Ania\Desktop>example 100abc 16
wartosc: 1051324, poprzedza < >
C:\Users\Ania\Desktop>example 0x100abc
wartosc: 1051324, poprzedza < >
C:\Users\Ania\Desktop>example 2345 2
wartosc: 0, poprzedza 2345 <2>
C:\Users\Ania\Desktop>example abc1000 2
wartosc: 0, poprzedza abc1000 <a>
C:\Users\Ania\Desktop>
```

Przykład 83

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

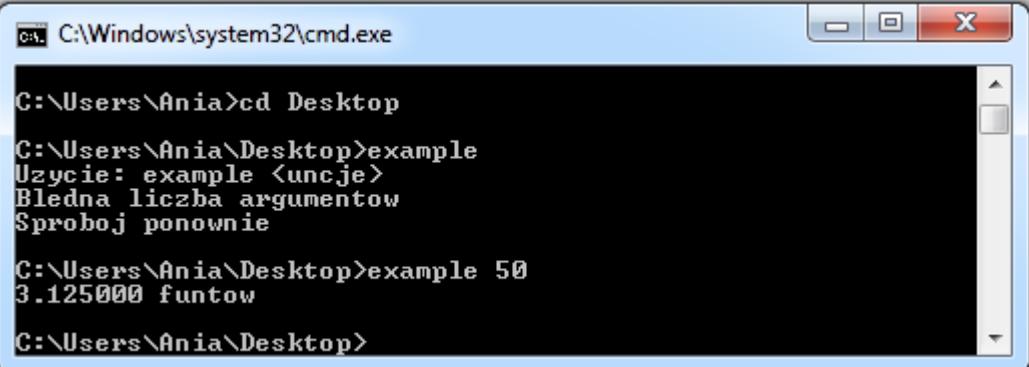
```
/*
program do zamiany uncji na funty
*/
int main(int argc, char *argv[]) {

    double funty;

    if(argc!=2) {
        printf("Uzycie: example <uncje>\n");
        printf("Bledna liczba argumentow\nSprobuj ponownie\n");
        exit(1);
    }

    funty = atof(argv[1])/16.0;
    printf("%f funtow\n", funty);

    return 0;
}
```



The screenshot shows a Windows Command Prompt window titled 'cmd C:\Windows\system32\cmd.exe'. The user has navigated to their desktop ('cd Desktop') and run the program ('example'). The program expects one argument ('example <uncje>'), but the user has provided two ('example 50'). It prints an error message: 'Bledna liczba argumentow' (Incorrect number of arguments) and 'Sprobuj ponownie' (Try again). When run again with the correct argument ('example 50'), it prints '3.125000 funtow'.

```
C:\Users\Ania>cd Desktop
C:\Users\Ania\Desktop>example
Uzycie: example <uncje>
Bledna liczba argumentow
Sprobuj ponownie

C:\Users\Ania\Desktop>example 50
3.125000 funtow

C:\Users\Ania\Desktop>
```

Sprawdzenie, czy dostarczono właściwą liczbę argumentów

Opcje w wierszu polecenia

- ▶ W języku C, oprócz parametrów w wierszu poleceń można przekazać również opcje:

```
kopiuj -s zrodlo.txt -t cel.txt -b  
kompresuj -ufz obraz.bmp  
kompresuj -u -f -z obraz.bmp
```

- ▶ W systemach zgodnych ze standardem POSIX (Mac OS X, Linux) obsługę opcji umożliwiają funkcje z pliku nagłówkowego **<unistd.h>**
 - ▶ plik ten może nie być dostępny w systemach Windows
- ▶ Na pobranie opcji pozwala funkcja **getopt()**

```
int getopt(int argc, char * const argv[], const char *optstring);
```

- ▶ Argumenty: **argc** - liczba argumentów, **argv** - tablica argumentów, **optstring** - obsługiwane opcje (jak opcja ma argumenty, należy podać dwukropki)
- ▶ Wartość zwracana: kod rozpoznanej opcji (jeden, na każde wywołanie funkcji), dla nierozpoznanej opcji: ?, po zakończeniu: -I, jeśli argument **optstring** zaczyna się od dwukropka: zwraca : jeśli po opcji nie podano argumentu

Opcje w wierszu polecenia

Zmienne związane z getopt()

- ▶ **optint** – wskazuje następny element w tablicy argumentów wiersza polecenia; wartość początkowa wynosi 1; po zakończeniu przetwarzania opcji wskazuje pierwszy argument do dalszego przetwarzania;
- ▶ **optarg** – wskazuje argument związany z opcją , o ile opcja go wymaga;
- ▶ **optopt** – w przypadku nieznanej opcji **getopt()** zwraca '?', **optopt** zawiera kod tej opcji; jeśli pominięto argument w opcji wymagającej argumentu, **getopt()** zwraca ':', **optopt** zawiera kod tej opcji (tak dzieje się wtedy, jeśli **optstring** zaczyna się od :)
- ▶ **opterr** – jeśli 1 (prawda), **getopt()** w przypadku nierozpoznanej opcji automatycznie wyświetla komunikat o błędzie na wyjściu stderr; jeśli 0 (fałsz) komunikat nie jest wyświetlany

```

#include <stdio.h>
#include <unistd.h>

int main(int argc, char *argv[]) {
    int opt;
    int kod_powrotu=0;

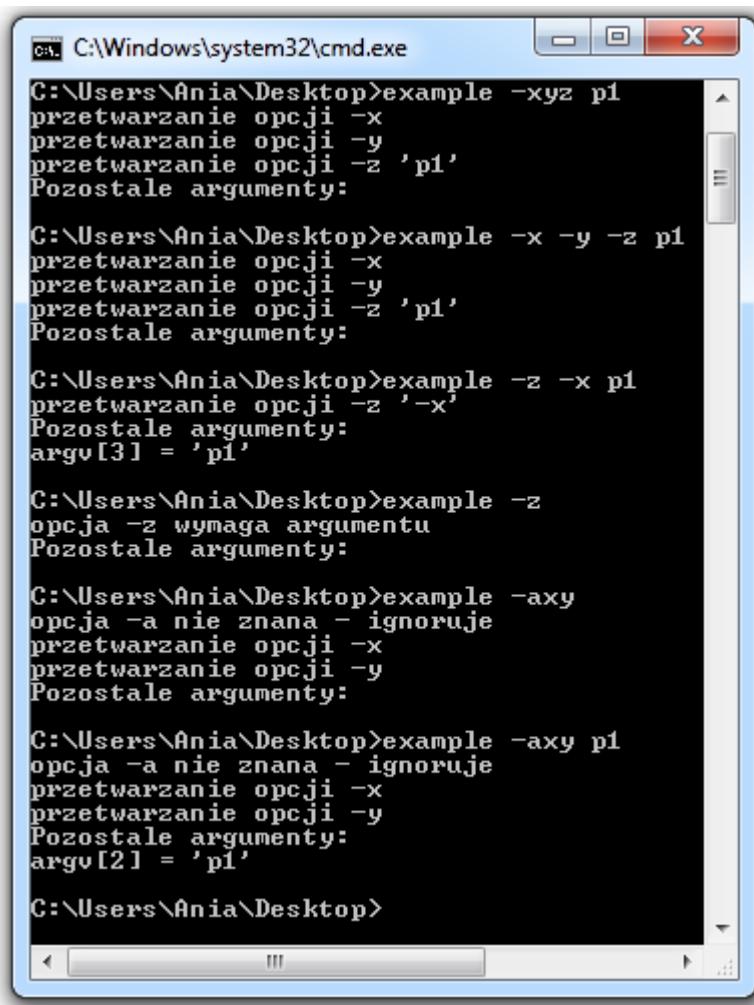
    char opcje[] = ":xyz:"; /* opcje interpretowane przez getopt */

    while ((opt = getopt(argc, argv,opcje)) != -1 )
        switch (opt) {
            case 'x' :
                printf("przetwarzanie opcji -x\n");
                break;
            case 'y' :
                printf("przetwarzanie opcji -y\n");
                break;
            case 'z' :
                printf("przetwarzanie opcji -z '%s'\n", optarg);
                break;
            case ':' : /* brak argumentu w opcji */
                printf("opcja -%c wymaga argumentu\n", optopt);
                kod_powrotu = -1; /* Blad */
                break;
            case '?' :
            default :
                printf("opcja -%c nie znana - ignoruje\n", optopt);
                kod_powrotu = -1; /* Blad */
                break;
        }
    printf("Pozostale argumenty:\n");
    for ( ; optind < argc; ++optind )
        printf("argv[%d] = '%s'\n", optind, argv[optind]);

    return kod_powrotu;
}

```

Przykład 84



The screenshot shows a Windows Command Prompt window titled 'C:\Windows\system32\cmd.exe'. It displays six separate command executions of the 'example' program:

- C:\Users\Ania\Desktop>example -xyz p1

przetwarzanie opcji -x
 przetwarzanie opcji -y
 przetwarzanie opcji -z 'p1'
 Pozostale argumenty:
- C:\Users\Ania\Desktop>example -x -y -z p1

przetwarzanie opcji -x
 przetwarzanie opcji -y
 przetwarzanie opcji -z 'p1'
 Pozostale argumenty:
- C:\Users\Ania\Desktop>example -z -x p1

przetwarzanie opcji -z '-x'
 Pozostale argumenty:
 argv[3] = 'p1'
- C:\Users\Ania\Desktop>example -z

opcja -z wymaga argumentu
 Pozostale argumenty:
- C:\Users\Ania\Desktop>example -axy

opcja -a nie znana - ignoruje
 przetwarzanie opcji -x
 przetwarzanie opcji -y
 Pozostale argumenty:
- C:\Users\Ania\Desktop>example -axy p1

opcja -a nie znana - ignoruje
 przetwarzanie opcji -x
 przetwarzanie opcji -y
 Pozostale argumenty:
 argv[2] = 'p1'

Operatory bitowe

- ▶ Język C zawiera cztery specjalne operatory działające na poziomie bitów:

- & bitowy iloczyn (AND)
- | bitowa suma (OR)
- ^ bitowa różnica symetryczna (XOR)
- ~ bitowe uzupełnienie do 1

\sim	0	1	$\&$	0	1	\mid	0	1	\wedge	0	1
-	-	-	-	-	-	-	-	-	-	-	-
1	0	0	0	1	0	1	0	1	0	1	0
1	0	1	1	1	1	1	1	0	1	0	1

- ▶ Ww. operatory **działają jedynie na liczbach całkowitych i znakach**; nie można ich używać z liczbami zmiennoprzecinkowymi

Przykład 85

```
#include <stdio.h>

int main(void) {
    int i = 100;
    printf("Wartosc poczatkowa i: %d\n", i);

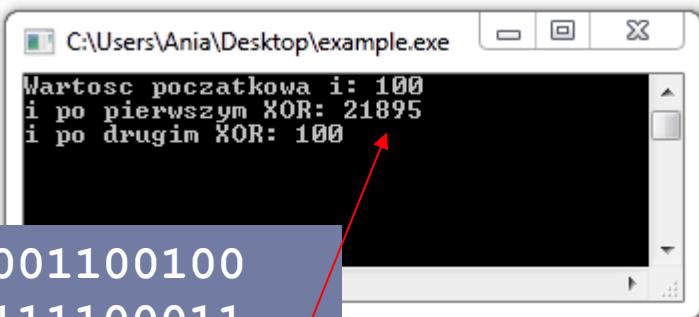
    i = i^21987;
    printf("i po pierwszym XOR: %d\n", i);

    i = i^21987;
    printf("i po drugim XOR: %d\n", i);

    return 0;
}
```

000000001100100
0101010111100011

0101010110000111



0101010110000111
0101010111100011

000000001100100

Operatory bitowe

```
unsigned short i, j, k;

i = 21;          /* wartość i:      21 (binarnie: 000000000010101) */
j = 56;          /* wartość j:      56 (binarnie: 0000000000111000) */
k = ~i;          /* wartość k: 65514 (binarnie: 1111111111101010) */
k = i & j;        /* wartość k:      16 (binarnie: 000000000010000) */
k = i ^ j;        /* wartość k:      45 (binarnie: 0000000000101101) */
k = i | j;        /* wartość k:      61 (binarnie: 0000000000111101) */
```

Priorytet (niższy niż operatory porównania):

(Największy) ~ & ^ | (Najmniejszy)

1

```
if (status & 0x4000 != 0) ...
```

Przykład 86

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

int main(void) {
    char ch;
    int i;

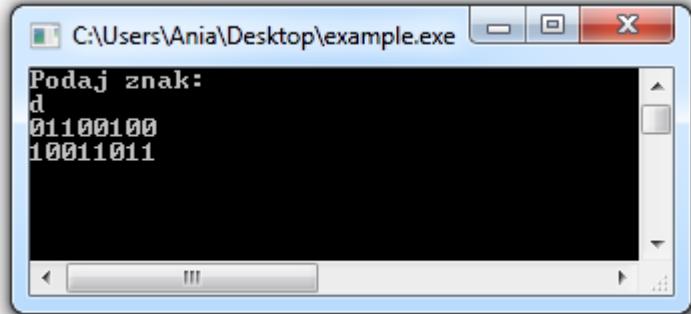
    printf("Podaj znak: \n");
    ch = getchar();

    /*wyswietlanie reprezentacji binarnej*/
    for(i=128; i>0; i=i/2)
        if(i&ch) printf("1");
        else printf("0");

    printf("\n");
    /*odwrócenie wzoru bitowego*/
    ch = ~ch;

    /*wyswietlanie reprezentacji binarnej*/
    for(i=128; i>0; i=i/2)
        if(i&ch) printf("1");
        else printf("0");

    return 0;
}
```



'd' ->(100)₁₀-> (01100100)₂

01100100 (100)
10000000 (128)

00000000 (&)

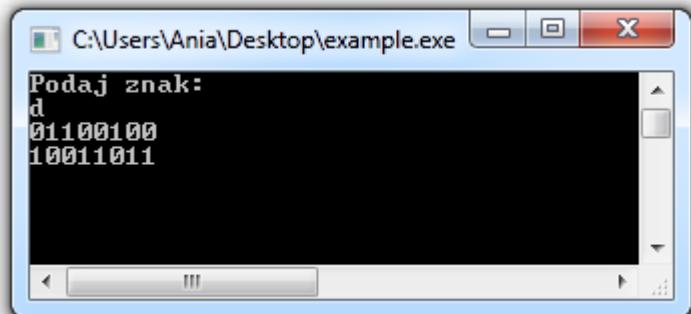
01100100 (100)
01000000 (64)

01000000 (&)

01100100 (100)
00100000 (32)

00100000 (&)

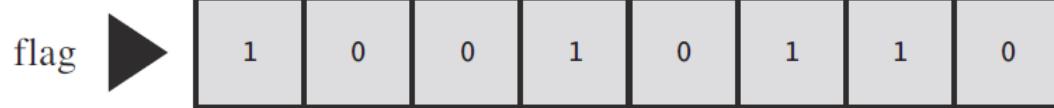
Przykład 87



'd' ->(100)₁₀-> (01100100)₂



AND



EQUALS

01100100 (100)
10000000 (128)

00000000 (&)

01100100 (100)
01000000 (64)

01000000 (&)

01100100 (100)
00100000 (32)

00100000 (&)

Przykład 88

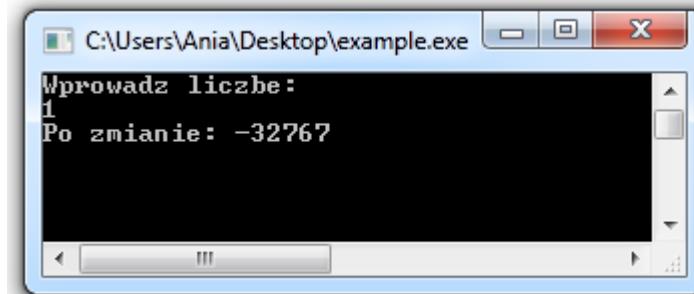
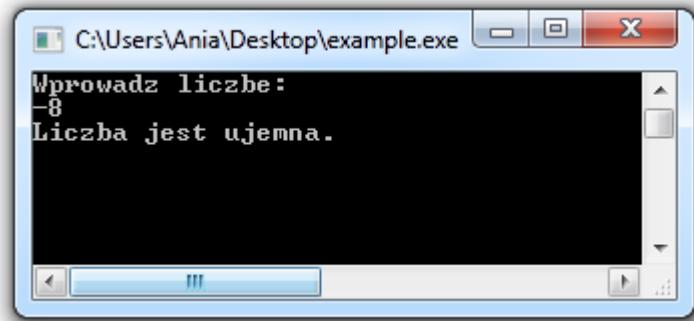
```
#include <stdio.h>

int main(void) {
    short i;

    printf("Wprowadz liczbe: \n");
    scanf("%hd", &i);

    if(i&32768)
        printf("Liczba jest ujemna.\n");
    else
    {
        i = i | 32768;
        printf("Po zmianie: %hd\n", i);
    }

    return 0;
}
```



1000000000000000
11111111111000

1000000000000000 (&)

1000000000000000
0000000000000001

1000000000000001 (|)

Operator przesunięcia

- ▶ Język C zawiera dwa operatory nie występujące powszechnie w innych językach programowania
 - ▶ operator przesunięcia lewostronnego <<
 - ▶ operator przesunięcia prawostronnego >>
- ▶ *liczba-bitów* określa, o ile miejsc w lewo bądź w prawo powinny zostać przesunięte bity *wartosci*
- ▶ Operatory te mogą być stosowane wyłącznie z argumentami znakowymi lub całkowitoliczbowymi
- ▶ Przesunięcie w lewo oznacza przesunięcie bitów o jedno miejsce w lewo i wprowadzenie z prawej strony zera (równoważne z mnożenie, przez 2)
- ▶ Przesunięcie w prawo oznacza przesunięcie bitów o jedno miejsce w prawo oraz powielenie najstarszego bitu na skrajnie lewą pozycję (równoważne z dzieleniem przez 2)
- ▶ Bity wysunięte poza granicę są tracone.

wartosc << liczba-bitów
wartosc >> liczba-bitów

Przykład 89a

```
#include <stdio.h>

void wyswietl_binarnie(unsigned u);

int main(void) {
    unsigned short u;
    u = 45678;

    wyswietl_binarnie(u); printf("\t%d\n", u);
    u = u >> 1;

    wyswietl_binarnie(u); printf("\t%d\n", u);
    u = u << 1;

    wyswietl_binarnie(u); printf("\t%d\n", u);
    return 0;
}

void wyswietl_binarnie(unsigned u) {
    unsigned n;

    for(n=32768; n>0; n=n/2)
        if (u & n) printf("1");
        else printf("0");

    printf("\n");
}
```

```
1011001001101110
45678
0101100100110111
22839
1011001001101110
45678
```

Przykład 89b

```
#include <stdio.h>

void wyswietl_binarnie(unsigned u);

int main(void) {

    unsigned short u;
    u = 45678;

    wyswietl_binarnie(u); printf("\t%d\n", u);
    u = u >> 1;

    wyswietl_binarnie(u); printf("\t%d\n", u);
    u = u << 1;

    wyswietl_binarnie(u); printf("\t%d\n", u);
    return 0;
}

void wyswietl_binarnie(unsigned u) {
    unsigned n;

    for(n=32768; n>0; n=n>>1)
        if (u & n) printf("1");
        else printf("0");

    printf("\n");
}
```

The screenshot shows a window titled "C:\Users\Ania\Desktop\example.exe". Inside the window, there is a black text area with white text. The text displays the binary representation of the decimal number 45678, followed by its decimal value, and then the process is repeated three more times. The output is:

```
1011001001101110
45678
0101100100110111
22839
1011001001101110
45678
```

Operacje na bitach

▶ Ustawianie bitu

```
/* ustawienie j-tego bitu */  
i |= 1 << j;
```

```
dla bitu nr 4:  
i = 0x0000; /* i: 0000000000000000 */  
i |= 0x0010; /* i: 0000000000010000 */
```

▶ Zerowanie bitu

```
/* zerowanie j-tego bitu */  
i &= ~(1 << j);
```

```
dla bitu nr 4  
i = 0x00ff; /* i: 000000011111111 */  
i &= ~0x0010; /* i: 000000011101111 */
```

▶ Sprawdzanie, czy bit jest ustawiony

```
dla bitu nr 4  
if ( i & 0x0010)
```

```
/* sprawdzanie j-ego bitu */  
if(i & 1 << j)
```

```

#include <stdio.h>

/*bit 0, 1 i 2 */
#define BLUE    1
#define GREEN   2
#define RED     4

void print_bits(unsigned short num);

int main(void) {
    unsigned short i = 0x000;
    print_bits(i);

    i |= BLUE; /*ustawia BLUE*/
    print_bits(i);

    i |= RED; /*ustawia RED*/
    print_bits(i);

    i &= ~BLUE; /*czyści BLUE*/
    print_bits(i);

    if (i & BLUE)
        printf("\nBit BLUE ustawiony");

    i |= BLUE|GREEN; /*ustawia BLUE i GREEN*/
    print_bits(i);

    i &= ~(BLUE | GREEN); /*czyści BLUE i GREEN*/
    print_bits(i);

    if (i & (BLUE | GREEN))
        printf("\nBit BLUE lub GREEN ustawiony");

    return 0;
}

```

Przykład 90

```

void print_bits(unsigned short num) {
    putchar('\n');
    for(int i=(sizeof(unsigned short)*8)-1;i>=0;i--) {
        char c=(num&(1<<i))?'1':'0';
        putchar(c);
    }
}

```

```

0000000000000000
0000000000000001
000000000000000101
000000000000000100
000000000000000111
000000000000000100
Process finished with exit code 0

```

Dyrektywa **#define**

- ▶ Dyrektywa **#define** nakazuje wykonanie podstawienia tekstu w całej treści programu → jedna sekwencja znaków zostaje zastąpiona inną
- ▶ proces ten jest ogólnie nazywany **makropodstawieniem**
- ▶ na dyrektywy preprocessora nie mają wypływu bloki kodu

Przykład 91

```
#include <stdio.h>

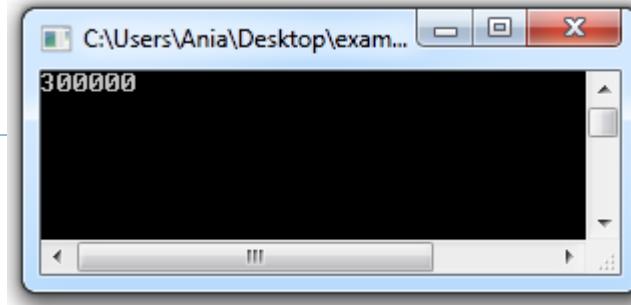
void f(void);

int main(void) {

    #define PREDKOSCSWIATLA 300000
    f();
    return 0;
}

void f(void) {

    printf("%ld", PREDKOSCSWIATLA);
}
```



Dyrektywa #define

- ▶ Oprócz definiowania za pomocą dyrektywy **#define** makr mogących zastępować dowolny ciąg znaków, można ją stosować do definiowania *makr funkcyjnych*
- ▶ W *makrach funkcyjnych* do makra rozwijanego przez preprocesor można przekazać argumenty

Przykład 92

```
#include <stdio.h>

#define TWO 2           /* you can use comments if you like */
#define OW "Consistency is the last refuge of the unimaginative. - Oscar Wilde" /* a backslash continues a definition */

#define FOUR TWO*TWO
#define PX printf("X is %d\n",
#define FMT "%d\n"

int main(void) {
    int x = TWO;
    PX
    x = FOUR);
    printf(FMT, x);
    printf("%s\n", OW);
    printf("TWO: OW\n");

    return 0;
}
```

```
X is 4
4
Consistency is the last refuge of the unimaginative. - Oscar Wilde
TWO: OW
```

Przykład 93

```
#include <stdio.h>

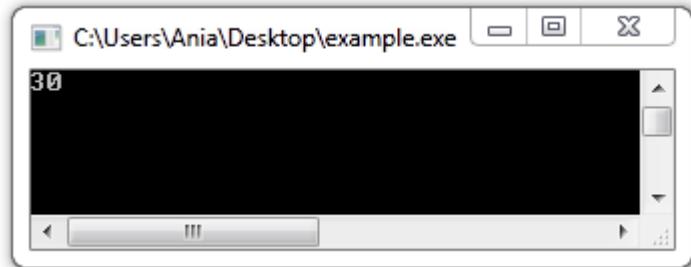
#define SUMA(i,j) i+j

int main(void)
{
    int suma;

    suma = SUMA(10,20);
    printf("%d", suma);

    return 0;
}
```

Bez spacji



Wartości 10 oraz 20 są automatycznie podstawiane w miejsce parametrów i oraz j.

Przykład 94

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

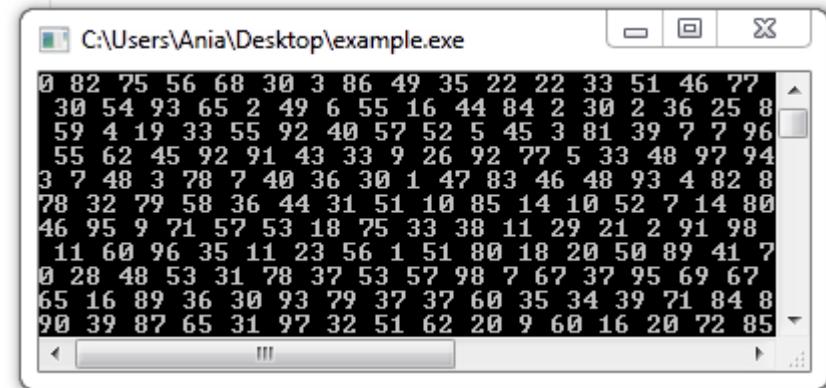
#define ZAKRES(i, min, maks) (i<min) || (i>maks) ? 1 : 0

int main(void) {

    int r, i=0;

    do{
        do{
            r = rand();
        }while(ZAKRES(r,1,100));
        printf("%d ", r);
        i++;
    }while(i<100);

    return 0;
}
```



The screenshot shows a window titled "C:\Users\Ania\Desktop\example.exe". Inside the window, there is a grid of 100 integers arranged in 10 rows and 10 columns. The numbers are generated by a random number generator and range from 0 to 100. The grid looks like this:

0	82	75	56	68	30	3	86	49	35
30	54	93	65	2	49	6	55	16	44
59	4	19	33	55	92	40	57	52	5
55	62	45	92	91	43	33	9	26	92
3	7	48	3	78	7	40	36	30	1
78	32	79	58	36	44	31	51	10	85
46	95	9	71	57	53	18	75	33	38
11	60	96	35	11	23	56	1	51	80
0	28	48	53	31	78	37	53	57	98
65	16	89	36	30	93	79	37	37	60

Przykład 95

```
#include <stdio.h>

#define MACRO(num, str) do{\
    printf("%d", num); \
    printf(" is"); \
    printf(" %s number", str); \
    printf("\n"); \
}while(0)

int main(void) {
    int num;

    printf("Enter a number: ");
    scanf("%d", &num);

    if (num & 1)
        MACRO(num, "Odd");
    else
        MACRO(num, "Even");

    return 0;
}
```

```
C:\Users\an_fab\Desktop\example.exe
Enter a number: 10
10 is Even number
-----
Process exited after 14.05 seconds with return value 0
Press any key to continue . . .
```

```
C:\Users\an_fab\Desktop\example.exe
Enter a number: 9
9 is Odd number
-----
Process exited after 1.581 seconds with return value 0
Press any key to continue . . .
```

Przykład 96

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

```
#define MAKS(i,j) i>j?i:j
```

```
int main(void) {
```

```
#define MAKS(i,j) ((i)>(j))?(i):(j)
```

```
    printf("%d\n", MAKS(1,2));
```

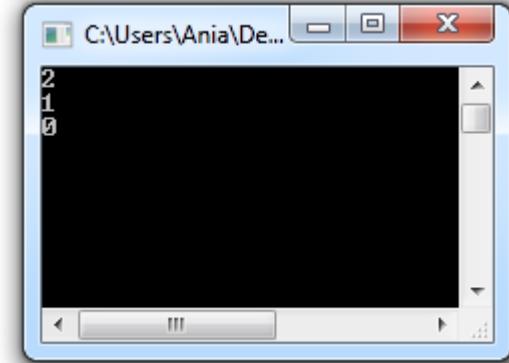
```
    printf("%d\n", MAKS(1,-1));
```

*/*ta instrukcja nie działa poprawnie*/*

```
    printf("%d\n", MAKS(1&&-1,0));
```

```
    getchar();
    return 0;
}
```

1 && - 1 > 0 ? 1 && -1 : 0
((1&&-1)>(0))?(1&&-1):(0)



Znaczenie nawiasów w makrach

- ▶ Aby zapewnić poprawność rozwinięcia makra, każdorazowo jego argument powinien być ujęty w nawiasy

```
#define TWO_PI 2*3.14159  
conversion_factor = 360/TWO_PI; //conversion_factor=360/2*3.14159;
```

vs

```
#define TWO_PI (2*3.14159)  
conversion_factor = 360/TWO_PI; //conversion_factor=360/(2*3.14159);
```

```
#define SCALE(x) (x*10)  
j = SCALE (i + 1); //j=(i+1*10)
```

vs

```
#define SCALE(x) ((x)*10)  
j = SCALE (i + 1); //j=((i+1)*10)
```

Operatory # oraz

- ▶ Preprocesor języka C zawiera dwa potencjalnie wartościowe operatory:
 - ▶ **Operator #** – zmienia argument makra funkcyjnego na napis w cudzysłowie prostym
 - ▶ **Operator ##** – scalą dwa identyfikatory

Przykład 97

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

```
#define NAPIS(str) #str
```

```
int main(void) {
```

```
    int wartosc;
```

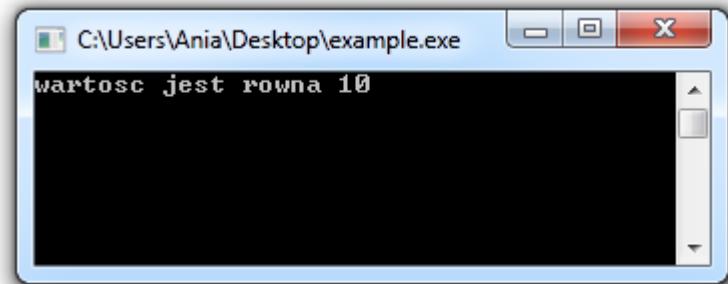
```
    wartosc = 10;
```

```
    printf("%s jest rowna %d", NAPIS(wartosc), wartosc);
```

```
    getchar();
```

```
    return 0;
```

```
}
```



Przykład 98

```
#include <stdio.h>

#define wyswietl(i) printf("%d %d\n", i##1, i##2)

int main(void) {

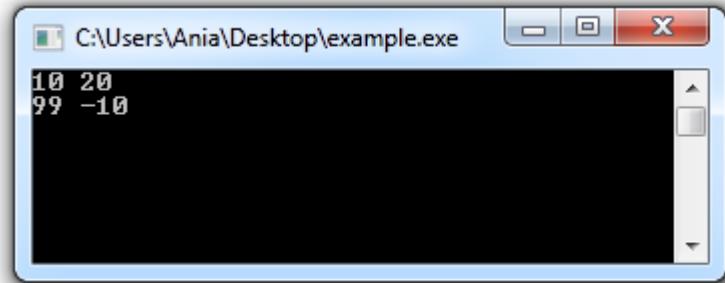
    int licz1, licz2;
    int i1, i2;

    licz1 = 10;
    licz2 = 20;
    i1 = 99;
    i2 = -10;

    wyswietl(licz);
    wyswietl(i);

    getchar();

    return 0;
}
```



Przykład 99

```
#include <stdio.h>

#define GENERIC_MAX(type) \
type type##_max(type x, type y) \
{ \
    return x > y ? x: y; \
}

GENERIC_MAX(int)
GENERIC_MAX(float)
GENERIC_MAX(double)

int main() {

    printf("max int: %d \n", int_max(19,16)); \
    printf("max float: %f \n", float_max(6,9)); \
    printf("max double: %lf \n", float_max(6.5,5.9)); \
    return 0;
}
```

```
max int: 19
max float: 9.000000
max double: 6.500000

Process finished with exit code 0
```

Kompilacja warunkowa

- ▶ Preprocesor języka C zawiera kilka dyrektyw, pozwalających na wybiórcze kompilowanie fragmentów programu.
- ▶ Jest to określane jako **kompilacja warunkowa**
- ▶ Ogólna postać dyrektywy **#if**:

```
#if wyrażenie-ustalone
    ciąg-instrukcji
#endif
```

- ▶ Jeśli wartością **wyrażenia-ustalonego** jest prawda, wówczas następuje kompilacja **ciągu-instrukcji**, w przeciwnym wypadku kompilator pomija instrukcje.

```
#if
#else
#elif
#endif
#ifndef
#endif
```

Kompilacja warunkowa

- ▶ Aby określić zdarzenie alternatywne, można wraz z dyrektywą **#if** użyć dyrektywy **#else**
 - ▶ jest ono komplilowane gdy wyrażenie ustalone jest fałszywe
- ▶ Można też tworzyć „drabinkę” if-else-if korzystając z dyrektywy **#elif**
 - ▶ w takim przypadku komplilowane są linie kodu związane z pierwszym prawdziwym wyrażeniem
 - ▶ pozostałe są ignorowane

```
#if wyrażenie-ustalone
    ciąg-instrukcji
#else
    ciąg-instrukcji
#endif
```

```
#if wyrażenie-ustalone-1
    ciąg-instrukcji
#elif wyrażenie-ustalone-2
    ciąg-instrukcji
#elif wyrażenie-ustalone-3
    ciąg-instrukcji
.
.
.
#endif
```

Kompilacja warunkowa

- ▶ Inny sposób komplikacji warunkowej prezentuje dyrektywa **#ifdef**

```
#ifdef nazwa-makra
    ciąg-instrukcji
#endif
```
- ▶ jeśli makro o nazwie *nazwa-makra* jest zdefiniowane, wówczas *ciąg-instrukcji* jest komplikowany, w przeciwnym przypadku jest pomijany
- ▶ użycie **#else** pozwala na określenie kodu alternatywnego
- ▶ Dopełnieniem **#ifdef** jest **#ifndef**
 - ▶ różnica polega na tym, że *ciąg-instrukcji* jest komplikowany, gdy nazwa makra jest niezdefiniowana
- ▶ Inny sposób sprawdzenia, czy makro jest zdefiniowane:
 - ▶ **defined nazwa-makra**

```
#ifdef WIN32 ⇔ #if defined WIN32
```

Przykład 100

```
#include <stdio.h>

#define ZESTAW_ZNAKOW 256

int main() {

    int i;

#if ZESTAW_ZNAKOW == 256
    printf("Znaki rozszerzonego kodu ASCII.\n");
#else
    printf("Znaki podstawowego kodu ASCII.\n");
#endif

    for(i=0; i<ZESTAW_ZNAKOW; i++)
        printf("%c ", i);

    return 0;
}
```



Przykład 101

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

#define DEBUG

int main(){
    FILE *skad, *dokad;
    char ch;
    char* zrodlo = "zrodlo.txt", *cel = "cel.txt";

    skad = fopen(zrodlo,"r");
    dokad = fopen(cel,"w");

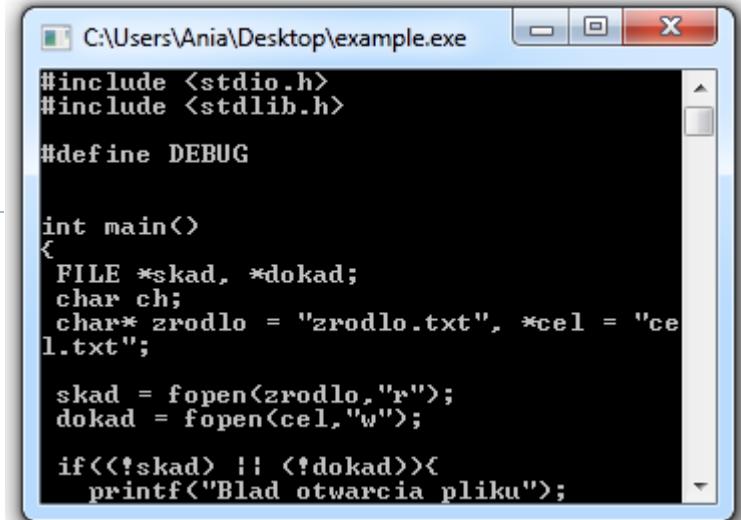
    if((!skad) || (!dokad)){
        exit(1);
    }

    while(!feof(skad)){
        ch = fgetc(skad);
        fputc(ch,dokad);

#ifndef DEBUG
        putchar(ch);
#endif
    }

    fclose(skad);
    fclose(dokad);

    return 0;
}
```



```
C:\Users\Ania\Desktop\example.exe
#include <stdio.h>
#include <stdlib.h>

#define DEBUG

int main()
{
    FILE *skad, *dokad;
    char ch;
    char* zrodlo = "zrodlo.txt", *cel = "cel.txt";

    skad = fopen(zrodlo,"r");
    dokad = fopen(cel,"w");

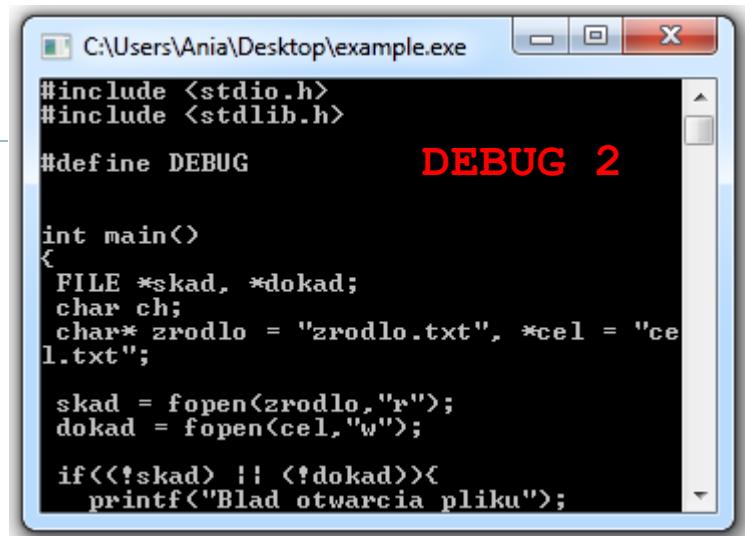
    if((!skad) || (!dokad)){
        printf("Blad otwarcia pliku");
    }
}
```

Przykład 102

```
#include <stdio.h>
#define DEBUG 2

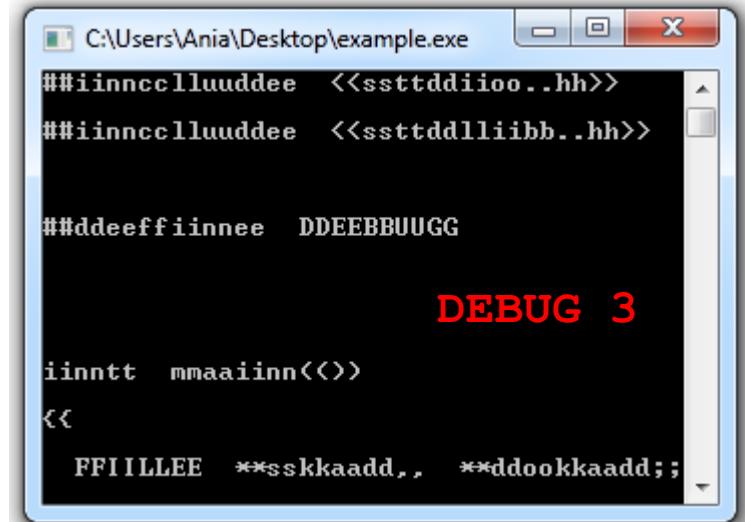
int main(){
    FILE *skad, *dokad;
    char ch;
    char* zrodlo = "zrodlo.txt", *cel = "cel.txt";
    skad = fopen(zrodlo,"r");
    dokad = fopen(cel,"w");
    if((!skad) || (!dokad)){
        printf("Blad otwarcia pliku");
        exit(1);
    }
    while(!feof(skad)){
        ch = fgetc(skad);
        #if DEBUG == 1 || DEBUG == 3
            putchar(ch);
        #endif
        fputc(ch,dokad);
        #if DEBUG >=2
            putchar(ch);
        #endif
    }
    fclose(skad);
    fclose(dokad);

    return 0;
}
```



```
C:\Users\Ania\Desktop\example.exe
#include <stdio.h>
#include <stdlib.h>
#define DEBUG 2

int main()
{
    FILE *skad, *dokad;
    char ch;
    char* zrodlo = "zrodlo.txt", *cel = "cel.txt";
    skad = fopen(zrodlo,"r");
    dokad = fopen(cel,"w");
    if((!skad) || (!dokad)){
        printf("Blad otwarcia pliku");
    }
}
```



```
C:\Users\Ania\Desktop\example.exe
##iinncclluuddee <<ssttddiioo..hh>>
##iinncclluuddee <<ssttddlliibb..hh>>

##ddeeffiinnee DDEEBBUUGG

FFIILLEE ***sskkaadd,, ***dookkaadd;;
```

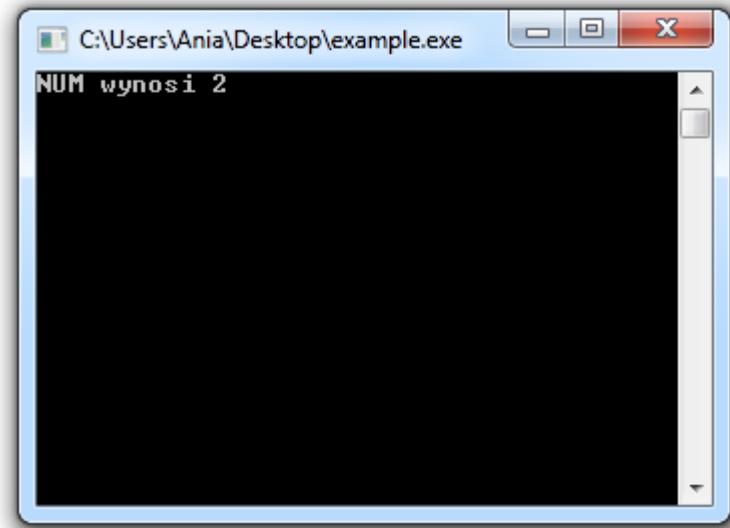
Przykład 103

```
#include <stdio.h>

#define NUM 2

int main()
{
#define NUM == 1
    printf("NUM wynosi 1\n");
#define NUM == 2
    printf("NUM wynosi 2\n");
#define NUM == 3
    printf("NUM wynosi 3\n");
    ten fragment nie jest komplgowany
#define NUM == 4
    printf("NUM wynosi 4\n");
#define NUM == 5

    return 0;
}
```



Przykład 104

```
#include <stdio.h>

#define PROJEKTTESTOWY 29

#if defined PROJEKTTESTOWY

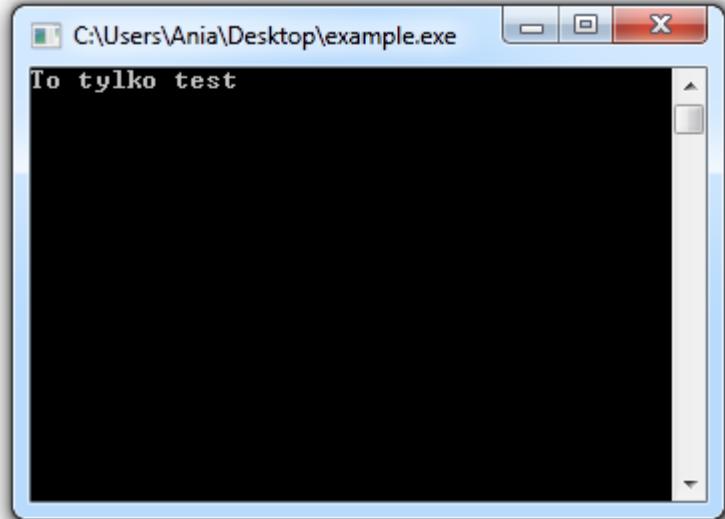
int main() {

    printf("To tylko test");

    getchar();

    return 0;
}

#endif
```



Dyrektywy preprocesora

- ▶ Preprocesor języka C dopuszcza cztery specjalne dyrektywy: **#error**, **#undef**, **#line** oraz **#pragma**

- ▶ Dyrektywa **#undef**
 - ▶ usuwa definicję makra, jeżeli makro nie jest zdefiniowane, dyrektywa nie robi nic

Przykład 105

```
#include <stdio.h>

#define DEF

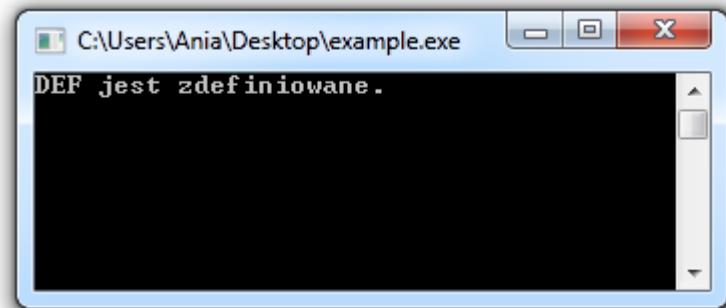
int main() {
    #ifdef DEF
        printf("DEF jest zdefiniowane.\n");
    #endif

    #undef DEF

    #ifdef DEF
        printf("Ta linia nie zostanie skompilowana.\n");
    #endif

    getchar();

    return 0;
}
```



Dyrektywy preprocesora

▶ Dyrektywa **#error**

- ▶ powoduje, że kompilator przerywa komplikację i wyświetla komunikat-o-błędzie razem z innymi informacjami zależnymi od implementacji
- ▶ jej głównym zadaniem jest debugowanie

#error komunikat-o-błędzie

```
#if INT_MAX < 100000
#error int type is too small
#endif
```

```
#if __STDC_VERSION__ != 201112L
#error Not C11
#endif
```

```
#if defined (WIN32)
...
#elif defined(MAC_OS)
...
#elif defined(LINUX)
...
#else
#error No operating system specified
#endif
```

Dyrektywy preprocesora

▶ Dyrektywa **#line**

- ▶ pozwala zmienić podczas komplikacji numer komplowanej linii oraz nazwę pliku zawierającego treść programu

```
#line numer-linii "nazwa-pliku"
```

- ▶ *numer-linii* – liczba z zakresu 1 – 32767
- ▶ *nazwa-pliku* – dowolny ciąg znaków składających się na poprawną nazwę pliku

▶ Dyrektywa **#pragma**

- ▶ pozwala implementatorom kompilatorów na definiowanie nowych dyrektyw

```
#pragma instrukcje
```

Przykład 106

```
#include <stdio.h>

#define DEF

int main(){

    int i;
    /* Ustalenie numeru linii na 1000, a nazwy pliku na mojprog.c
     */

    #line 1000 "mojprog.c"
    #error Nalezy sprawdzic numer linii i nazwe pliku

    getchar();

    return 0;
}
```

Line	File	Message
1000:3	C:\Users\Ania\Desktop\mojprog.c	#error Nalezy sprawdzic numer linii i nazwe pliku

Makra wbudowane języka C

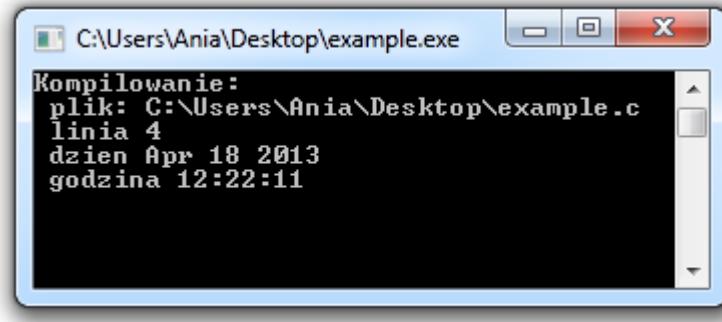
- ▶ Jeżeli kompilator jest zgodny ze standardem ANSI języka C powinien zawierać co najmniej pięć predefiniowanych makr:
 - ▶ **_LINE** określa liczbę całkowitą odpowiadającą numerowi aktualnie komplilowanej linii kodu źródłowego
 - ▶ **_FILE** określa napis będący nazwą aktualnie komplilowanego pliku
 - ▶ **_DATE** określa napis przechowujący bieżącą datę systemową postaci *miesiąc/dzien/rok*
 - ▶ **_TIME** określa napis przechowujący godzinę datę systemową postaci *godziny:minuty:sekundy*
 - ▶ **_STDC** jest określone jako 1, jeżeli kompilator dostosowuje się do standardu ANSI.

Przykład 107a

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

```
int main(void) {  
  
    printf("Kompilowanie:\n"  
           "plik: %s\n linia %d\n dzien %s\n godzina %s",  
           __FILE__, __LINE__, __DATE__, __TIME__);  
  
    getchar();  
  
    return 0;  
}
```

Wartości makr ustalane są w czasie kompilacji. Jeżeli podany program ma nazwę *example* i został skompilowany 18 kwietnia 2013 roku o godz. 12:22:11, to po jego uruchomieniu zawsze wyświetli się napis jak wyżej.



Przykład 107b

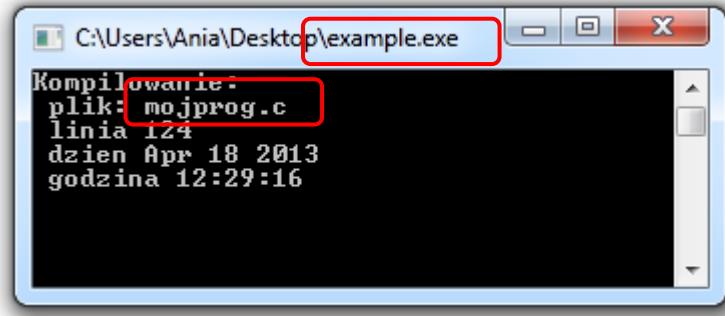
```
#include <stdio.h>
```

```
int main(void) {
```

```
#line 123 "mojprog.c"
printf("Kompilowanie:\n
plik: %s\n linia %d\n dzien %s\n godzina %s",
__FILE__, __LINE__, __DATE__, __TIME__);
```

```
getchar();
```

```
return 0;
}
```



Przykład 108

```
#include <stdio.h>

void why_me();

int main() {
    printf("The file is %s.\n", __FILE__);
    printf("The date is %s.\n", __DATE__);
    printf("The time is %s.\n", __TIME__);
    printf("The version is %ld.\n", __STDC_VERSION__);
    printf("This is line %d.\n", __LINE__);
    printf("This function is %s\n", __func__);
    why_me();
    return 0;
}

void why_me()
{
    printf("This function is %s\n", __func__);
    printf("This is line %d.\n", __LINE__);
}
```

```
The file is /cygdrive/c/Users/an_fab/CLionProjects/new_example/main.c.
The date is May 6 2021.
The time is 12:53:34.
The version is 201112.
This is line 10.
This function is main
This function is why_me
This is line 18.

Process finished with exit code 0
```

Makra o zmiennej liczbie argumentów

- ▶ W języku C, zmienną liczbę parametrów mogą mieć również makra (ang. *variadic macros*) C99
- ▶ Analogicznie jak w przypadku funkcji, ostatnim argumentem w definicji makra powinien być wówczas wielokropek
- ▶ Aby dostać się do parametrów makra, należy użyć identyfikatora **_VA_ARGS_** w jego miejsce zostaną podstawione wszystkie 'nieokreślone' parametry (z wyjątkiem pierwszego), oddzielone przecinkami

```
#define PR(...) printf(_VA_ARGS_)

PR("Cześć i czołem!");                                // jeden argument
PR("waga=%d, wysylka=$%.2f\n", wg, wy); // trzy argumenty
```

Przykład 109

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

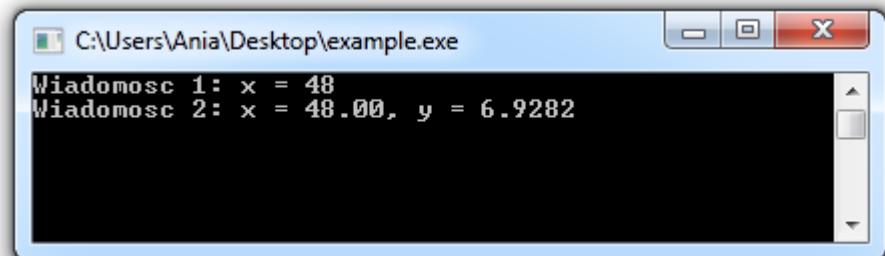
#define PR(X, ...) printf("Wiadomosc " #X ": " __VA_ARGS__)

int main(void) {

    double x = 48;
    double y;

    y = sqrt(x);
    PR(1, "x = %g\n", x);
    PR(2, "x = %.2f, y = %.4f\n", x, y);

    getchar();
    return 0;
}
```



Makro czy funkcja?

- ▶ Makra są trudniejsze w użyciu niż funkcje (mogą spowodować nieoczekiwane efekty uboczne)
- ▶ Niektóre kompilatory ograniczają definicję makr do jednego wiersza
- ▶ Wybór między makrem a funkcją jest wyborem między szybkością, a oszczędnością pamięci
 - ▶ makra tworzą kod wplatany (ang. in-line code)
 - ▶ w przypadku funkcji w programie znajduje się tylko jedna kopia instrukcji (wykorzystywane mniej miejsca), kompilator jednak musi przechodzić w miejsce w którym znajduje się funkcja a następnie powracać do funkcji wywołującej
- ▶ Makra nie analizują typów

Funkcje wplatane (ang. inline)

- „Deklaracja funkcji jako wplatanej sugeruje, że jej wywołanie powinno być tak szybkie, jak to jest tylko możliwe. Zakres w jakim sugestie zostaną zrealizowane jest określany przez konkretne implementacje” C99/C11

```
inline TYP nazwa-funkcji (lista-argumentów) ;
```

- Funkcje wplatane definiuje się przed ich pierwszym wywołaniem w danym pliku
- Kompilator może zdecydować, czy zastąpić wywołanie funkcji **inline** jej kodem
- Ponieważ funkcja wplatana nie tworzy oddzielnego bloku z kodem źródłowym, nie można pobrać do niej adresu
- Funkcje wplatane mogą też być niewidoczne w debuggerze

Funkcje wplatane (ang. inline)

```
#include <stdio.h>

inline void pobierz_linie(){
    while(getchar() != '\n')
        continue;
}

int main(){
...
    pobierz_linie();
...
}
```

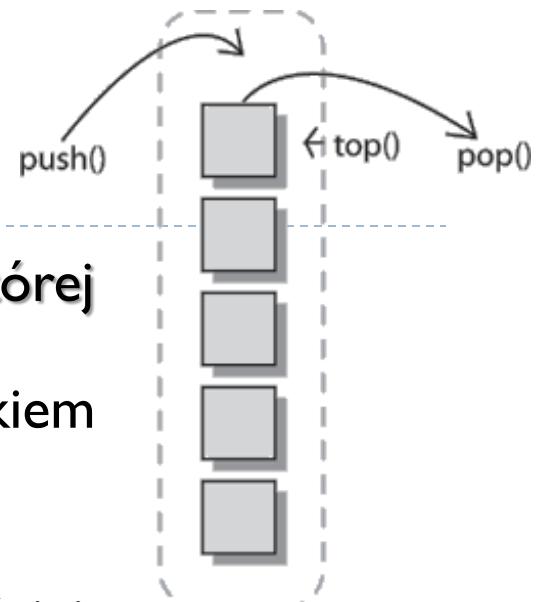
```
#include <stdio.h>

inline void pobierz_linie(){
    while(getchar() != '\n')
        continue;
}

int main(){
...
    while(getchar() != '\n')
        continue;
...
}
```

wywołanie funkcji zastąpione kodem
(funkcje inline powinny być krótkie)

Stos



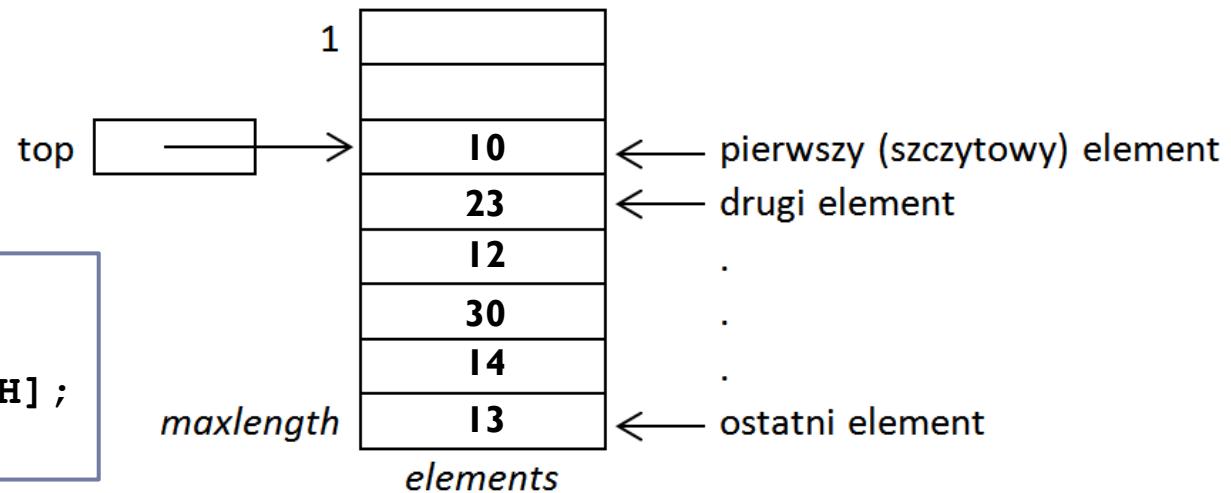
- ▶ **Stos** (ang. *stack*) jest liniową strukturą danych, w której operacje wstawiania oraz usuwania elementu mają miejsce tylko na jednym końcu, zwanym wierzchołkiem lub szczytem
 - ▶ często oznaczany skrótem LIFO (ang. *last-in-first-out*)
- ▶ Abstrakcyjne typy danych z rodziny stosów najczęściej implementują pięć następujących operacji podstawowych:
 - ▶ **MAKENULL(S)** – usunięcie ze stosu S wszystkich elementów
 - ▶ **TOP(S)** – zwrócenie szczytowego elementu stosu S
 - ▶ **POP(S)** – usunięcie szczytowego elementu stosu S i zwrócenie jego wartości
 - ▶ **PUSH(x, S)** – odłożenie na wierzchołek stosu S wartości x oraz przesunięcie istniejących elementów na dalsze pozycje
 - ▶ **EMPTY(S)** – zwraca wartość prawda, jeżeli stos S jest pusty oraz wartość fałsz w przeciwnym przypadku

Stos – implementacja tablicowa

▶ Stos można reprezentować w postaci tablicy

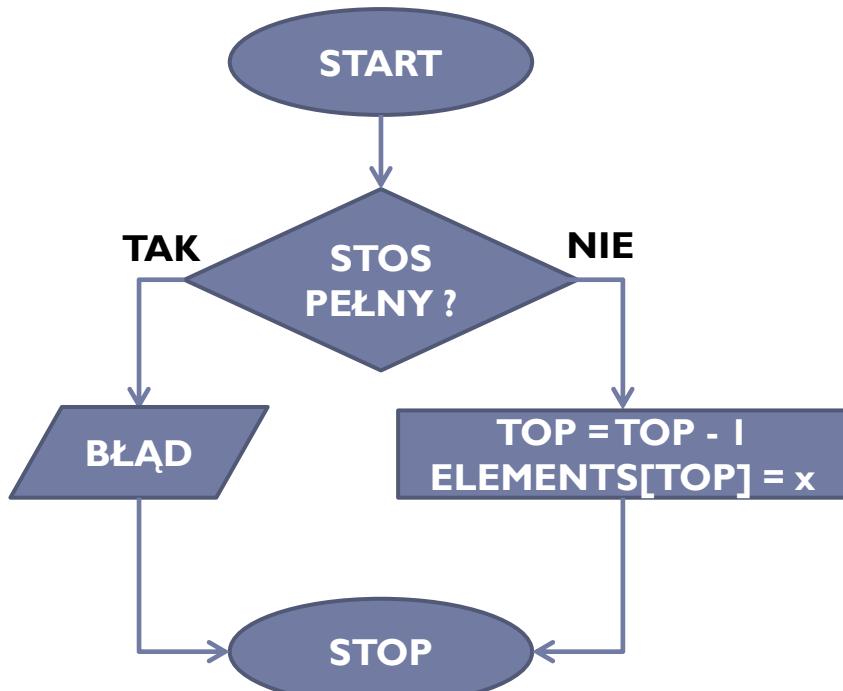
- ▶ wykorzystując fakt, że dodawanie i usuwanie elementów stosu zachodzi na jednym z jego końców, można stos zakotwiczyć w tablicy tak, aby jego najgłębiej położony element pokrywał się zawsze z ostatnim elementem tablicy
- ▶ zmienna *top* wskazywać będzie komórkę zawierającą szczytowy element stosu

```
struct stack{  
    int top;  
    TYP elements[MAXLENGTH] ;  
}
```



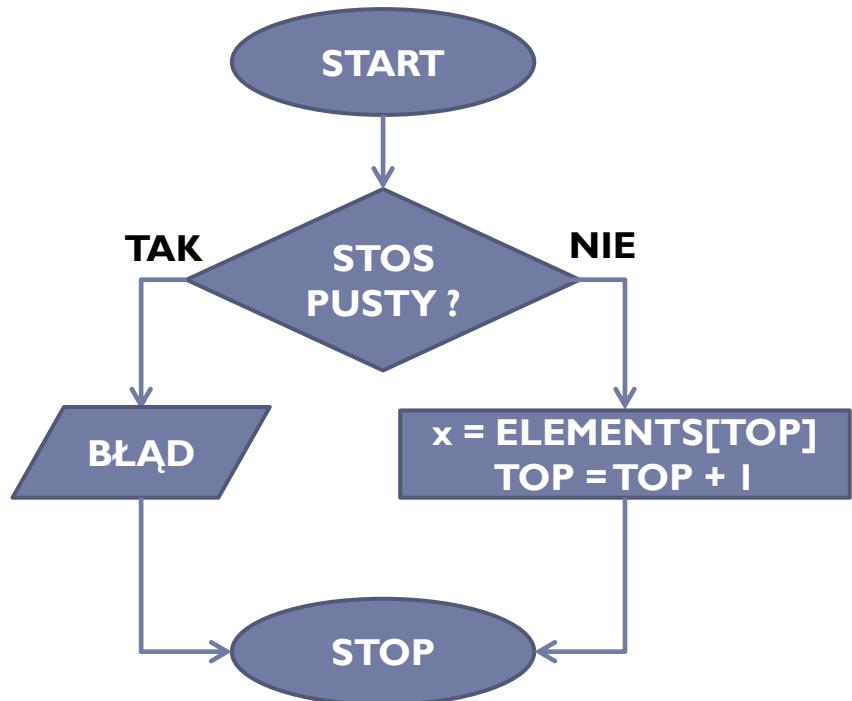
Stos – implementacja tablicowa

▶ **PUSH(x, S)**



Stos pełny, gdy $TOP = 0$

▶ **x = POP(S)**



Stos pusty, gdy $TOP = MAXLENGTH$

Przykład 110₍₁₎

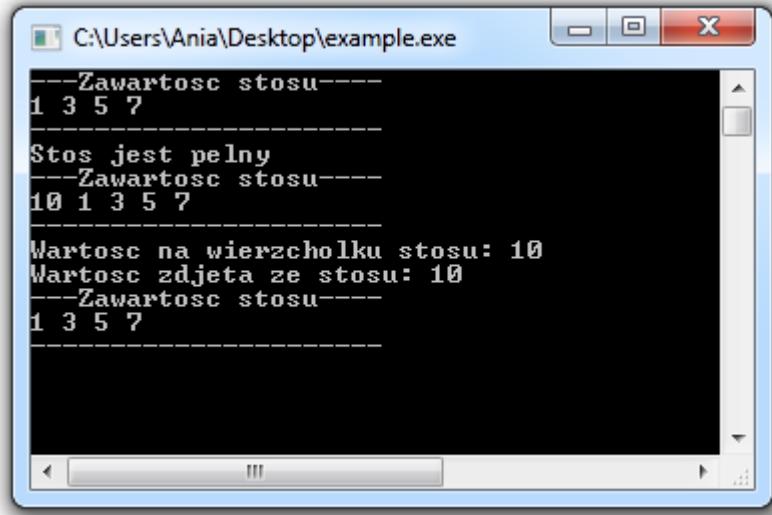
```
#include <stdio.h>

#define MAXLENGTH 5
/*****************/
struct stack{
    int top;
    int elements[MAXLENGTH];
};

/*****************/
int top (struct stack S);
int pop (struct stack* S);
int empty (struct stack S);
void print(struct stack S);
void makenull(struct stack* S);
void push (int x, struct stack* S);
/*****************/
```

Przykład 110₍₂₎

```
int main(void){\n\n    int x;\n    struct stack stos;\n\n    makenull(&stos);\n\n    push (7, &stos);\n    push (5, &stos);\n    push (3, &stos);\n    push (1, &stos);\n\n    print(stos);\n\n    push (10, &stos);\n    push (11, &stos);\n\n    print(stos);\n\n    x = top(stos);\n    printf("Wartosc na wierzcholku stosu: %d\n", x);\n\n    x = pop(&stos);\n\n    printf("Wartosc zdjeta ze stosu: %d\n", x);\n\n    print(stos);\n\n    return 0;\n}
```



Przykład 110₍₃₎

```
void makenull(struct stack* S){  
    S->top = MAXLENGTH;  
}  
/******************************************/  
int empty (struct stack S){  
  
    if(S.top == MAXLENGTH)  
        return 1;  
    else  
        return 0;  
}  
/******************************************/  
int top (struct stack S){  
  
    if(empty(S)){  
        printf("Stos jest pusty\n");  
        return -1;  
    }  
    else  
        return S.elements[S.top];  
}  
/******************************************/  
void push (int x, struct stack* S){  
  
    if (S->top == 0)  
        printf("Stos jest pelny\n");  
    else {  
        S->top --;  
        S->elements[S->top] = x;  
    }  
}
```

Przykład 110₍₄₎

```
void print(struct stack S){  
    int i;  
    printf(" ---Zawartosc stosu---\n");  
    if(empty(S)){  
        printf("Stos jest pusty\n");  
    }  
    else{  
        for(i=S.top; i<MAXLENGTH; i++){  
            printf("%d ", S.elements[i]);  
        }  
    }  
    printf("\n-----\n");  
}  
  
/******************************************/  
  
int pop (struct stack* S){  
    int x;  
    if(empty(*S)){  
        printf("Stos jest pusty\n");  
        return -1;  
    }  
    else{  
        x = S->elements[S->top];  
        S->top ++;  
    }  
    return x;  
}
```

Dynamiczne struktury danych

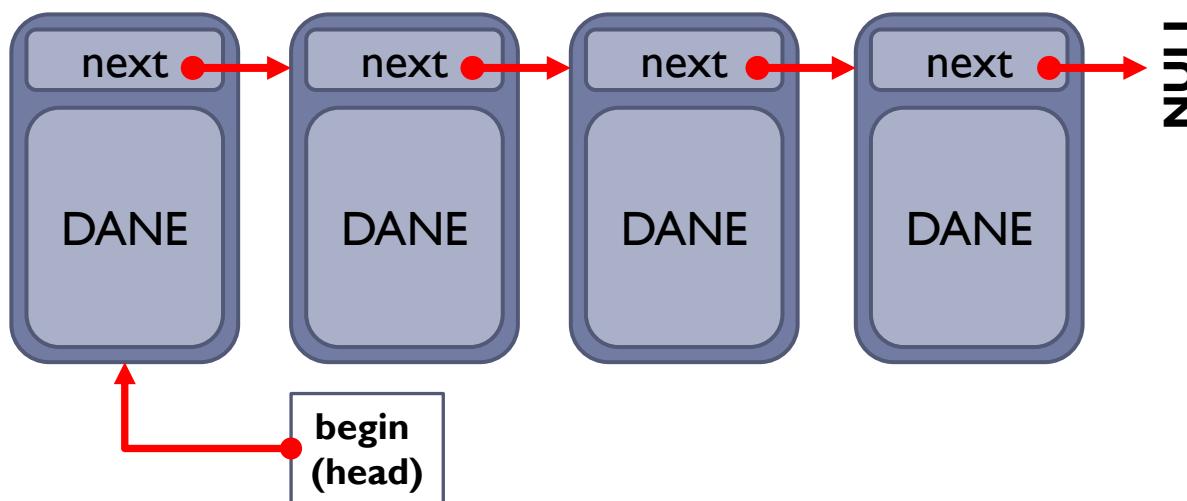
- ▶ **Dynamiczne struktury** danych to takie, których organizacja jest budowana w czasie wykonania programu. Wymaga to:
 - ▶ **dynamicznego zarządzania pamięcią** (bieżąca alokacja i zwalnianie);
 - ▶ organizacji danych w struktury (elementy) zawierające m.in. **powiązania pomiędzy obiektami**
- ▶ **Ze względu na organizację oraz sposoby dołączania, wstawiania i usuwania elementów, dynamicznych struktur danych można wyróżnić m.in.:**
 - ▶ **kolejkę** (dołączanie elementów tylko na jednym końcu, usuwanie elementów - tylko w drugim – przeciwnielego końca);
 - ▶ **stos** (dołączanie i usuwanie elementów tylko na jednym końcu);
 - ▶ **listy** (dołączanie i usuwanie elementów w dowolnym miejscu listy).
- ▶ **Istotną właściwością dynamicznych struktur danych jest wprowadzenie do elementów co najmniej jednego powiązania (wskaźnika) z sąsiadującymi elementami.**

Lista

- ▶ **Lista** – liniowa struktura danych, w której elementy ułożone są w porządku liniowym
- ▶ Operacje dodawania i usuwania elementów do listy można wykonywać w jej **dowolnym miejscu**, pomiędzy elementem początkowym i końcowym (włącznie)
- ▶ Rozróżnia się następujące rodzaje list:
 - ▶ **lista jednokierunkowa** (dla każdego elementu listy oprócz ostatniego zdefiniowany jest **element następujący**)
 - ▶ **lista dwukierunkowa** (dla każdego elementu listy zdefiniowany jest **element następujący** (z wyjątkiem ostatniego) oraz **element poprzedzający** (z wyjątkiem pierwszego))
 - ▶ **lista cykliczna** (nie występuje pierwszy ani ostatni element)

Lista jednokierunkowa

ang. *singly-linked list*



- ▶ Każdy z elementów listy jednokierunkowej zawiera dwa komponenty: **dane** oraz **wskaźnik do następnej komórki tego samego typu**;
- ▶ Listę jednokierunkową można przeglądać „**w przód**” począwszy od głowy (**head**), zgodnie z kierunkiem wskaźników **next**.

Element listy:

```
struct node{  
    struct node* next;  
    TYP dane;  
};
```

Lista:

```
struct list{  
    struct node* head;  
};
```

Podstawowe operacje:

Dodawanie elementu:

- na początek listy
- na koniec listy
- w środku listy

Usunięcie elementu:

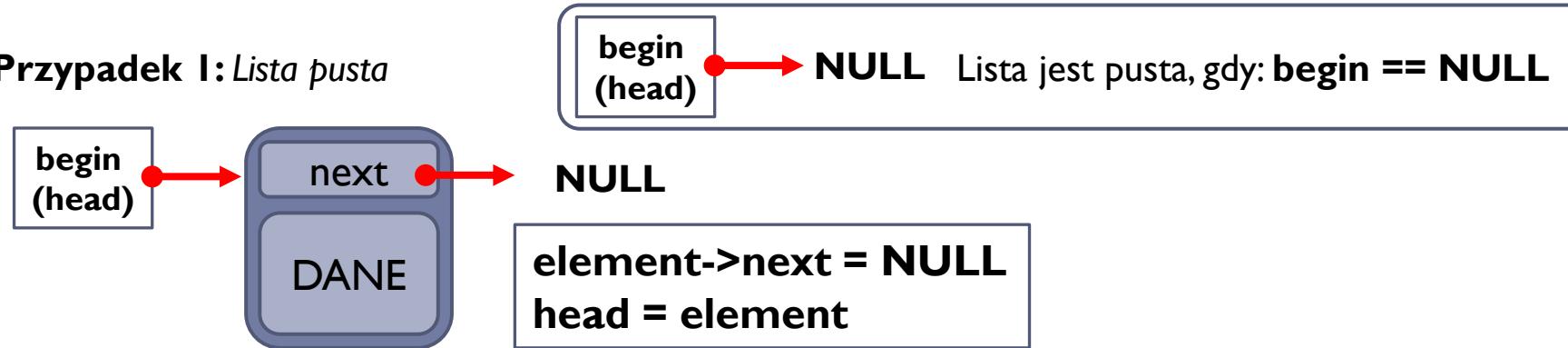
- z początku listy
- z końca listy
- ze środka listy

Przeglądanie (w jedną stronę)

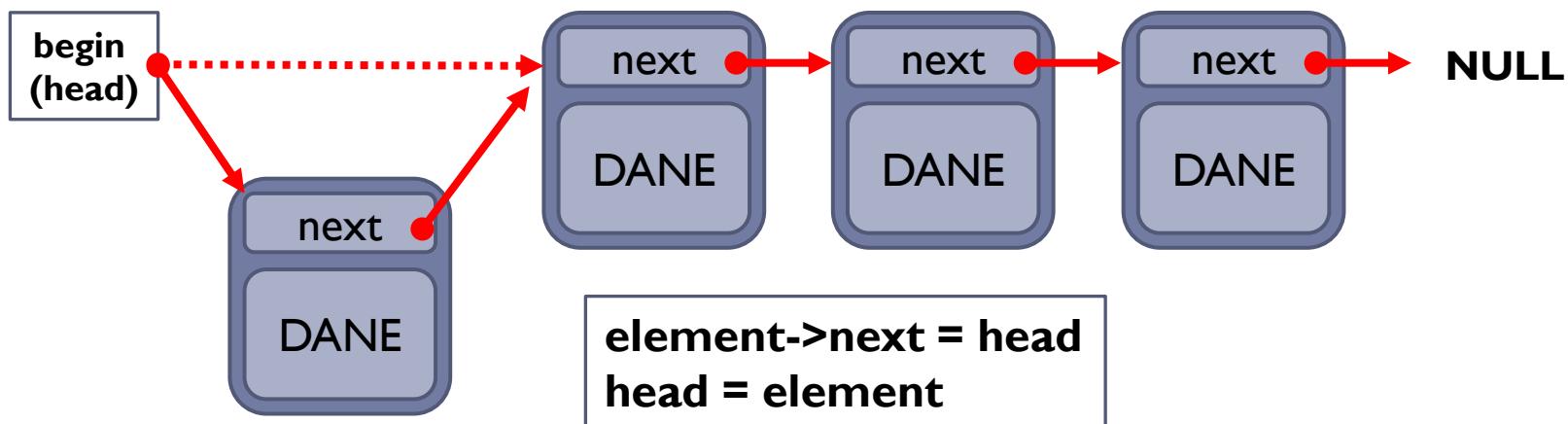
Lista jednokierunkowa

▶ Dodanie elementu na początek:

Przypadek 1: Lista pusta



Przypadek 2: Na liście sq elementy



Lista jednokierunkowa

▶ Dodanie elementu na początek:

Wskaźnik do listy

```
void push_front(struct list* l, struct node* el){  
    if (l->head == NULL)  
    {  
        el->next = NULL;  
        l->head = el;  
    }  
  
    else  
    {  
        el->next = l->head;  
        l->head = el;  
    }  
}
```

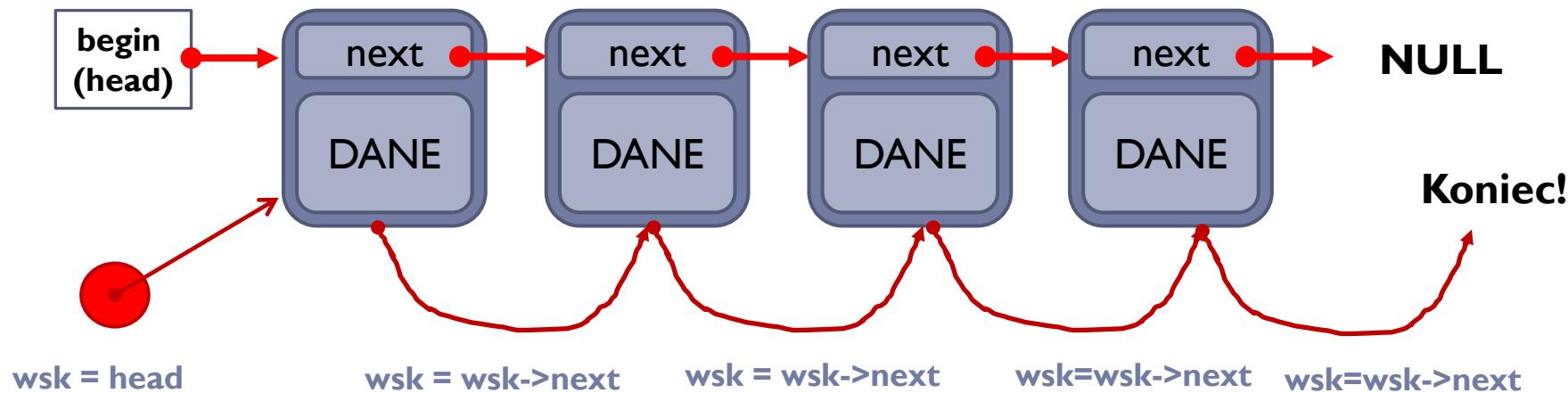
Przypadek pustej listy

Przypadek nie pustej listy

Lista jednokierunkowa

▶ Przeglądanie listy:

- ▶ realizowane za pomocą wskaźnika, który początkowo ustawiony jest na początek listy, a następnie „wędruje” wskaźnikami *next* do poszczególnych elementów, aż do napotkania wartości **NULL**

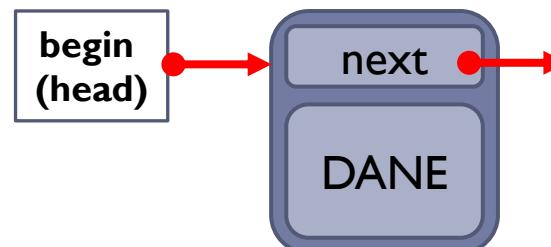


```
void moveForward(struct list* l){  
    struct node* wsk = l->head;  
    while(wsk) {  
        wsk = wsk->next;  
        ...  
    }  
}
```

Lista jednokierunkowa

Dodanie elementu na koniec

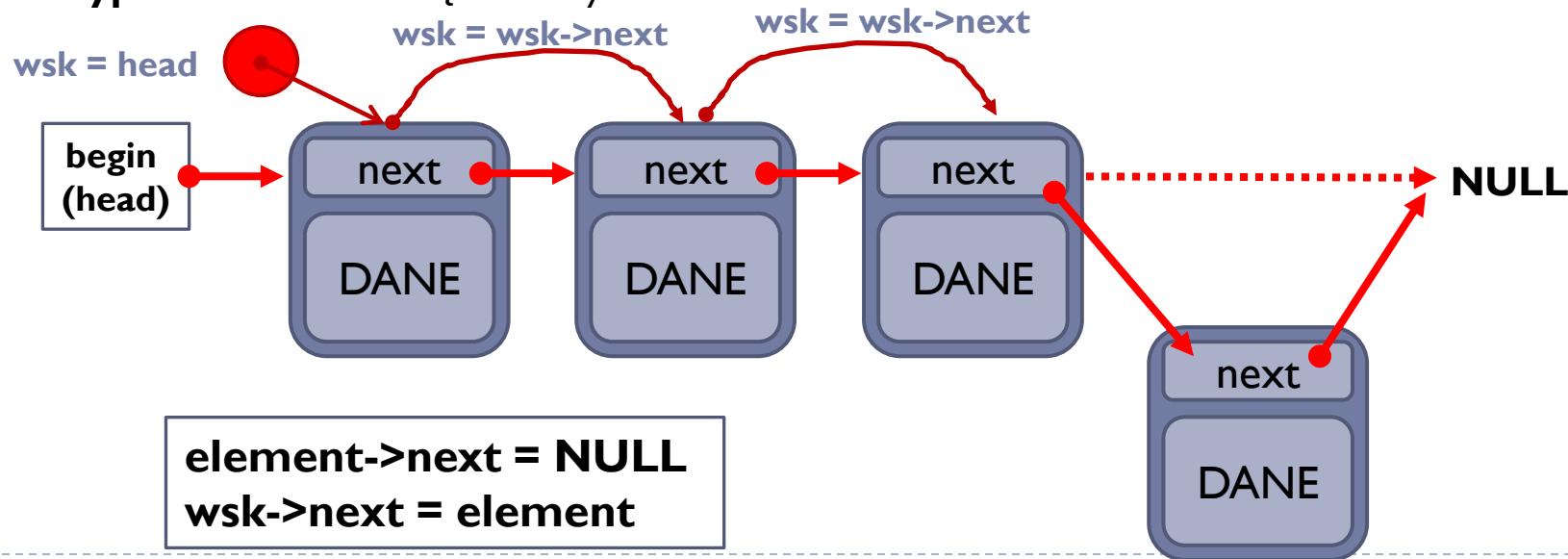
Przypadek 1: Lista pusta



Lista jest pusta, gdy: $\text{begin} == \text{NULL}$

$\text{element} \rightarrow \text{next} = \text{NULL}$
 $\text{head} = \text{element}$

Przypadek 2: Na liście nq elementów



Lista jednokierunkowa

Dodanie elementu na koniec

Wskaźnik do listy

```
void push_back(struct list *l, struct node* el)
```

```
{  
    if (l->head == NULL) {  
        el->next = NULL;  
        l->head = el;  
    }  
}
```

```
else  
{
```

```
    struct node *wsk = l->head;
```

```
    while(wsk->next!=NULL)  
        wsk = wsk->next;
```

```
    el->next = NULL;  
    wsk->next = el;
```

```
}
```

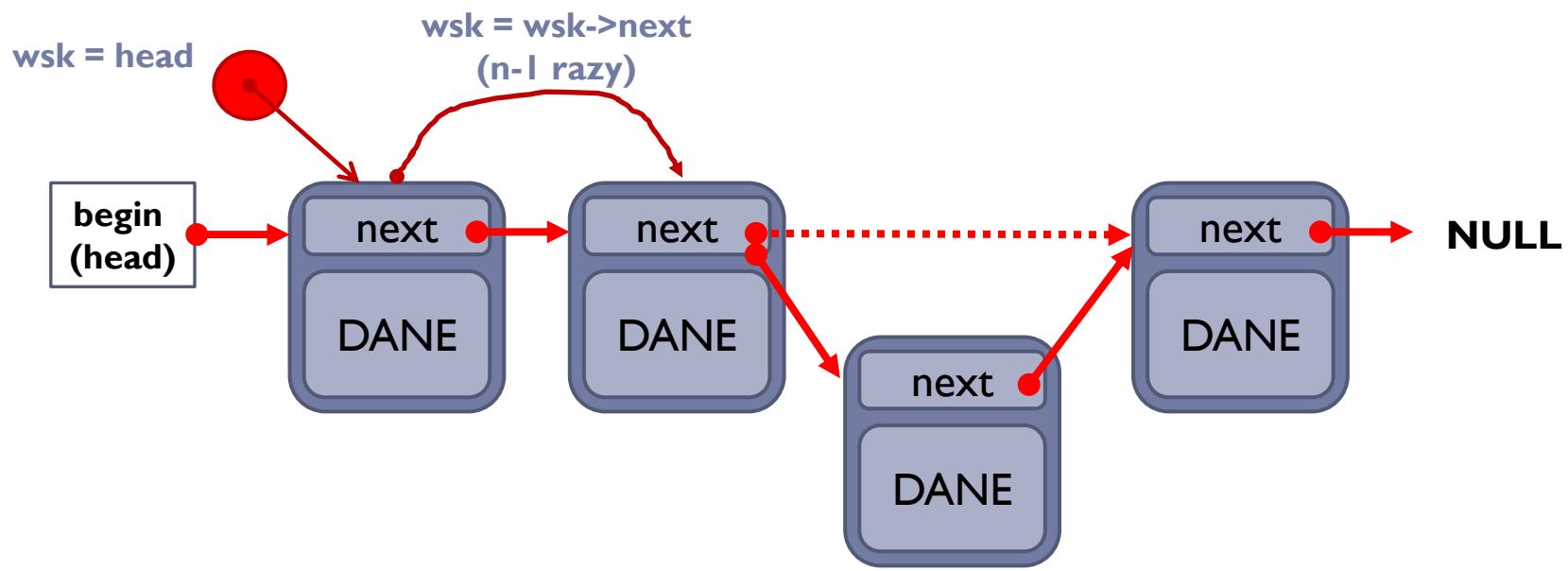
Przypadek pustej listy

Przesunięcie na koniec listy

Dodanie elementu

Lista jednokierunkowa

▶ Dodawanie elementu na n-tej pozycji:



```
element->next = wsk->next  
wsk->next = element
```

(zakładamy, że element był wcześniej zaalokowany i wypełniony danymi)

Lista jednokierunkowa

Dodawanie elementu na n-tej pozycji:

Wskaźnik do listy

pozycja

```
void insert(struct list *l, int n, struct node* el){
```

```
    int i = 0;
```

```
    int len = listLength(l);
```

Sprawdzenie długości listy

```
    if(n<=1)
```

```
        push_front(l, el);
```

Dodanie elementu na początek listy

```
    else if(n>len)
```

```
        push_end(l, el);
```

Dodanie elementu na koniec listy

```
    else {
```

```
        struct node *wsk = l->head;
```

```
        for(i=1; i<n-1; i++)
            wsk = wsk->next;
```

Przesunięcie na pozycję n-1

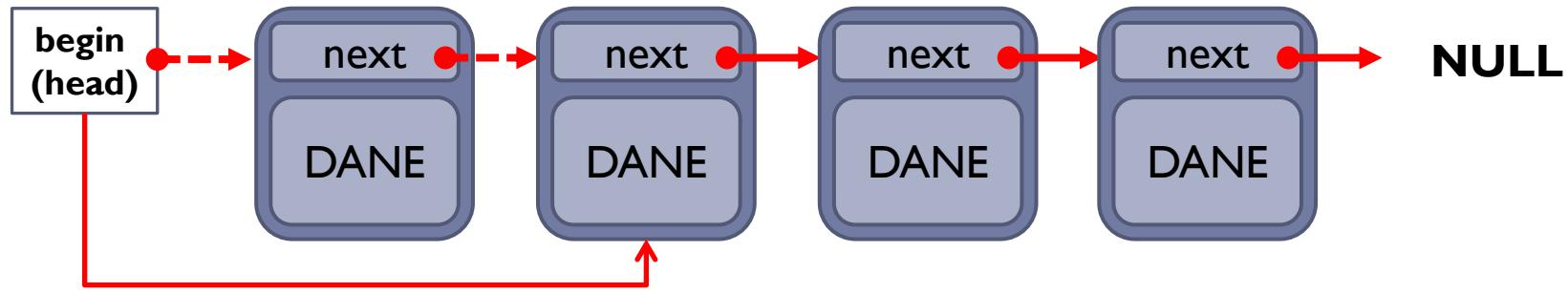
```
        el->next = wsk->next;
        wsk->next = el;
```

Dodanie elementu

```
}
```

Lista jednokierunkowa

► **Usuwanie elementu z początku listy:**



```
temp = head
head = head->next;
+ zwolnienie pamięci lub zwrócenie elementu temp
```

Konieczność „zapamiętania” starej głowy w zmiennej pomocniczej **temp**, aby nie utracić do niej dostępu.

Aby zrealizować usuwanie, lista nie może być pusta

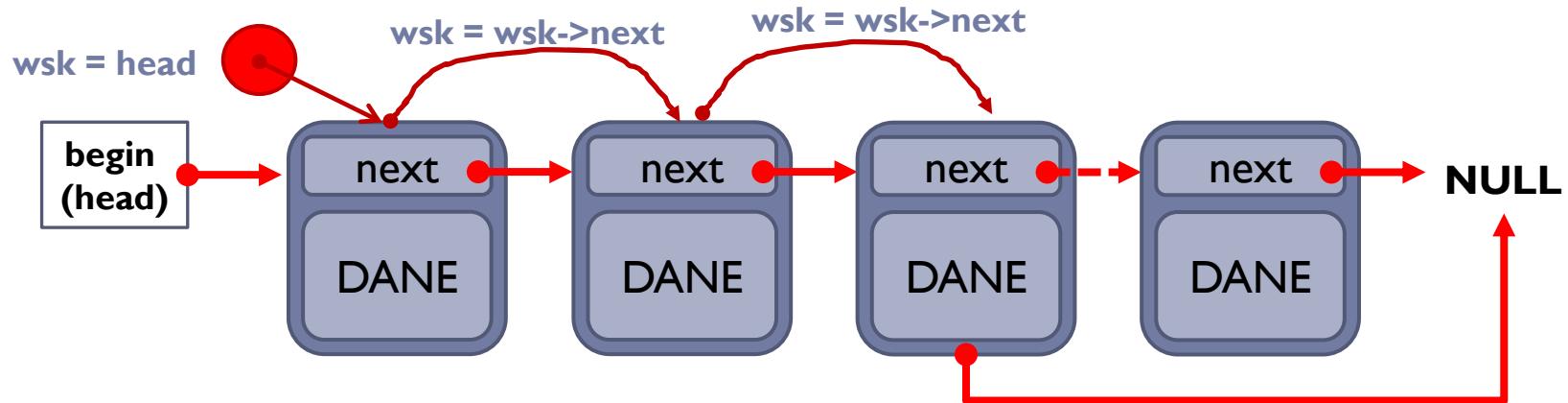
Lista jednokierunkowa

▶ **Usuwanie elementu z początku listy:**

```
struct node* pop_front(struct list *l) {  
  
    if (l->head != NULL) {  
        struct node* wsk = l->head;  
        l->head = l->head->next;  
        return wsk;  
    }  
    else{  
        printf("Lista pusta! \n");  
        return NULL;  
    }  
}
```

Lista jednokierunkowa

▶ Usuwanie elementu z końca listy:



```
temp = wsk->next  
wsk->next = NULL  
+ zwolnienie pamięci lub zwrócenie elementu temp
```

Usuwanie elementu z końca listy:

Lista jednokierunkowa

```
struct node* pop_back(struct list *l){  
    struct node* temp;  
    if (l->head != NULL) {  
        if(l->head->next) {  
            struct node* wsk = l->head;  
            while((wsk->next)->next!=NULL)  
                wsk = wsk->next;  
            }  
            temp = wsk->next;  
            wsk->next = NULL;  
        }  
        else{  
            temp = l->head;  
            l->head = NULL;  
        }  
    }  
    else{  
        printf("Lista pusta!\n");  
        temp = NULL;  
    }  
    return temp;  
}
```

Sprawdzenie, czy lista nie jest pusta

Sprawdzenie, czy jest więcej niż 1 element

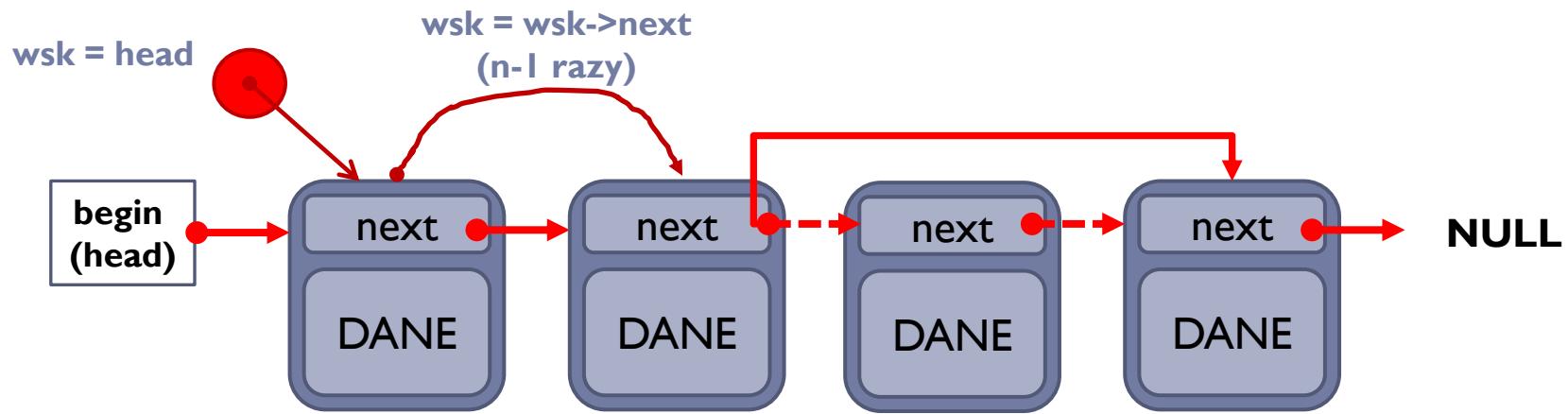
Przesunięcie na koniec listy

Usunięcie elementu

Lista z jednym elementem

Lista jednokierunkowa

▶ Usuwanie elementu z n-tej pozycji:



```
temp = wsk->next  
wsk->next = wsk->next->next  
+ zwolnienie pamięci lub zwrócenie elementu temp
```

Lista jednokierunkowa

numer pozycji

Usuwanie elementu z n-tej pozycji:

```
struct node* del(struct list *l, int n) {
```

```
    int i = 0;
    int len = listLength(l);
    struct node* temp;
```

Sprawdzenie długości listy

```
    if(n<=1)
        temp = pop_front(l);
```

Usunięcie elementu z początku listy

```
    else if(n>len)
        temp = pop_back(l);
    else {
```

Usunięcie elementu z końca listy

```
        struct node *wsk = l->head;
```

```
        for(i=1; i<n-1; i++)
            wsk = wsk->next;
```

Przesunięcie na pozycję n-1

```
        temp = wsk->next;
        wsk->next = wsk->next->next;
```

Usunięcie elementu

```
    }
```

```
    return temp;
```

```
}
```

Przykład 111(1)

```
#include <stdio.h>
#include <stdlib.h>
/*****************/
struct node{
    struct node *next;
    int dane;
};
/*****************/
struct list{
    struct node* head;
};
/*****************/
void init(struct list *l);
void print(struct list *l);
struct node* createElement(int data);
struct node* pop_back(struct list *l);      /*zdefiniowane wcześniej*/
struct node* pop_front(struct list *l);      /*zdefiniowane wcześniej*/
struct node* del(struct list *l, int n);     /*zdefiniowane wcześniej*/
void push_back(struct list *l, struct node* el); /*zdefiniowane wcześniej*/
void push_front(struct list *l, struct node* el); /*zdefiniowane wcześniej*/
void insert(struct list *l, int n, struct node* el); /*zdefiniowane wcześniej*/
int listLength(struct list *l);
void clear(struct list *l);
```

Przykład 111₍₂₎

```
int main(){

    struct list l;
    init (&l);

    struct node* el1 = createElement(11);
    struct node* el2 = createElement(12);
    struct node* el3 = createElement(13);
    struct node* el4 = createElement(0);
    struct node* el5 = createElement(100);

    push_front(&l, el1); push_front(&l, el2); push_front(&l, el3); print(&l);
    insert(&l, 2, el4); print(&l); printf("Dlugosc: %d\n", listLength(&l));
    push_back(&l, el5); print(&l);
    printf("Zdjeto: %d\n", (del(&l, 3))->dane); print(&l); //wyciek pamięci
    printf("Zdjeto: %d\n", (pop_back(&l))->dane); print(&l); //wyciek pamięci
    printf("Zdjeto: %d\n", (pop_front(&l))->dane); print(&l); //wyciek pamięci

    printf("\nDlugosc: %d\n", listLength(&l));

    clear(&l);
    printf("\nDlugosc: %d\n", listLength(&l));

    return 0;
}
```

```
C:\Users\Ania\Dropbox\example.exe
13      12      11
13      0       12      11
Dlugosc: 4
13      0       12      11      100
Zdjeto: 12
13      0       11      100
Zdjeto: 100
13      0       11
Zdjeto: 13
0       11
Dlugosc: 2
Dlugosc: 0
```

Przykład 111(3)

```
void init(struct list *l){  
    l->head = NULL;  
}  
/*********************************************/  
struct node* createElement(int data){  
  
    struct node* el = (struct node*) malloc (sizeof(struct node));  
    el->dane = data;  
    el->next = NULL;  
  
    return el;  
}  
/*********************************************/  
void print(struct list *l){  
  
    struct node* wsk = l->head;  
  
    while(wsk){  
        printf("%d\t", wsk->dane);  
        wsk = wsk -> next;  
    }  
  
    printf("\n");  
}
```

Przykład 111₍₄₎

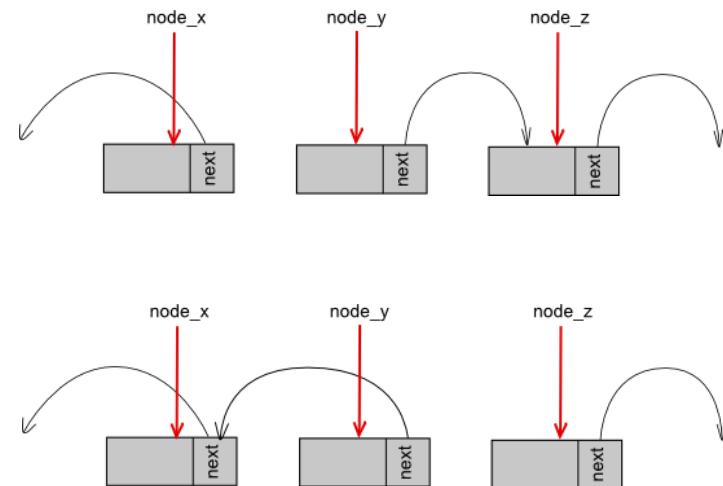
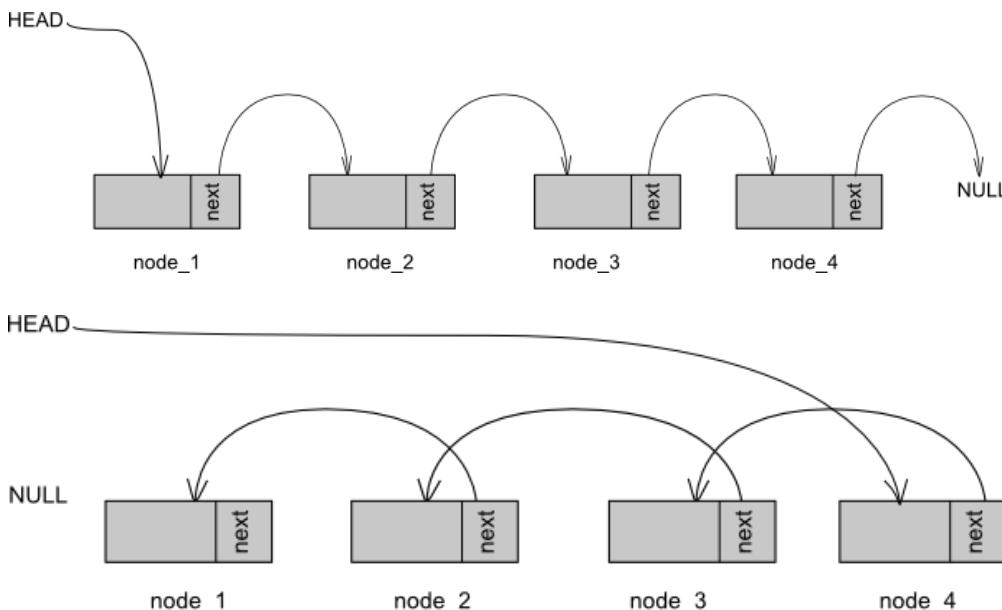
```
int listLength(struct list *l){  
    int i = 0;  
  
    struct node* wsk = l->head;  
  
    while(wsk){  
        i++;  
        wsk = wsk->next;  
    }  
    return i;  
}  
  
/********************************************/  
  
void clear(struct list *l){  
  
    struct node *wsk = l->head;  
    struct node *temp;  
  
    while(wsk){  
        temp = wsk;  
        wsk = wsk->next;  
        free(temp);  
    }  
  
    l->head = NULL;  
}
```

Lista jednokierunkowa

▶ Odwracanie kolejności elementów na liście jednokierunkowej

▶ Jeżeli lista nie jest pusta:

- ▶ zapamiętać adres pierwszego elementu
- ▶ dopóki istnieje następnik zapamiętanego elementu:
 - „wypiąć” następnik z listy
 - przenieść następnik na początek

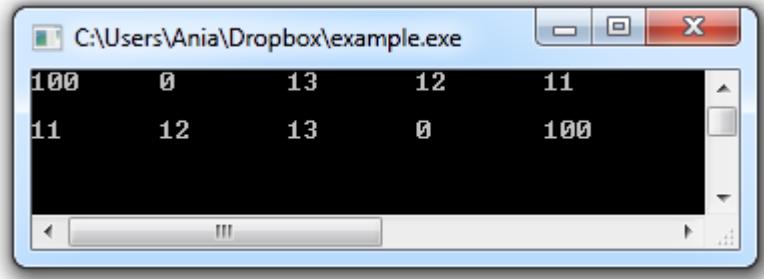


Przykład 112(1)

```
void reverse(struct list *l) {  
  
    struct node* wsk;           /*kontynuacja implementacji  
    struct node* temp;          listy z Przykładu 111*/  
  
    if(l->head) {  
  
        wsk = l->head;         zapamiętanie pierwszego elementu  
  
        while(wsk->next) {     dopóki istnieje następnik zapamiętanego elementu  
  
            temp = wsk->next;   zapamiętanie następnika  
  
            wsk->next = temp->next;  wypisanie następnika z listy  
  
            temp->next = l->head;  ustawienie następnika  
            l->head = temp;       zapamiętanego elementu na  
                                początku listy  
        }  
    }  
}
```

Przykład 112₍₂₎

```
int main() {  
  
    struct list l;  
    init (&l);  
  
    struct node* el1 = createElement(11);  
    struct node* el2 = createElement(12);  
    struct node* el3 = createElement(13);  
    struct node* el4 = createElement(0);  
    struct node* el5 = createElement(100);  
  
    push_front(&l, el1);  
    push_front(&l, el2);  
    push_front(&l, el3);  
    push_front(&l, el4);  
    push_front(&l, el5);  
  
    print(&l);  
  
    reverse(&l);  
  
    print(&l);  
    clear(&l);  
    return 0;  
}
```



Lista jednokierunkowa

▶ **Liniowe przeszukiwanie listy:**

- ▶ począwszy od głowy listy należy przejrzeć wszystkie elementy aż do napotkania poszukiwanego elementu lub końca listy
- ▶ zwrócić adres znalezionego elementu, lub *NULL* gdy element nie został znaleziony

Przykład 113

```
int main(){

    struct list l;  init (&l);

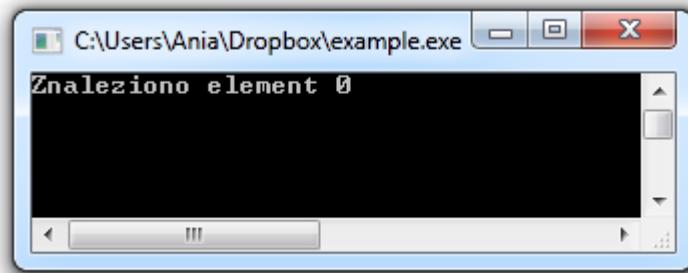
    push_front(&l, createElement(11));
    push_front(&l, createElement(12));
    push_front(&l, createElement(13));
    push_front(&l, createElement(0));
    push_front(&l, createElement(100));

    struct node* wsk = find(&l, 0);

    if(wsk!=NULL)
        printf("Znaleziono element %d\n", wsk->dane);
    else
        printf("Elementu nie znaleziono\n");

    clear(&l)
    return 0;
}
```

```
struct node* find(struct list* l, int d) {
    struct node* wsk = l->head;
    while(wsk) {
        if(wsk->dane == d) {
            return wsk;
        }
        wsk = wsk->next;
    }
    return NULL;
}
```



/*kontynuacja implementacji
listy z Przykładu 111*/

Lista jednokierunkowa

- ▶ **Podział listy jednokierunkowej na dwie listy**
 - ▶ przez wybieranie z początku rozdzielanej listy kolejnych elementów i wprowadzanie ich raz na jedną, raz na drugą listę docelową

Przykład 114₍₁₎

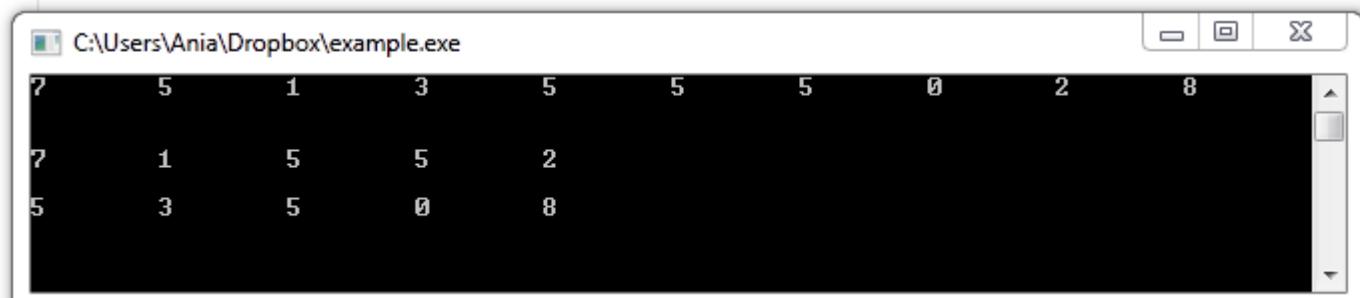
Lista oryginalna
const, bo nie będzie modyfikowana

Listy
wynikowe

```
void split(const struct list *l, struct list *l1, struct list *l2){  
    struct node* wsk = l->head;  
  
    int s = 1;  
  
    while(wsk) {  
  
        if(s%2) push_back(l1, createElement(wsk->dane));  
        else push_back(l2, createElement(wsk->dane));  
  
        wsk = wsk->next;  
        s++;  
    }  
}
```

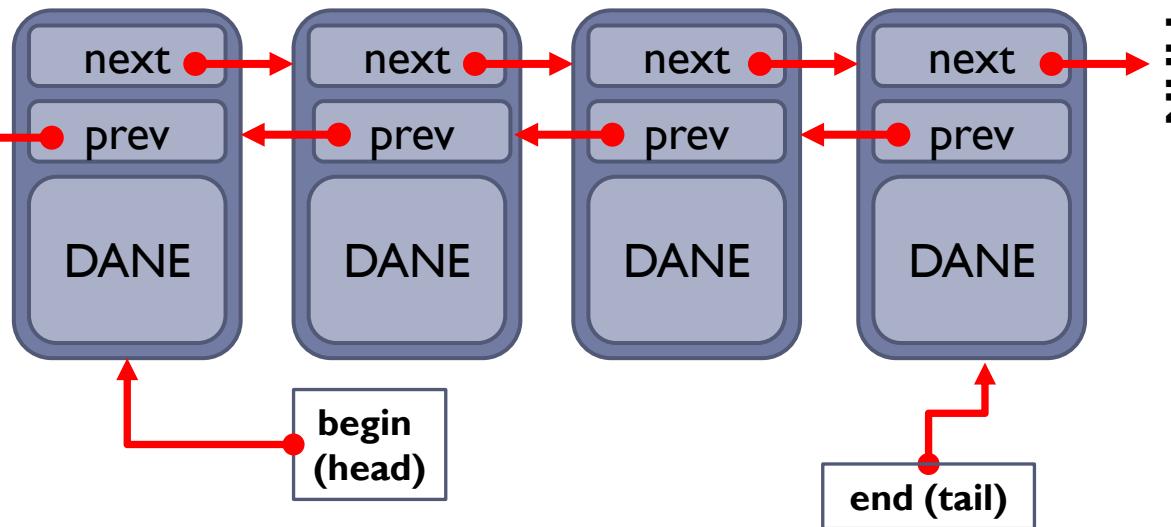
Przykład 114₍₂₎

```
int main() {  
  
    srand(time(NULL));  
    int i;  
  
    struct list l;  
    init (&l);  
  
    for(i=0; i<10; i++)  
        push_front(&l, createElement(rand()%10));  
  
    print(&l);  
  
    struct list l1, l2;  
    init (&l1);  
    init (&l2);  
  
    split(&l, &l1, &l2);  
  
    print(&l1);  
    print(&l2);  
  
    getchar();  
    clear(&l);  
    clear(&l1);  
    clear(&l2);  
  
    return 0;
```



Lista dwukierunkowa

NULL



- ▶ Każdy z elementów listy dwukierunkowej zawiera trzy komponenty: **dane** oraz **wskaźniki do poprzedniej i następnej komórki** tego samego typu;
- ▶ Listę dwukierunkową można przeglądać „**w przód**” począwszy od głowy (**head**), zgodnie z kierunkiem wskaźników **next**, oraz „**w tył**” począwszy od ogona (**tail**) zgodnie z kierunkiem wskaźników **prev**

Element listy:

```
struct node{  
    struct node* next;  
    struct node* prev;  
    TYP dane;  
};
```

Lista:

```
struct list{  
    struct node* head;  
    struct node* tail;  
};
```

Podstawowe operacje:

Dodawanie elementu:

- na początek listy
- na koniec listy
- w środku listy

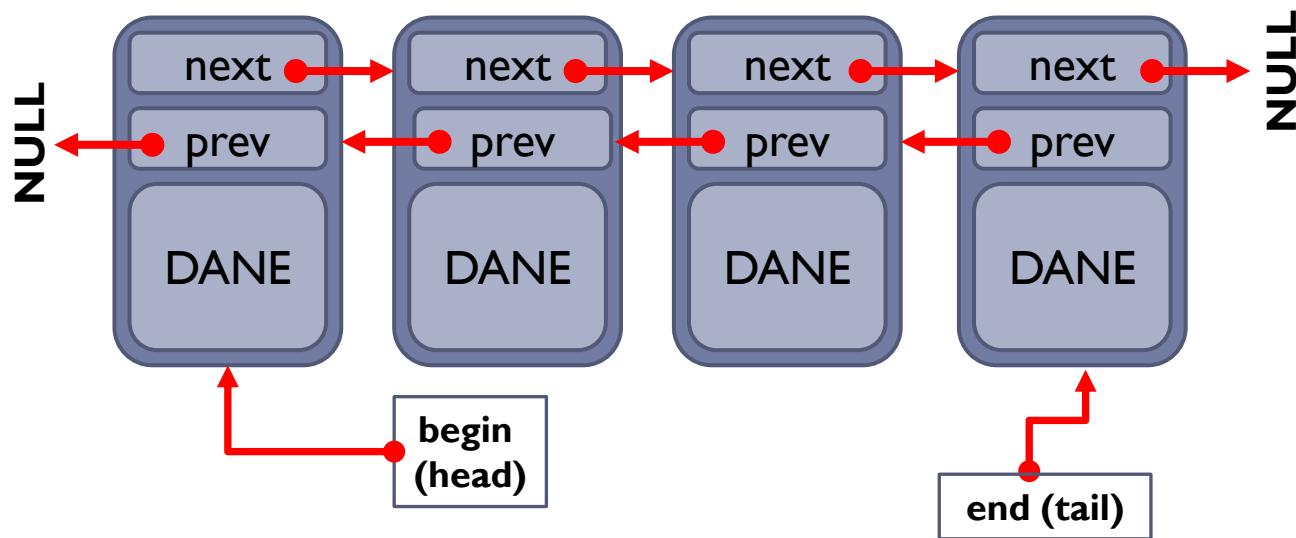
Usunięcie elementu:

- z początku listy
- z końca listy
- ze środka listy

Przeglądanie (w dwie strony)

Lista dwukierunkowa

ang. *doubly linked list*



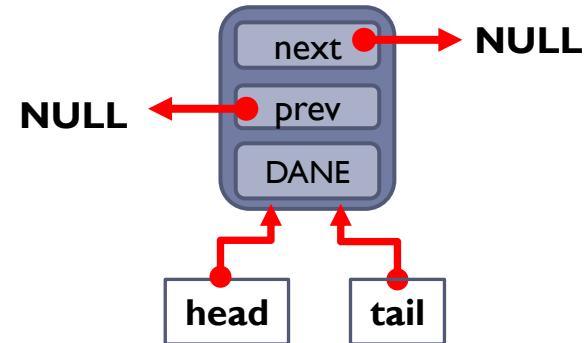
- ▶ Dla pustej listy: **head == tail == NULL**
- ▶ Dla pierwszego elementu listy: **head->prev == NULL**
- ▶ Dla ostatniego elementu listy: **tail->next == NULL**
- ▶ Dla listy jednoelementowej: **head == tail != NULL**

Lista dwukierunkowa

▶ Dodawanie elementu **element** na początek listy **list**:

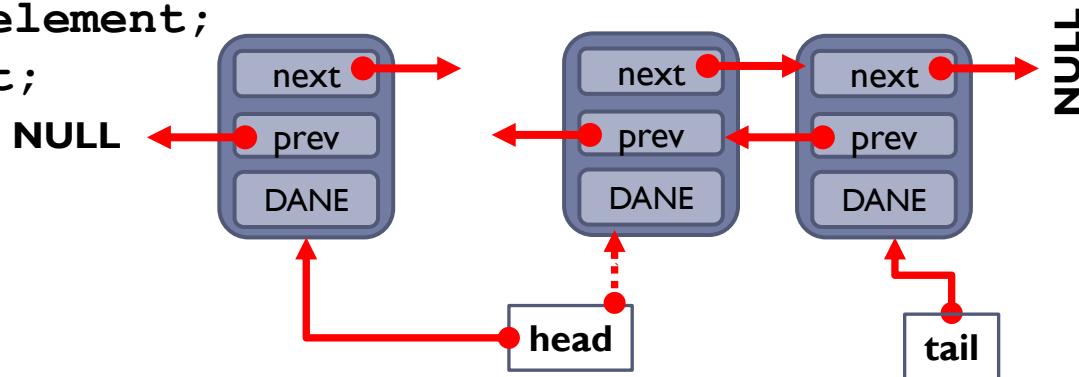
▶ Przypadek 1 – lista pusta

```
element->next = NULL;  
element->prev = NULL;  
list->head = element;  
list->tail = element;
```



▶ Przypadek 2 – lista niepusta

```
element->next = head;  
element->prev = NULL;  
list->head->prev = element;  
list->head = element;
```

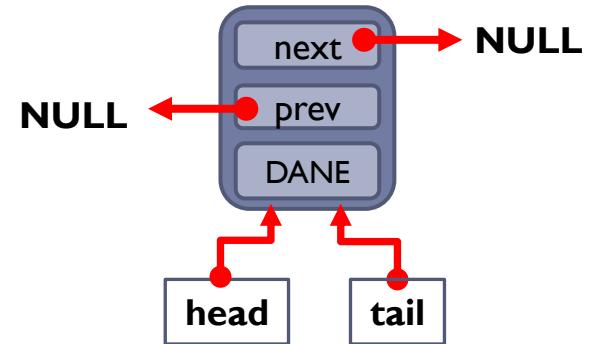


Lista dwukierunkowa

▶ Dodawanie elementu **element** na koniec listy **list**:

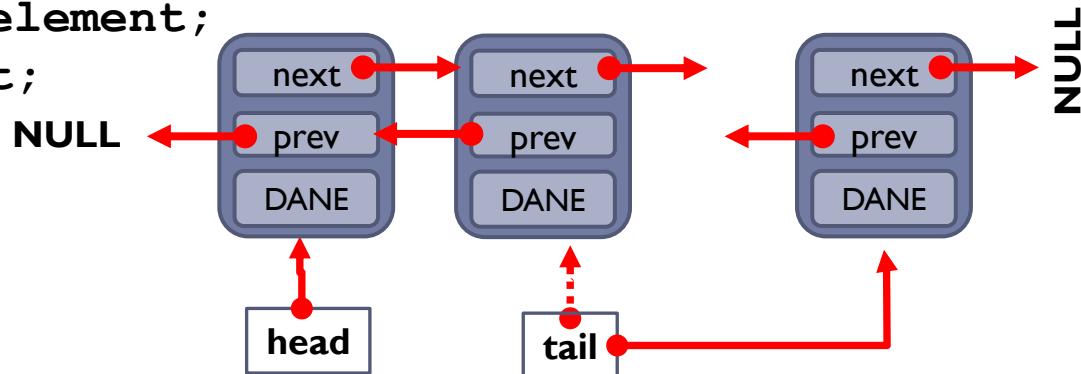
▶ Przypadek 1 – lista pusta

```
element->next = NULL;  
element->prev = NULL;  
list->head = element;  
list->tail = element;
```



▶ Przypadek 2 – lista niepusta

```
element->next = NULL;  
element->prev = list->tail;  
list->tail->next = element;  
list->tail = element;
```

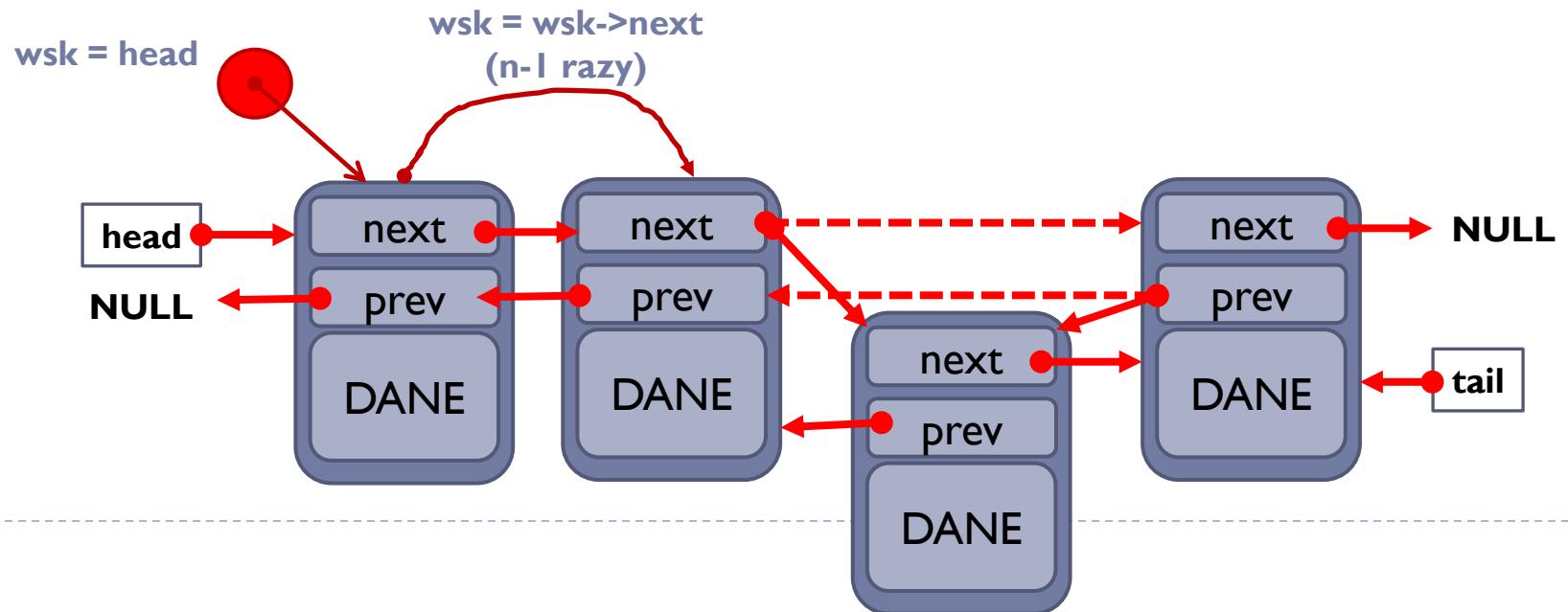


Lista dwukierunkowa

▶ Dodawanie elementu **element** w środku listy **list**:

- ▶ **wsk** – wskaźnik do elementu za którym dodajemy

```
element->next = wsk->next;  
element->prev = wsk;  
wsk->next = element;  
wsk->next->prev = element;
```



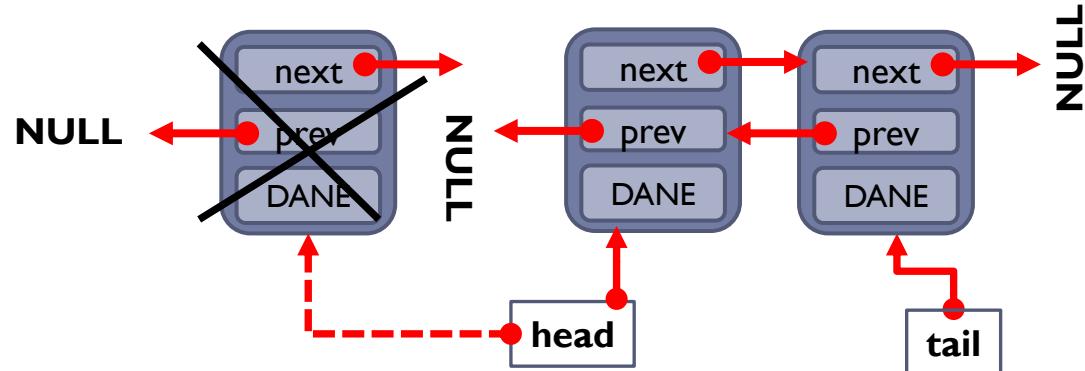
Lista dwukierunkowa

- ▶ Usuwanie elementu **element** z początku listy **list**:
 - ▶ Przypadek 1 – lista jest jednoelementowa

```
list->tail=NULL;  
free(list->head);  
list->head=NULL;
```

- ▶ Przypadek 2 – lista posiada więcej niż 1 elementów

```
temp=list->head->next;  
list->head->next->prev=NULL;  
free(list->head);  
list->head=temp;
```



Lista dwukierunkowa

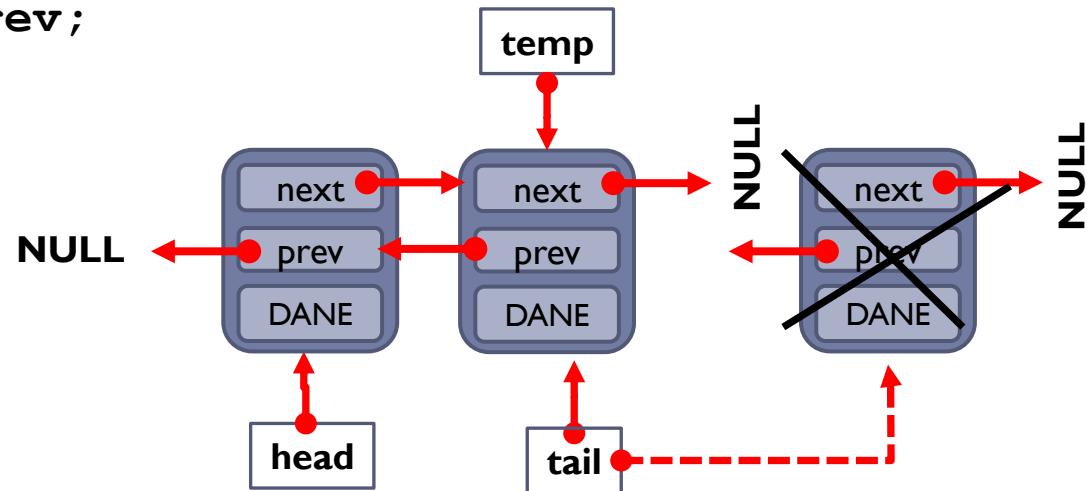
- ▶ Usuwanie elementu **element** z końca listy **list**:

- ▶ Przypadek 1 – lista jest jednoelementowa

```
list->tail=NULL;  
free(list->head);  
list->head=NULL;
```

- ▶ Przypadek 2 – lista posiada więcej niż 1 elementów

```
temp=list->tail->prev;  
temp->next=NULL;  
free(list->tail);  
list->tail=temp;
```

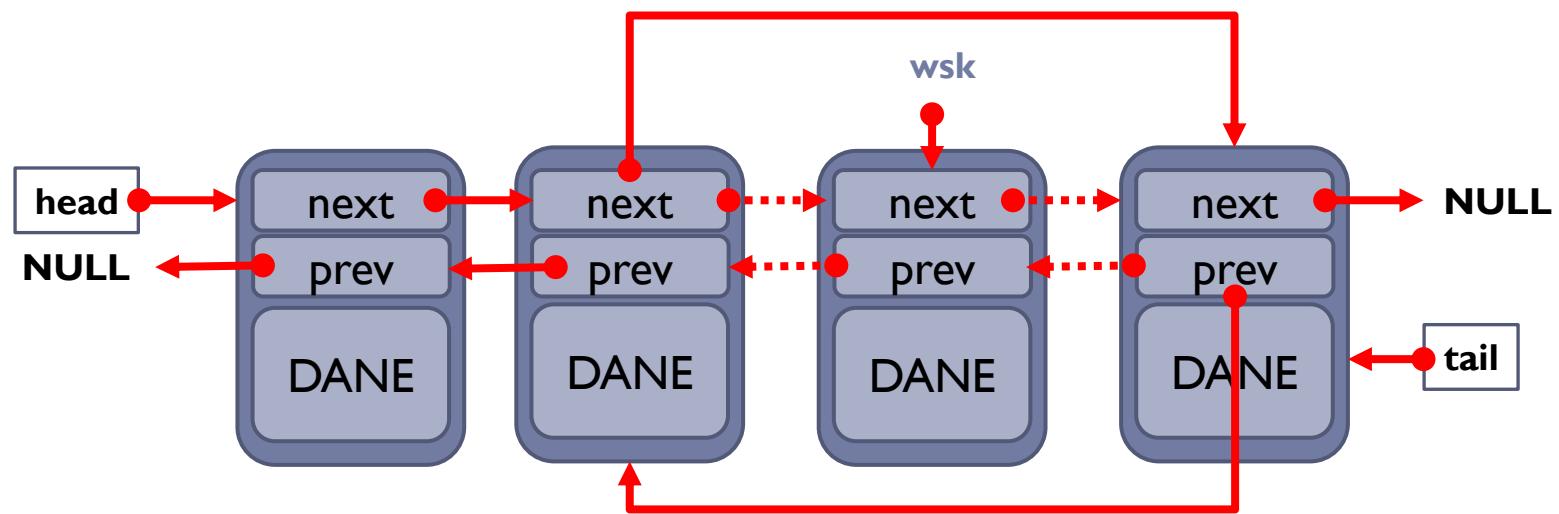


Lista dwukierunkowa

▶ Usuwanie elementu **element** ze środka listy *list*:

- ▶ **wsk** – wskaźnik do elementu który usuwamy

```
wsk->prev->next = wsk->next;  
wsk->next->prev = wsk->prev;
```



Lista dwukierunkowa

▶ Przeglądanie listy

- ▶ od głowy do ogona (ang. *forward*)

```
wsk = list->head;  
while(wsk) {  
    //do sth  
    wsk = wsk->next;  
}
```

- ▶ od ogona do głowy (ang. *backward*)

```
wsk = list->tail;  
while(wsk) {  
    //do sth  
    wsk = wsk->prev;  
}
```

Kolejka

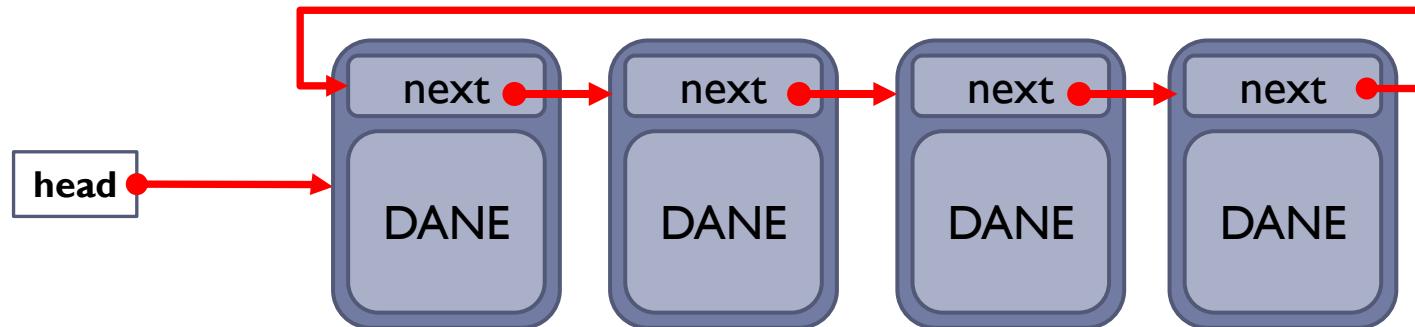
- ▶ Kolejka (ang. *queue*) jest szczególnym przypadkiem listy
 - ▶ jej elementy dodawane są na jednym końcu (zwany *tyłem*), a usuwane z drugiego końca (zwanego *czolem*);
 - ▶ oznaczana często skrótem **FIFO** (ang. *first-in-first-out*)
- ▶ Operacje podstawowe kolejki:
 - ▶ **MAKENULL(Q)** – usuwa wszystkie elementy z kolejki
 - ▶ **FRONT(Q)** – zwraca czolowy element kolejki
 - ▶ **ENQUEUE(x,Q)** – wstawia na tył kolejki element x
 - ▶ **DEQUEUE(Q)** – usuwa element z czoła kolejki
 - ▶ **EMPTY(Q)** – zwraca true kiedy kolejka jest pusta

Stos

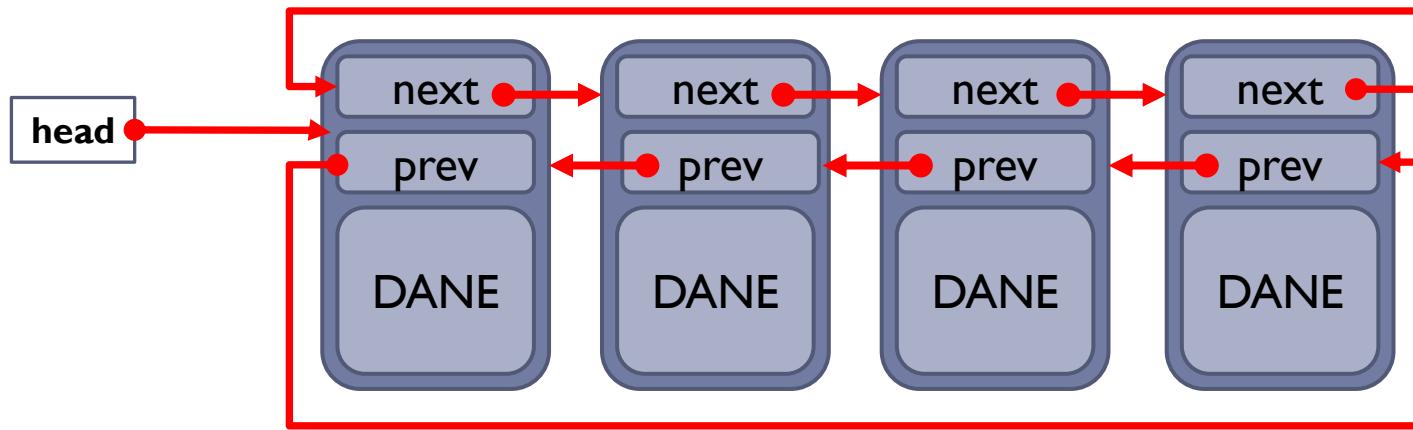
- ▶ Stos (ang. *stack*) jest szczególnym przypadkiem listy
 - ▶ jego elementy są dodawane i usuwane na jednym końcu (zwany *wierzchołkiem*);
 - ▶ oznaczana często skrótem *LIFO* (ang. *last-in-first-out*)
- ▶ Operacje podstawowe kolejki:
 - ▶ **MAKENULL(Q)** – usuwa wszystkie elementy ze stosu
 - ▶ **TOP(Q)** – zwraca szczytowy element stosu
 - ▶ **PUSH(x,Q)** – odkłada element x na wierzchołek stosu
 - ▶ **POP(Q)** – usuwa element z wierzchołka stosu
 - ▶ **EMPTY(Q)** – zwraca *true* kiedy stos jest pusty oraz *false* w przeciwnym

Lista cykliczna

▶ Jednokierunkowa



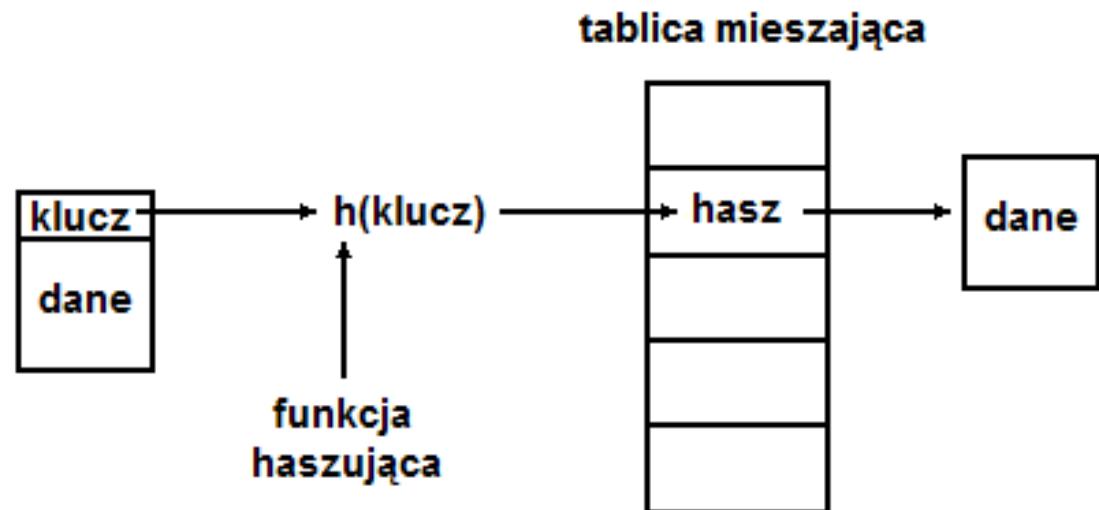
▶ Dwukierunkowa



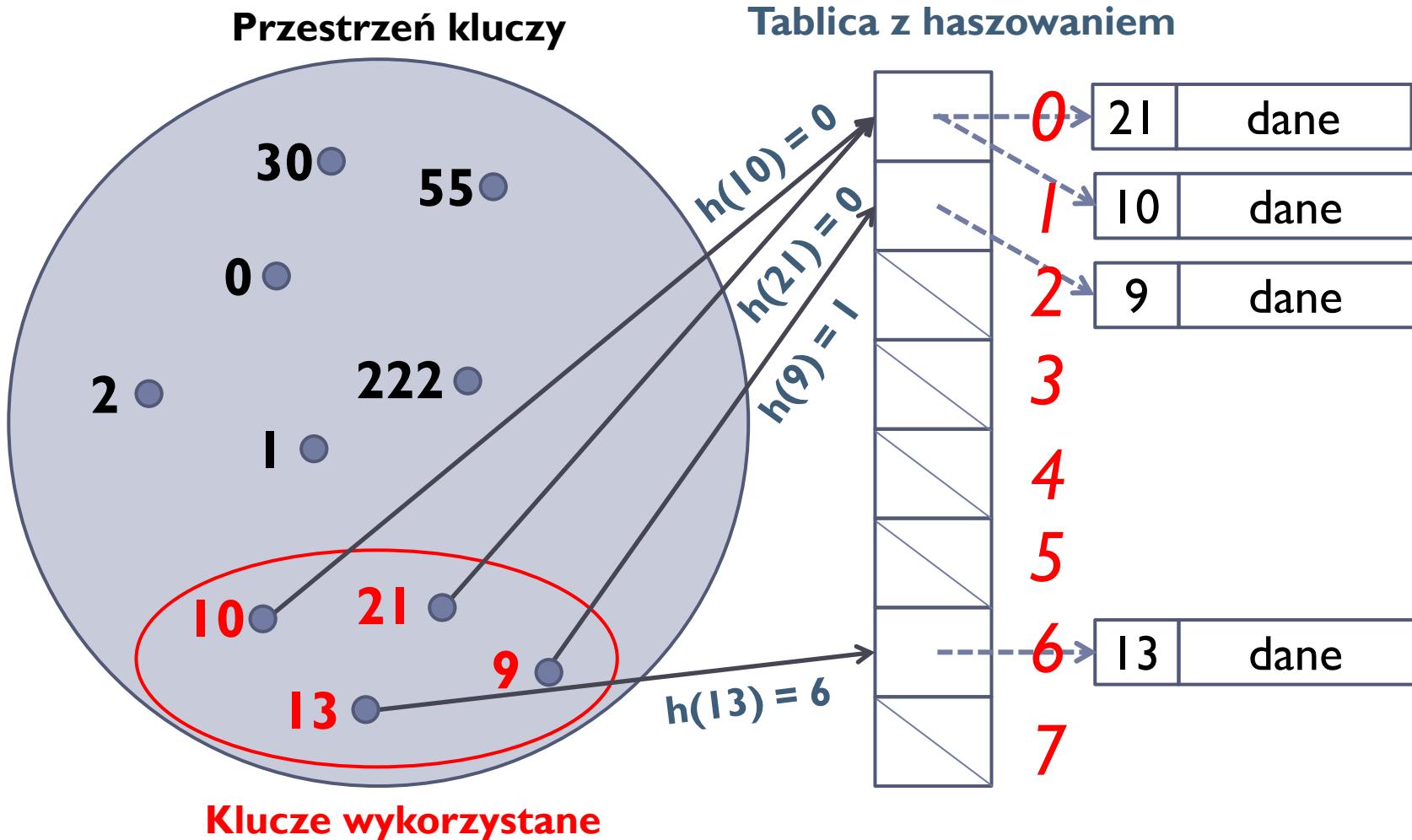
Haszowanie

- ▶ **Haszowanie** polega na uzyskaniu bezpośredniego odniesienia do elementów w tablicy za pomocą operacji arytmetycznych przekształcających klucz w adres tablicy
- ▶ Idea haszowania polega na odnalezieniu takiej funkcji $h(\cdot)$, która na podstawie danej opisanej przez klucz, wskazywałaby indeks w tablicy mieszającej, pod którym jest przechowana dana.

Funkcja $h(\cdot)$ jest zwana **funkcją haszującą** lub **funkcją skrótu**.



Haszowanie



Haszowanie

Przykładowe funkcje haszujące:

$$h(x) = x \% p$$

gdzie: p jest liczbą pierwszą

$$h(x) = (x*p + r) \% q$$

gdzie: p i q są liczbami pierwszymi, r jest liczbą całkowitą

$$h(x) = [((x*A) \bmod l)^*m]$$

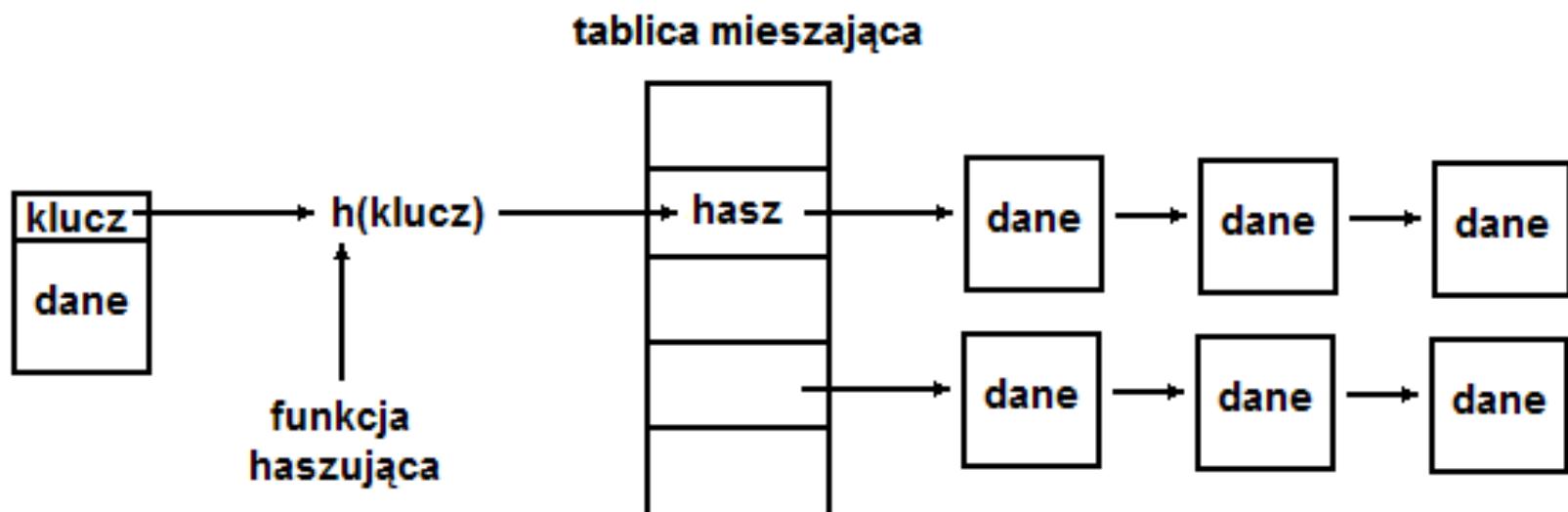
gdzie: A jest liczbą z przedziału $(0, l)$, m jest liczbą całkowitą

$$h(x) = (x[0] + x[1]*p + x[2]*p^2 + \dots + x[N-1]*p^{N-1}) \% q$$

gdzie: x jest napisem, a p i q są dużymi liczbami pierwszymi

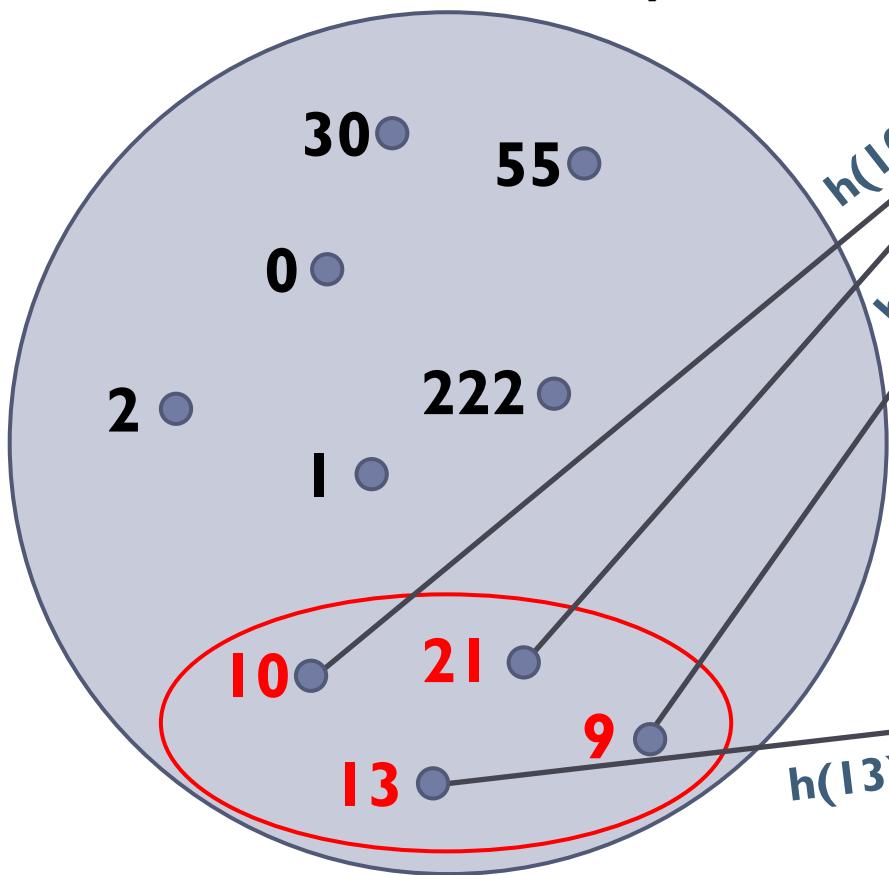
Haszowanie łańcuchowe

- ▶ **Haszowanie łańcuchowe** (ang. chaining) polega na tym, że wszystkie elementy, którym w wyniku haszowania odpowiada ta sama pozycja w tablicy haszującej, zostają umieszczone na jednej oddzielnej liście

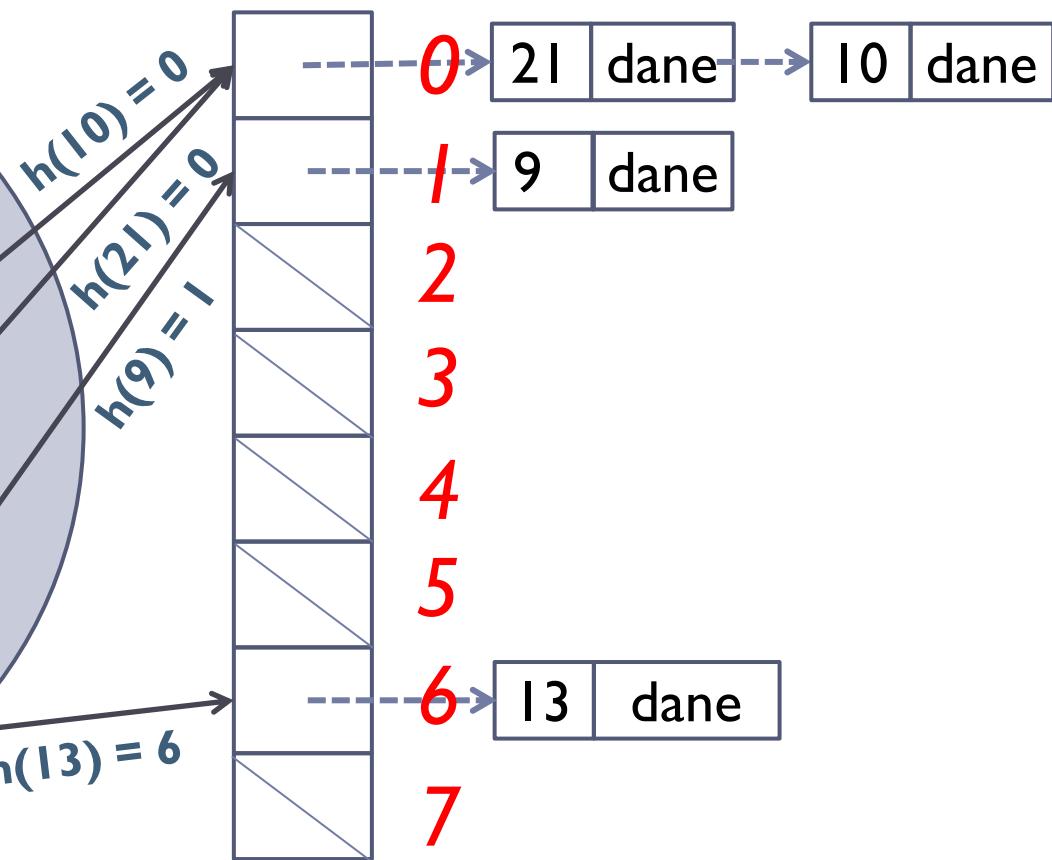


Haszowanie łańcuchowe

Przestrzeń kluczy



Tablica z haszowaniem



Klucze wykorzystane

Haszowanie łańcuchowe

Operacje na tablicy z haszowaniem:

- ▶ **Insert(T, x)**

wstawia element x na początek listy $T [h (\text{key}[x])]$

- ▶ **Search(T, k)**

wyszukuje element o kluczu k na liście $T [h (k)]$

- ▶ **Delete(T, x)**

usuwa element x z listy $T [h (\text{key}[x])]$

gdzie:

- ▶ $T []$ - tablica z haszowaniem
- ▶ $h ()$ - funkcja haszująca
- ▶ $\text{key}[x]$ - klucz, dla elementu x

▶ **Terminy kolokwium C2:**

- **17 czerwiec 2024** (poniedziałek), godz. **17-20**, CTI,
 - grupy: 2I01, 2I02, 2I05, 2I07, 2I09, 2I10
- **18 czerwiec 2024** (wtorek), godz. **16-19**, CTI,
 - grupy: 2I03, 2I04 , 2I06, 2I08, 2I11

▶ **Egzaminy z Podstaw Programowania 2 odbędą się w następujących terminach:**

- Termin 1: **26 czerwiec** (środa), godz. 8:15, sala E1
- Termin 2: **3 lipiec** (środa), godz. 8:15, sala E1
- Termin 3: **11 wrzesień** (środa), godz. 8:15, sala E1

▶ Do egzaminu można podejść max. dwukrotnie. Warunkiem podejścia jest zaliczenie laboratorium i Dante.

▶ **Poprawy kolokwiów z PP2 w sesji czerwiec/lipiec:**

- Poprawa C1: **24 czerwiec 2024** (poniedziałek), godz. **11-14** (CTI)
- Poprawa C2: **1 lipiec 2024** (poniedziałek), godzina do ustalenia

▶ **Poprawy kolokwiów z PP2 w sesji wrzesień:**

- Poprawa C1: **9 wrzesień 2024** (poniedziałek), godzina do ustalenia
- Poprawa C2: **10 wrzesień 2024** (wtorek), godzina do ustalenia

▶ Na terminy kolokwium w sesji będą obowiązywały zapisy.

Przykład 115(1)

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

#define MAX 10
// ****
struct abonent{
    char imie[10];
    char nazwisko[50];
    char telefon[15];
};

// ****
struct node{
    struct node *next;
    struct abonent dane;
};

// ****
struct list{
    struct node* head;
};

// ****
struct hashTable{
    struct list* table[MAX]; // struktura tablicy haszującej (10 - elementowej)
};
```

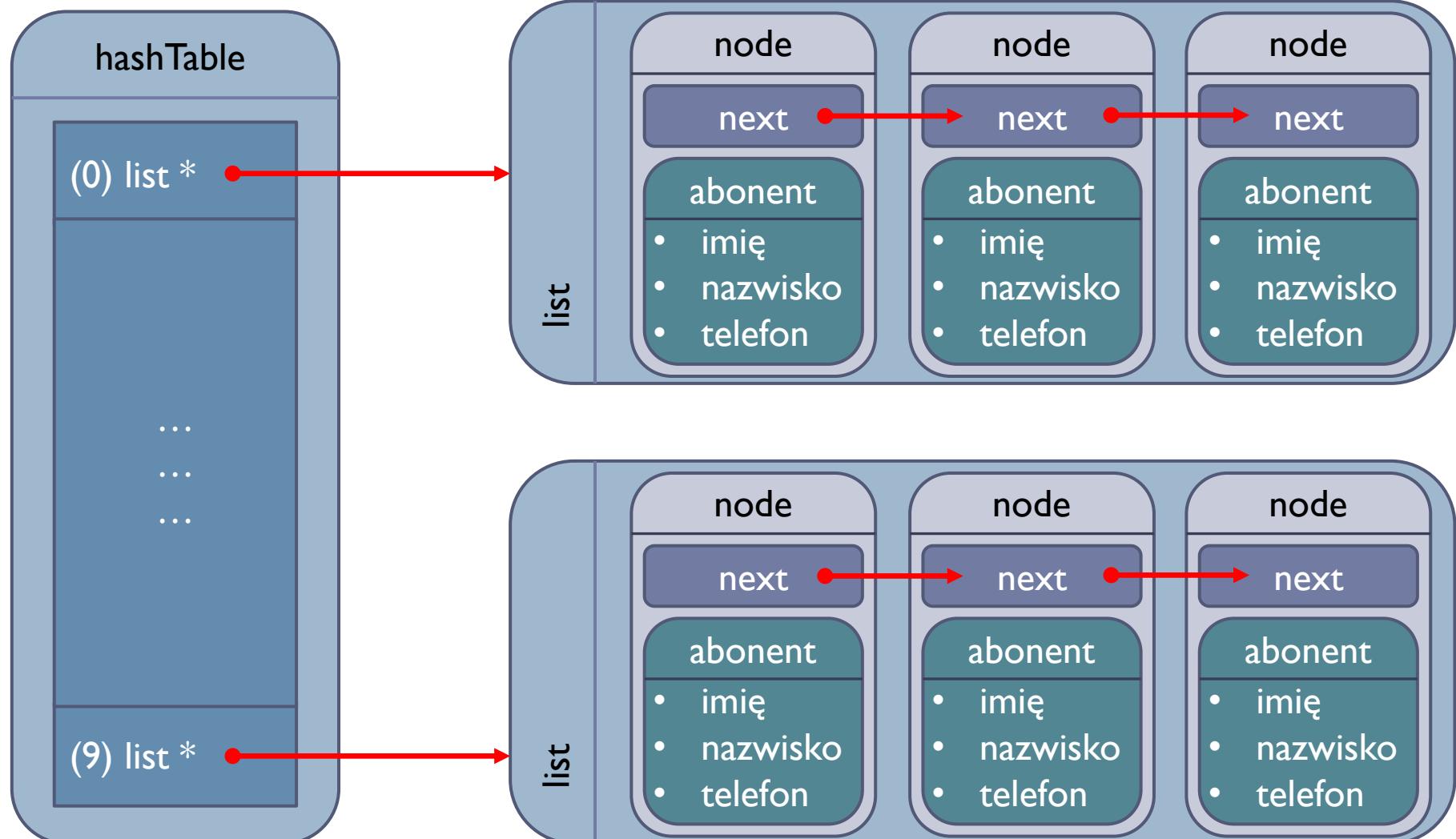
struktura przechowująca dane abonenta

struktura elementu listy

struktura listy

struktura tablicy haszującej (10 - elementowej)

Relacje struktur z przykładu 115



Przykład 115 (2)

```
*****  
/** Funkcje obsługi listy */  
*****  
  
void init(struct list *l); // jak w przykładzie 111  
void print(struct list *l);  
  
struct node* pop_back(struct list *l); // jak w przykładzie 111  
struct node* pop_front(struct list *l); // jak w przykładzie 111  
struct node* del(struct list *l, int n); // jak w przykładzie 111  
struct node* find(struct list* l, char* nazwisko) ;  
struct node* createElement(char* imie, char* nazwisko, char* telefon);  
  
void push_back(struct list *l, struct node* el); // jak w przykładzie 111  
void push_front(struct list *l, struct node* el); // jak w przykładzie 111  
void insert(struct list *l, int n, struct node* el); // jak w przykładzie 111
```

Przykład 115(3)

```
*****  
/** Funkcje obsługi tablicy haszującej */  
*****
```

```
/*1*/ int h(char*) ;  
  
/*2*/ void hTab_init(struct hashTable *hTab) ;  
/*3*/ void hTab_insert(struct hashTable* hTab, struct node *x) ;  
/*4*/ void hTab_print(struct hashTable *hTab) ;  
/*5*/ void hTab_delete(struct hashTable* hTab, struct node *x) ;  
/*6*/ void hTab_printKey(struct hashTable* hTab, char* key) ;  
  
/*7*/ struct node* hTab_search(struct hashTable* hTab, char* key) ;  
/*8*/ void hTab_free(struct hashTable *hTab) ;
```

```
/*  
1 - funkcja haszująca  
2 - inicjalizuje tablicę  
3 - dodaje element do tablicy  
4 - drukuje zawartość tablicy  
5 - usuwa element  
6 - drukuje elementy o podanym kluczu  
7 - szuka elementu o podanym kluczu, zwraca wskaźnik do pierwszego  
wystąpienia, lub NULL w przypadku, gdy element nie istnieje  
8 - czyści listy, zwalnia pamięć
```

Przykład 115₍₄₎

```
*****  
/* Funkcja haszująca */  
/* przyjmuje: klucz (nazwisko) */  
/* zwraca: indeks komórki w tablicy */  
*****  
  
int h(char* key) {  
  
    int len = strlen(key);  
    int i, sum = 0;  
  
    for(i=0; i<strlen(key); i++) {  
        sum += *(key+i);  
    }  
  
    return sum % MAX;  
}
```

suma kodów ASCII
znaków w kluczu

Przykład 115(5)

```
struct node* find(struct list* l, char* nazwisko) {  
    struct node* wsk = l->head;  
  
    while(wsk) {  
  
        if(!strcmp(wsk->dane.nazwisko, nazwisko)) {  
            return wsk;  
        }  
        wsk = wsk->next;  
    }  
    return NULL;  
}  
/**********************************************************/  
struct node* createElement(char* imie, char* nazwisko, char* telefon){  
  
    struct node* el = (struct node*) malloc (sizeof(struct node));  
    strcpy(el->dane.imie, imie);  
    strcpy(el->dane.nazwisko, nazwisko);  
    strcpy(el->dane.telefon, telefon);  
    el->next = NULL;  
  
    return el;  
}
```

Przykład 115(6)

```
void hTab_init(struct hashTable *hTab) {  
  
    int i;  
  
    for(i=0; i<MAX; i++) {      Alokacja pamięci dla każdej z list w tablicy mieszającej  
        hTab->table[i] = (struct list*)malloc(sizeof(struct list));  
        init((hTab->table[i]));  
    }  
    Inicjalizacja każdej z list (ustawienie głowy na NULL)  
}  
  
/*********************************************  
  
void hTab_insert(struct hashTable* hTab, struct node *x) {  
  
    int ind = h(x->dane.nazwisko);  
    push_front(hTab->table[ind], x);  
}  
Ustalenie indeksu tablicy  
Dodanie elementu do listy będącej w  
komórce o wyznaczonym indeksie
```

Przykład 115(7)

```
void hTab_print(struct hashTable *hTab) {  
  
    int i;  
    for(i=0; i<MAX; i++) {  
        printf("%d:\n", i);  
        print((hTab->table[i]));  
    }  
}  
/**********************************************************/  
void hTab_free(struct hashTable *hTab) {  
  
    int i;  
  
    for(i=0; i<MAX; i++) {  
  
        clear((hTab->table[i]));  
        hTab->table[i] = NULL;  
    }  
}
```

Drukowanie list z kolejnych komórek tablicy mieszającej

Przykład 115(8)

```
void hTab_printKey(struct hashTable* hTab, char* key) {  
    int i, ind = h(key);  
    struct node* el = NULL;  
  
    if(!hTab->table[ind])  
        return;  
    else{  
        el = find(hTab->table[ind],key);  
  
        if(!el)  
            return;  
        else{  
            struct node* el2 = el;  
  
            printf("\n\nElementy o kluczu \"%s\":\n", key);  
  
            while(el2){  
                if(!strcmp(el2->dane.nazwisko, key))  
                    printf("%s\t%s\t%s\n",  
                           el2->dane.nazwisko, el2->dane.imie, el2->dane.telefon);  
                el2 = el2->next;  
            }  
        }  
    }  
    return;  
}
```

Przykład 115(9)

```
void hTab_delete(struct hashTable* hTab, struct node *x) {  
    int ind = h(x->dane.nazwisko);  
    int pos = 0, flag = 0;  
  
    struct node* wsk = hTab->table[ind]->head;  
  
    if(wsk){  
        while(wsk){  
            pos++;  
            int f1=0, f2=0, f3=0;  
  
            if (!strcmp(wsk->dane.imie, x->dane.imie)) f1 = 1;  
            if (!strcmp(wsk->dane.nazwisko, x->dane.nazwisko)) f2 = 1;  
            if (!strcmp(wsk->dane.telefon, x->dane.telefon)) f3 = 1;  
  
            if(f1 && f2 && f3) {  
                flag = 1;  
                break;  
            }  
            wsk = wsk->next;  
        };  
  
        if (flag){  
            wsk = del(hTab->table[ind],pos);  
            printf("\nUsunięto: %s %s %s\n\n",  
                  wsk->dane.nazwisko, wsk->dane.imie, wsk->dane.telefon);  
        }  
    }  
}
```

Przykład 115(10)

```
int main(){
    int i;
    struct hashTable hTab;

    hTab_init(&hTab);

    struct node* a1 = createElement("Jan", "Kowalski");
    struct node* a2 = createElement("Julian", "Kowalski");
    struct node* a3 = createElement("Aldona", "Dabrowska");
    struct node* a4 = createElement("Joanna", "Iksinska");
    struct node* a5 = createElement("Michał", "Iksinski");

    hTab_insert(&hTab, a1);
    hTab_insert(&hTab, a2);
    hTab_insert(&hTab, a3);
    hTab_insert(&hTab, a4);
    hTab_insert(&hTab, a5);

    hTab_print(&hTab);
    hTab_search(&hTab, "Iksinski");
    hTab_printKey(&hTab, "Kowalski");

    hTab_delete(&hTab, a4);
    hTab_print(&hTab);
    hTab_free(&hTab);
    free(a1); free(a2); free(a3); free(a4); free(a5);
    return 0;
}
```

0:
Lista pusta

1:
Dabrowska Aldona 42 631-27-50

2:
Iksinski Michał 18 245-52-19
Kowalski Julian 42 631-27-52
Kowalski Jan 42 631-27-51

3:
Lista pusta

4:
Iksinska Joanna 18 245-52-19
Element o kluczu "Iksinski" znaleziono
Elementy o kluczu "Kowalski":
Kowalski Julian 42 631-27-52
Kowalski Jan 42 631-27-51
Usunięto: Iksinska Joanna 18 245-52-19

0:
Lista pusta

1:
Dabrowska Aldona 42 631-27-50

2:
Iksinski Michał 18 245-52-19
Kowalski Julian 42 631-27-52
Kowalski Jan 42 631-27-51

3:
Lista pusta

4:
Lista pusta

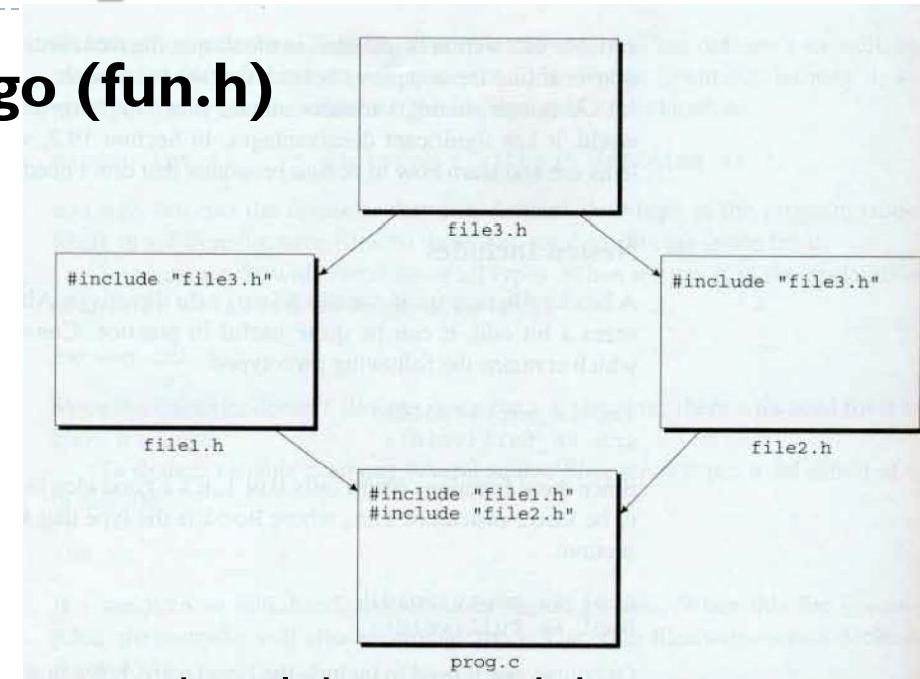
Podział programu na pliki

- ▶ Program napisany w języku C może zawierać dwa zasadnicze rodzaje plików:
 - ▶ **Pliki nagłówkowe** (ang. *header files*)
 - ▶ z rozszerzeniem *.h
 - ▶ zawierają nagłówki funkcji oraz definicje własnych typów danych
 - ▶ **Pliki źródłowe** (ang. *source files*)
 - ▶ z rozszerzeniem *.c
 - ▶ zawierają definicje funkcji oraz funkcję `main()`

Podział programu na pliki

Struktura pliku nagłówkowego (fun.h)

```
#ifndef fun_h
#define fun_h
/*
  definicje własnych typów danych
  deklaracje (nagłówki) funkcji
*/
#endif
```



- ▶ Dyrektywy preprocessora zabezpieczają przed wielokrotnym dołączaniem tego samego kodu do programu
- ▶ Bez ww. dyrektyw co najmniej dwukrotne dołączenie w programie tego samego pliku nagłówkowego (nawet w różnych plikach) spowoduje błąd komilacji.
- ▶ Nazwy zmiennych, definiowanych za pomocą preprocessora muszą być unikatowe

Dyrektywa #error w pliku nagłówkowym

- ▶ Informacja o sytuacji, w której plik nagłówkowy nie może być podłączony

```
#ifndef __STDC__
#error This header requires a Standard C compiler
#endif
```

Podział programu na pliki

Struktura pliku źródłowego (fun.c)

```
#include "fun.h"  
/*  
tutaj definicje funkcji  
*/
```

W przypadku programów wieloplikowych, **tylko jeden** z plików źródłowych może zawierać funkcję **main ()**

Funkcja **main ()** musi wystąpić w programie, inaczej kompilator zgłosi następujący błąd:

C:\Users\Ania\Desktop\Makefile.wir

[Linker error] undefined reference to 'WinMain@16'
ld returned 1 exit status
[Build Error] [Project1.exe] Error 1

Dyrektywa #include

- ▶ `#include "header.h"`
 - ▶ dla własnych plików nagłówkowych
- ▶ `#include <header.h>`
 - ▶ dla plików nagłówkowych biblioteki języka
- ▶ „warunkowa” dyrektywa `#include`

```
#if defined(IA32)
    #define CPU_FILE "ia32.h"
#elif defined(IA64)
    #define CPU_FILE "ia64.h"
#elif defined (AMD64)
    #define CPU_FILE "amd64.h"
#endif

#include CPU_FILE
```

Podział programu na pliki

The screenshot shows the Dev-C++ 4.9.9.2 IDE interface. The title bar reads "Dev-C++ 4.9.9.2 - [Project1] - Project1.dev". The menu bar includes File, Edit, Search, View, Project, Execute, Debug, Tools, CVS, Window, and Help. The toolbar contains various icons for file operations like Open, Save, Print, and search. Below the toolbar is a palette with icons for New, Insert, Toggle, and Goto. The main window has tabs for Project, Classes, and Debug, with example.c currently selected. The Project view shows a tree with "Project1" containing "example.c", "fun.c", and "fun.h". The code editor displays the following C code:

```
#include "fun.h"

#include <stdio.h>
#include <stdlib.h>
#include <time.h>

/*****************
int main(){

    struct hashTable hTab;

    hTab_init(&hTab);

    int i;
```

example.c – główny plik źródłowy programu – zawiera funkcję `main()`

fun.h – plik nagłówkowy – zawiera deklaracje funkcji oraz definicje typów danych

fun.c – plik źródłowy, zawiera definicje funkcji zadeklarowanych w **fun.h**

Plik fun.h

```
#ifndef FUN_H
#define FUN_H

#define MAX 5
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
/*************************************************************************************************/
struct abonent{
    char imie[10];
    char nazwisko[50];
    char telefon[15];};
/*************************************************************************************************/
.....  
/*************************************************************************************************/
struct hashTable{
    struct list* table[MAX];};
/*************************************************************************************************/
void init(struct list *l);
void print(struct list *l);
struct node* createElement(char* imie, char* nazwisko, char* telefon);
struct node* del(struct list *l, int n);
.....  
void hTab_printKey(struct hashTable* hTab, char* key);
void hTab_delete(struct hashTable* hTab, struct node *x);
struct node* hTab_search(struct hashTable* hTab, char* key);

#endif
```

Plik fun.c

```
#include "fun.h"

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
/*************************************************************************************************/
int h(char* key){

    int len = strlen(key);
    int i, sum = 0;

    for(i=0; i<strlen(key); i++){
        sum += *(key+i);
    }
    return sum % MAX;
}
/*************************************************************************************************/
void hTab_insert(struct hashTable* hTab, struct node *x){

    int ind = h(x->dane.nazwisko);
    push_front(hTab->table[ind],x);
}
/*************************************************************************************************/
/* dalsze definicje wszystkich funkcji wskazanych w fun.h */
```

Plik example.c

```
#include "fun.h"
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main(){

    struct hashTable hTab;

    hTab_init(&hTab);

    int i;

    struct node* a1 = createElement("Jan", "Kowalski", "42 631-27-51");
    struct node* a2 = createElement("Julian", "Kowalski", "42 631-27-52");
    struct node* a3 = createElement("Aldona", "Dabrowska", "42 631-27-50");
    struct node* a4 = createElement("Joanna", "Iksinska", "18 245-52-19");
    struct node* a5 = createElement("Michal", "Iksinski", "18 245-52-19");

    hTab_insert(&hTab, a1);
    hTab_insert(&hTab, a2);
    hTab_insert(&hTab, a3);
    hTab_insert(&hTab, a4);
    hTab_insert(&hTab, a5);

    hTab_print(&hTab);
    hTab_search(&hTab, "Iksinski");
    hTab_printKey(&hTab, "Kowalski");

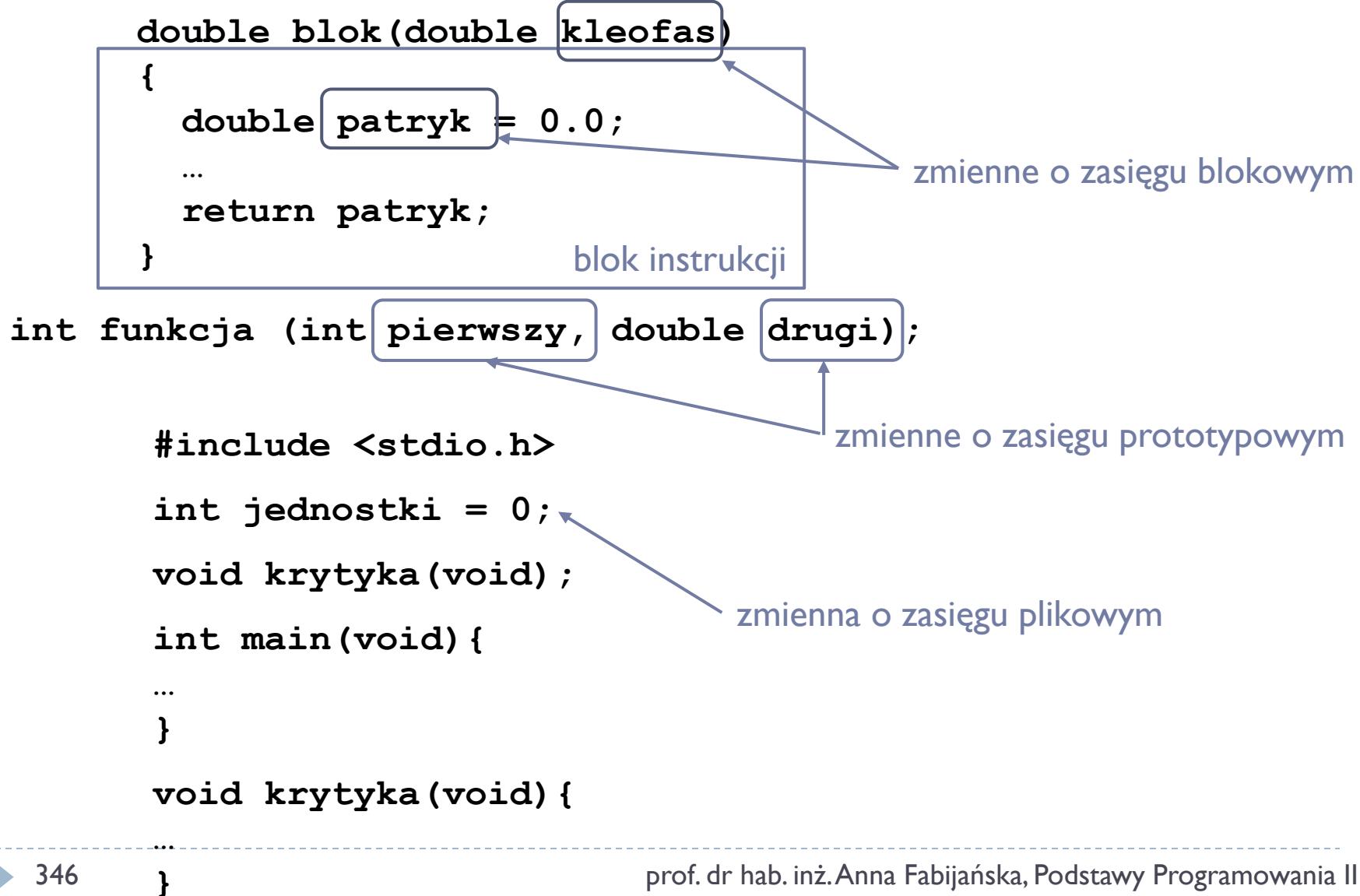
    hTab_delete(&hTab, a4);
    hTab_print(&hTab);

    return 0;
}
```

Zasięg zmiennych

- ▶ Zasięg zmiennej określa region lub regiony programu, które mają dostęp do identyfikatora zmiennej
- ▶ Zmienna w języku C może posiadać jeden z następujących zasięgów:
 - ▶ **blokowy** – zmienne zadeklarowane wewnątrz bloku (...) mają zasięg ograniczony jedynie do tego bloku
 - ▶ **prototypowy** – dotyczy zmiennych użytych w prototypach funkcji, obejmuje obszar od miejsca gdzie zdefiniowano zmienną do końca prototypu funkcji;
 - ▶ **plikowy** – dotyczy zmiennych, których definicja znajduje się poza blokami funkcji, takie zmienne są dostępne w całym pliku, począwszy od miejsca ich definicji

Zasięg zmiennych



Łączność zmiennej

- ▶ Zmienna w języku C może charakteryzować się jednym z następujących typów łączności:
 - ▶ **łączność zewnętrzna** (ang. *external linkage*)
 - ▶ zmienne tego typu mogą być użyte w dowolnym miejscu programu złożonego z wielu plików
 - ▶ zmienna zewnętrzna zdefiniowana w jednym pliku nie jest dostępna w innych plikach, jeżeli nie zostanie w nich zadeklarowana (z pomocą słowa kluczowego **extern**)
 - ▶ **łączność wewnętrzna** (ang. *internal linkage*)
 - ▶ zmienne tego typu mogą być użyte w dowolnym miejscu, ale tylko w obszarze jednego pliku
 - ▶ **brak łączności**
 - ▶ zmienne o zasięgu blokowym i prototypowym (są one prywatne w obrębie bloku lub prototypu, w którym zostały zdefiniowane)

Modyfikatory sposobu przechowywania obiektów

- ▶ Język C definiuje cztery modyfikatory typów wpływające na sposób przechowywania zmiennych:
 - ▶ **auto**
 - ▶ **extern**
 - ▶ **register**
 - ▶ **static**
- ▶ Modyfikatory powinny poprzedzać nazwę typu:

```
extern float licznik;
```

```
static int ile = 0;
```

```
register int j;
```

Modyfikator **extern**

- ▶ Język C dopuszcza rozbijanie programu na kilka plików
- ▶ Pliki te można oddziennie kompilować a następnie scalać
- ▶ W programach „wieloplikowych” dane globalne mogą być definiowane tylko raz - mogą być jednak wykorzystywane w wielu plikach tworzących program
 - ▶ w takim przypadku każdy plik programu musi poinformować kompilator o wykorzystywanych danych globalnych słowem kluczowym **extern**

Modyfikator extern

```
#include <stdio.h>

int licznik;

void f1(void);

int main(void) {

    int i
    f1();
    for(i=0; i<licznik; i++)
        printf("%d ",i);

    return 0;
}
```

PLIK 1

```
#include <stdlib.h>

void f1(void) {
    licznik = rand();
}
```

PLIK 2

Błąd przy próbie komplikacji
- niezadeklarowana zmienna licznik

Modyfikator extern

```
#include <stdio.h>

int licznik;

void f1(void);

int main(void) {

    int i

    f1();

    for(i=0; i<licznik; i++)
        printf("%d ",i);

    return 0;
}
```

PLIK I

```
#include <stdlib.h>

int licznik;

void f1(void) {
    licznik = rand();
}
```

PLIK 2

Błąd przy próbie komplikacji
- redeklaracja zmiennej licznik

Modyfikator extern

```
#include <stdio.h>

int licznik;

void f1(void);

int main(void) {

    int i

    f1();

    for(i=0; i<licznik; i++)
        printf("%d ",i);

    return 0;
}
```

PLIK I

```
#include <stdlib.h>
```

```
extern int licznik;
```

```
void f1(void) {
    licznik = rand();
}
```

PLIK 2

Poinformowanie kompilatora
o istnieniu zmiennej danego typu
bez alokacji miejsca na jej przechowanie

Modyfikator extern

deklaracja

```
#include <stdio.h>

char nap[80];
void dajimie(void);

int main(void)
{
    dajimie();
    printf("Witaj %s", nap);

    return 0;
}
```

PLIK 1

definicja

```
#include <stdlib.h>

extern char nap[80];

void dajimie(void)
{
    printf("Wprowadz imię: ");
    fgets(nap, 79, stdin);
}
```

PLIK 2

Ta sama zmienna!

Modyfikator register

- ▶ Modyfikator **register** określa, że potrzebny jest szybszy dostęp do określonej zmiennej
- ▶ We wczesnych wersjach języka mógł się odnosić do zmiennych lokalnych typu **int** lub **char**, które mogły być przechowywane w rejestrach procesora
- ▶ We współczesnej wersji języka definicja **register** została rozszerzona na wszystkie typy zmiennych, a założenie przechowywania w rejestrach – uchylone
- ▶ Tylko ograniczoną liczbę zmiennych można uczynić zmiennymi typu **register**
- ▶ Zmienne typu register mogą nie mieć adresu

Przykład 116

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

int main(void) {
    int i;
    register int j;

    clock_t początek, koniec;

    początek = clock();
    for(i=0; i<5000000000; i++);
    koniec = clock();

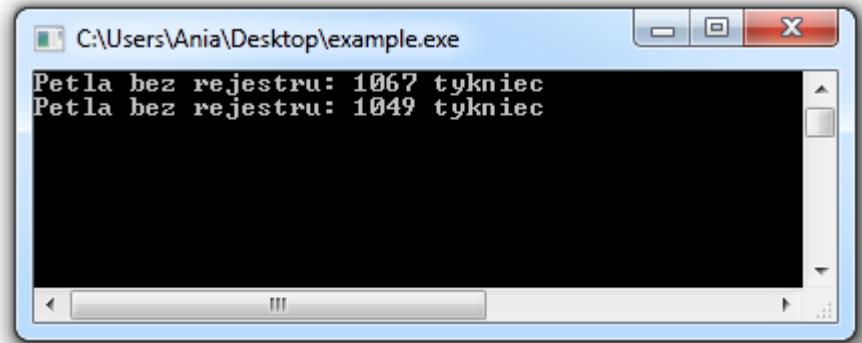
    printf("Petla bez rejestrów: %ld tykniec\n", koniec-początek);

    początek = clock();
    for(j=0; j<5000000000; j++);
    koniec = clock();

    printf("Petla bez rejestrów: %ld tykniec\n", koniec-początek);

    return 0;
}
```

pomiar czasu działania pętli



Modyfikator static

- ▶ Modyfikator **static** pozwala na zachowanie wartości zmiennej lokalnej pomiędzy wywołaniami funkcji
- ▶ W przeciwieństwie do zwykłych zmiennych lokalnych inicjalizowanych przy każdym wywołaniu funkcji, zmienne **static** są inicjalizowane tylko raz.
- ▶ Funkcja może zwrócić wskaźnik do zmiennej statycznej
- ▶ Użyty do zmiennej poza blokiem instrukcji (zasięg globalny) ogranicza łączność zmiennej (ukrywa jej widoczność do bieżącego pliku).

Przykład 117(1)

```
#include <stdio.h>

void f(void);

int main(void) {

    int i;

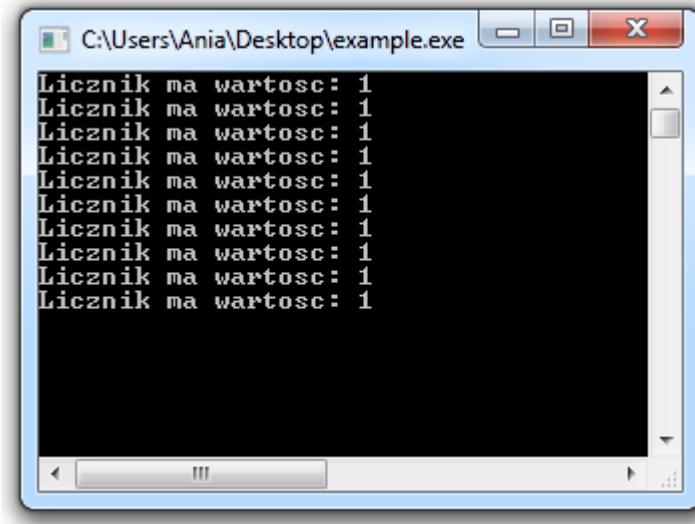
    for(i=0; i<10; i++) f();

    return 0;
}

void f(void) {

    int licznik = 0;

    licznik++;
    printf("Licznik ma wartosc: %d\n", licznik);
}
```



Przykład 117₍₂₎

```
#include <stdio.h>

void f(void);

int main(void) {

    int i;

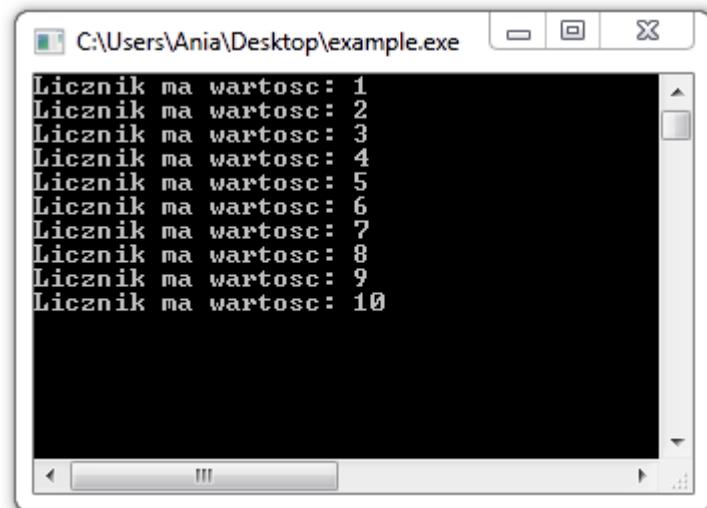
    for(i=0; i<10; i++) f();

    return 0;
}

void f(void) {

    static int licznik = 0;

    licznik++;
    printf("Licznik ma wartosc: %d\n", licznik);
}
```



Modyfikator static

- ▶ **Zastosowanie:** kontrola liczby wykonania danej czynności

```
void mojafunkcja(void) {  
  
    static int pierwszy = 1;  
  
    if (pierwszy) { /*inicjalizacja systemu*/  
  
        a=0;  
        poz = 0;  
        printf("System zainicjalizowany");  
        pierwszy = 0;  
    }  
    .  
    .  
}
```

Modyfikator static

- ▶ **Zastosowanie:** kontrolowanie funkcji rekurencyjnej

```
#include <stdio.h>

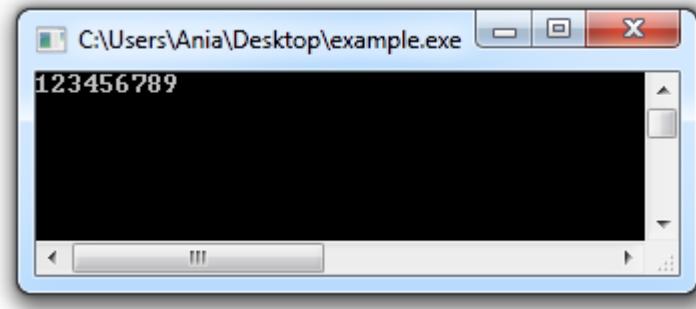
void f(void);

int main(void) {
    f();
    return 0;
}

void f(void) {
    static int koniec = 0;
    koniec++;

    if (koniec == 10) return;

    printf("%d", koniec);
    f(); /*wywolanie rekurencyjne*/
}
```



Efektywność wykonania programu

```
char digit_to_hex_char(int digit)
{
    // kopiowanie danych do tablicy przy każdym wywołaniu funkcji
    const char hex_chars[16] = "0123456789ABCDEF";
    return hex_chars[digit];
}
```

vs.

```
char digit_to_hex_char(int digit)
{
    // kopiowanie danych do tablicy przy pierwszym wywołaniu funkcji
    static const char hex_chars[16] = "0123456789ABCDEF";
    return hex_chars[digit];
}
```

Modyfikator static

- ▶ Modyfikator **static** może być używany także ze zmiennymi globalnymi
- ▶ Tak określona zmienna globalna staje się widoczna jedynie dla funkcji zawartych w tym samym pliku, w którym została zadeklarowana -> funkcje z innych plików nie mają do niej dostępu -> pozwala to uniknąć konfliktów nazw

```
int licznik;  
.  
. .  
. .  
licznik = 10;  
printf("%d",licznik);  
  
// wydrukowane: 10
```

PLIK 1

```
static int licznik;  
.  
. .  
. .  
licznik = 5;  
printf("%d",licznik);  
  
// wydrukowane: 5
```

PLIK 2

Modyfikatory dostępu

- ▶ Język C ma dwa rodzaje modyfikatorów wpływających na sposób dostępu do zmiennych – zarówno przez sam program, jak i kompilator:
 - ▶ **const** (typ stały)
 - ▶ zapobiega modyfikacji zmiennej przez program
 - ▶ uniemożliwia funkcji modyfikację obiektu wskazywanego przez parametr
 - ▶ **volatile** (typ ulotny)
 - ▶ sygnalizuje kompilatorowi, że wartość zmiennej może się zmieniać z przyczyn niezależnych od programu
 - ▶ wymusza konieczność sprawdzania wartości zmiennej przy każdym odwołaniu
 - ▶ **restrict** (tylko dla wskaźników)

Modyfikator **const**

- ▶ Zmiennej **const** należy nadać wartość początkową na etapie definicji
- ▶ Wartość ta nie może być zmieniana w programie

```
#include <stdio.h> {

    const int i = 10;
    printf("%d", i);

    i = 20;      //BŁĄD

    printf("%d", i);

    return 0;
}
```

Przykład 118 – plik czesca.c

```
#include <stdio.h>

void podaj_liczbe();
void sumuj(int k);

int liczba = 0;           //zasięg plikowy, łączność wewnętrzna

int main(void) {
    int wartosc;          //zmienna automatyczna
    register int i;        //zmienna rejestrowa

    printf("Podaj dodatnia liczbe całkowita (0 to koniec): ");
    while(scanf("%d", &wartosc) == 1 && wartosc > 0) {
        ++ liczba;         //zmienna o zasięgu plikowym

        for(i=wartosc; i>=0; i--)
            sumuj(i);
        printf("Podaj dodatnia liczbe całkowita (0 to koniec): ");
    }
    podaj_liczbe();
    return 0;
}

void podaj_liczbe() {
    printf("Petle opuszczono po %d cyklach\n", liczba);
}
```

Przykład 118 – plik czescb.c

```
#include <stdio.h>

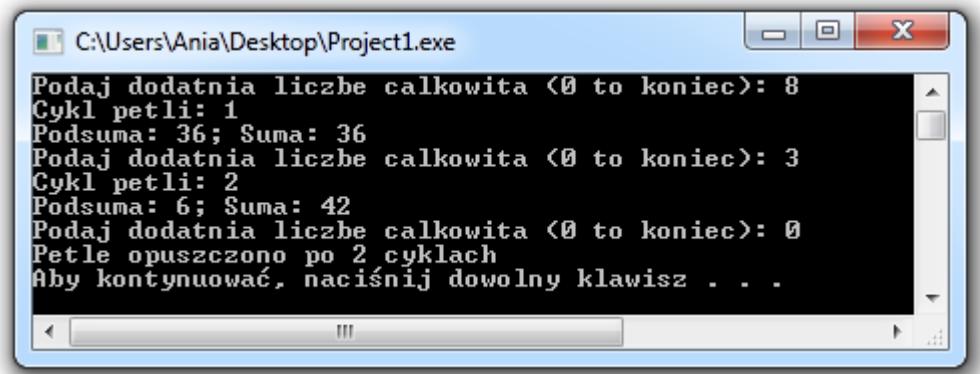
extern int liczba;      //deklaracja nawiązująca, łączność zewnętrzna
static int suma = 0;    //definicja static, łączność wewnętrzna

void sumuj(int k);     //k ma zasięg prototypowy i brak łączności

void sumuj(int k){     // k ma zasięg blokowy i brak łączności

    static int podsuma = 0; //statyczna, brak łączności

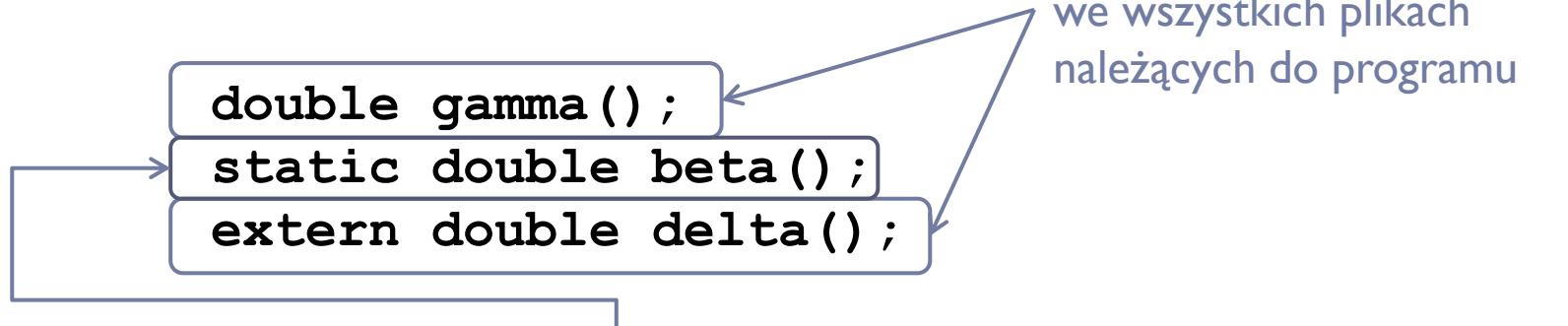
    if( k <= 0 ){
        printf("Cykl petli: %d\n", liczba);
        printf("Podsuma: %d; Suma: %d\n", podsuma, suma);
        podsuma = 0;
    }
    else{
        podsuma += k;
        suma += k;
    }
}
```



Programy wieloplikowe a funkcje

▶ Funkcje w języku C mogą być:

- ▶ **zewnętrzne** (domyślnie)
 - ▶ dostępne w innych plikach
- ▶ **statyczne**
 - ▶ dostępne jedynie w pliku, w którym zostały zdefiniowane

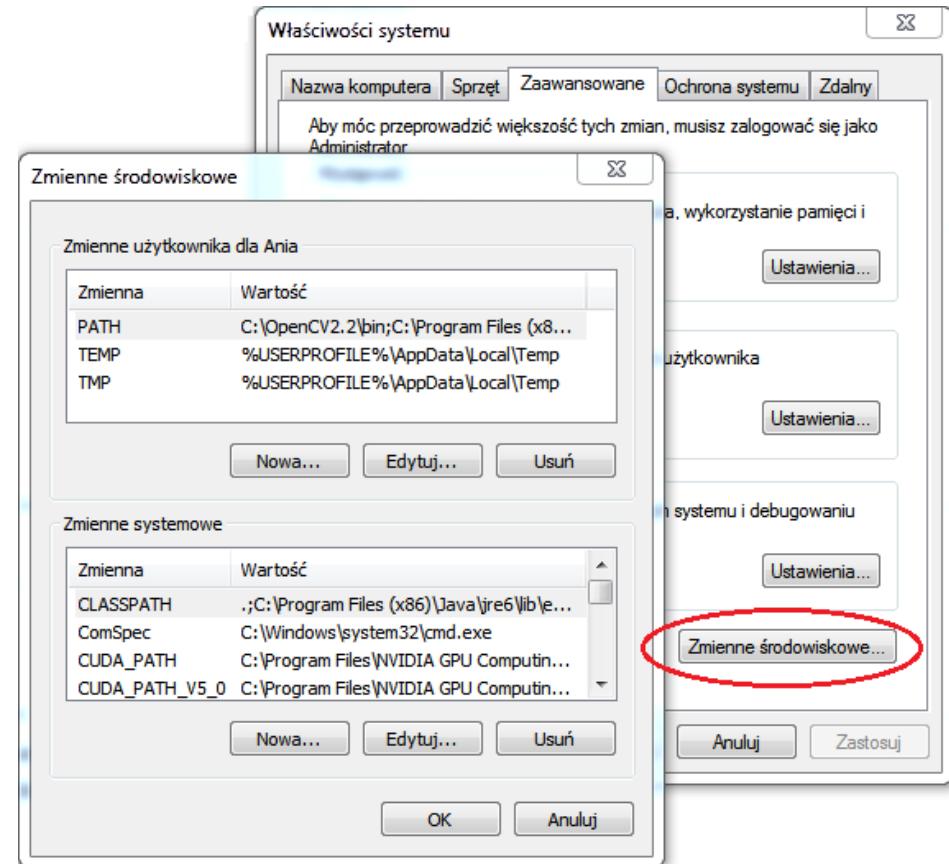


Zalety programów wieloplikowych

- ▶ Zalety programów wieloplikowych:
 - ▶ przyspieszenie czasu komplikacji
 - nie ma potrzeby każdorazowego kompilowania całego programu, a jedynie pliki, które uległy zmianie;
 - ▶ zwiększenie czytelności projektu
 - poprzez podział kodu programu na „tematyczne” fragmenty o wspólnej funkcjonalności (np. funkcje do drukowania, obliczeń, itp.);
 - ▶ ponowne używanie modułów
 - „wydzielony” kod może być wielokrotnie wykorzystany w przyszłych projektach

Zmienne środowiskowe

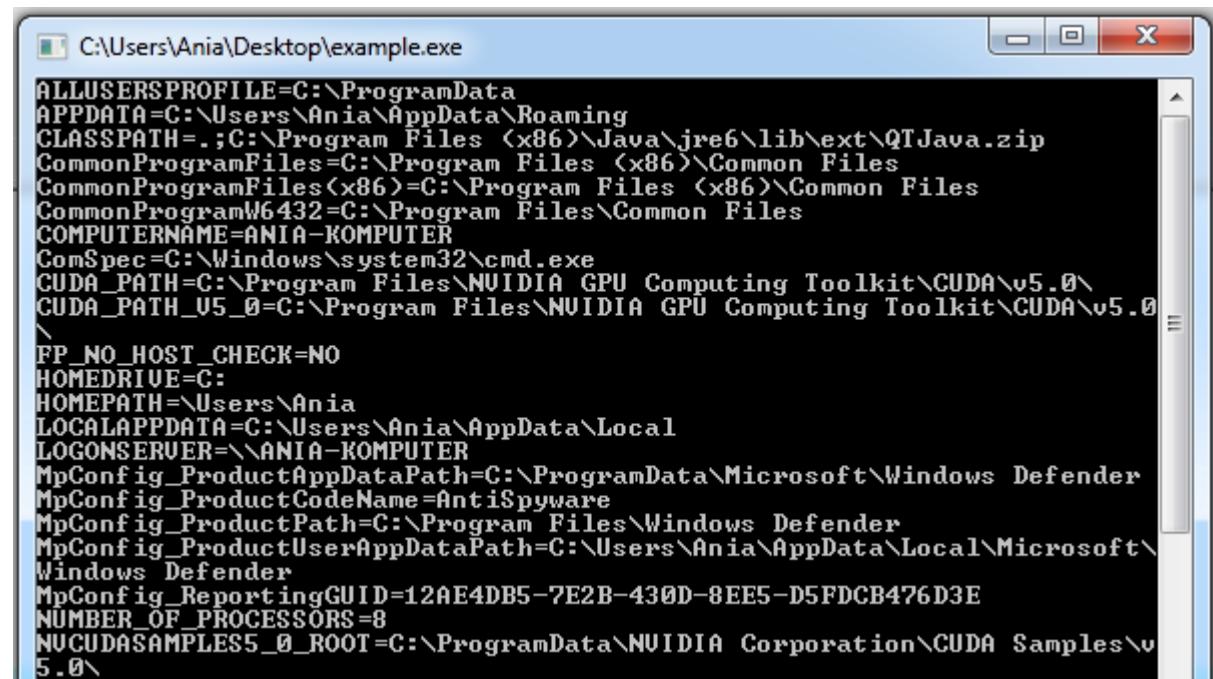
- ▶ Zmienne środowiskowe
(ang. *environment variables*)
 - ▶ zbiór dynamicznych wartości, wpływających na sposób działania uruchomionych programów (procesów)



Przykład 119

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

main(int argc, char *argv[], char *env[]) {
    while(*env) {
        printf("%s\n", *env++);
    }
    return 0;
}
```



Zmienne środowiskowe

Obsługę zmiennych środowiskowych zapewniają funkcje z pliku nagłówkowego **<stdlib.h>**

```
char* getenv(const char* varname);
```

- ▶ zwraca wskaźnik do łańcucha reprezentującego wartość zmiennej środowiskowej wskazywanej przez `varname` lub `NULL` w przypadku niepowodzenia

```
int setenv(const char *varname, const char *value,  
           int overwrite);
```

- ▶ nadaje wartość `value` zmiennej środowiskowej `varname`, jeżeli zmienna nie istnieje jest dodawana do listy zmiennych środowiskowych z daną wartością; jeżeli zmienna istnieje dla niezerowej wartości parametru `overwrite` jest nadpisywana, w przeciwnym wypadku jest pozostawiona bez zmian

Zmienne środowiskowe

```
int putenv (const char *string)
```

- ▶ dodaje nową zmienną środowiskową lub modyfikuje istniejącą, parametr *string* ma postać: *varname=value*

```
void unsetenv (const char *varname)
```

- ▶ usuwa zmienną *varname* z listy zmiennych środowiskowych

Przykład 120

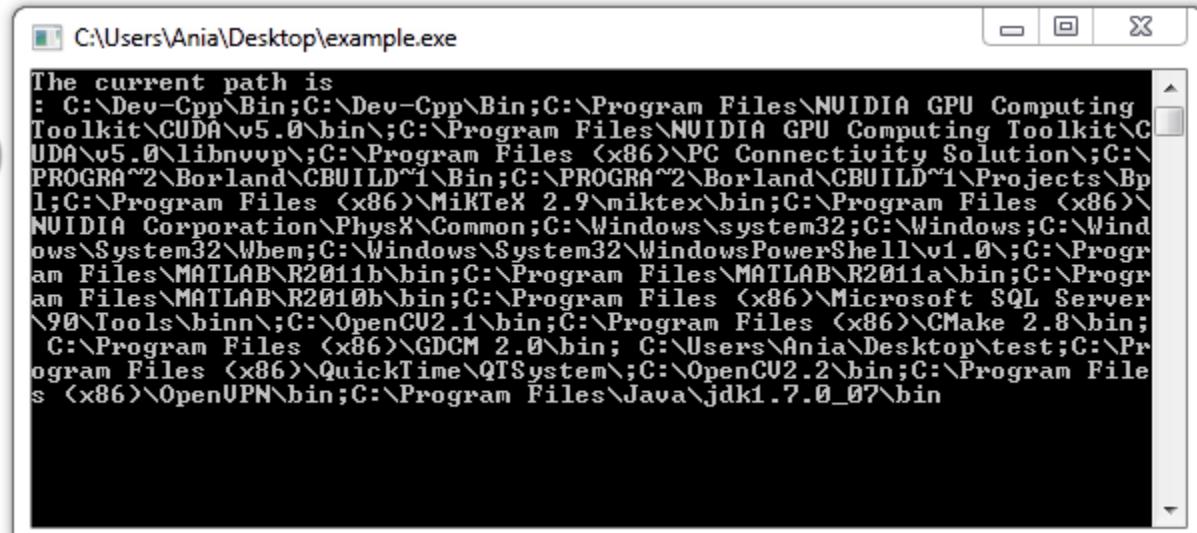
```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    char *pPath, *pRoot;

    pPath = getenv("PATH");
    if(pPath != NULL)
        printf( "The current path is\n: %s", pPath);

    pRoot = getenv("ROOT");
    if(pRoot != NULL)
        printf("The current root is\n: %s", pRoot);

    getchar();
    return 0;
}
```



Przykład 121

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

void testVariable(char* varName, char* value);

int main(){

    char *pRoot;

    pRoot = getenv("ROOT");

    testVariable("ROOT", pRoot);

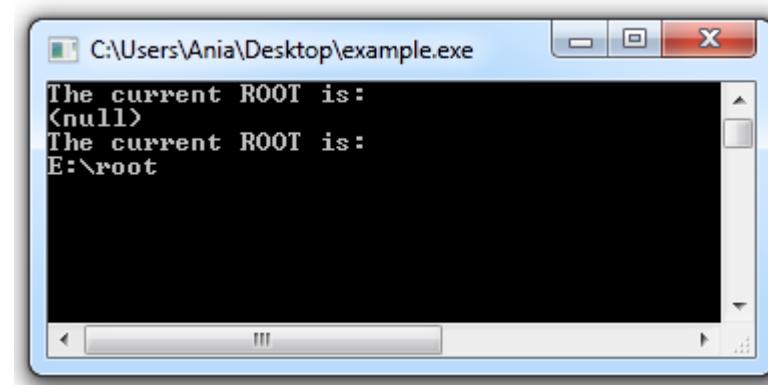
    if(putenv("ROOT=E:\\root")!= -1){

        pRoot = getenv( "ROOT" );
        testVariable("ROOT",pRoot);
    }

    return 0;
}

void testVariable(char* varName, char* value){

    if(varName != NULL)
        printf("The current %s is:\n%s\n", varName, value);
    else
        printf("Unknown variable %s\n", varName);
}
```



Rekurencja



- ▶ Język C pozwala, aby funkcja (pośrednio lub bezpośrednio) wywoływała samą siebie
- ▶ Proces ten nosi nazwę **rekurencji** (ang. *recursion*)
- ▶ Każda definicja rekurencyjna potrzebuje przynajmniej jednego przypadku bazowego (nie rekurencyjnego).
- ▶ Jeżeli funkcja rekurencyjna nie zawiera odpowiednio sformułowanego warunku stopu – wówczas wywołuje się bez końca



Przykład 122

```
#include <stdio.h>

/***********************
void gora_i_dol(int) ;
/**********************/

int main(void) {
    gora_i_dol(1);
    return 0;
}

/***********************/

void gora_i_dol(int n){
    printf("Poziom: %d: adres zmiennej n: %p\n\n", n, &n);

    if(n<4)
        gora_i_dol(n+1);

    printf("Poziom: %d: adres zmiennej n: %p\n\n", n, &n);
}
```

```
Poziom: 1: adres zmiennej n: 0028FF30
Poziom: 2: adres zmiennej n: 0028FF10
Poziom: 3: adres zmiennej n: 0028FEF0
Poziom: 4: adres zmiennej n: 0028FED0
Poziom: 4: adres zmiennej n: 0028FED0
Poziom: 3: adres zmiennej n: 0028FEF0
Poziom: 2: adres zmiennej n: 0028FF10
Poziom: 1: adres zmiennej n: 0028FF30
```

Rekurencja

Istotne sprawy dot. rekurencji:

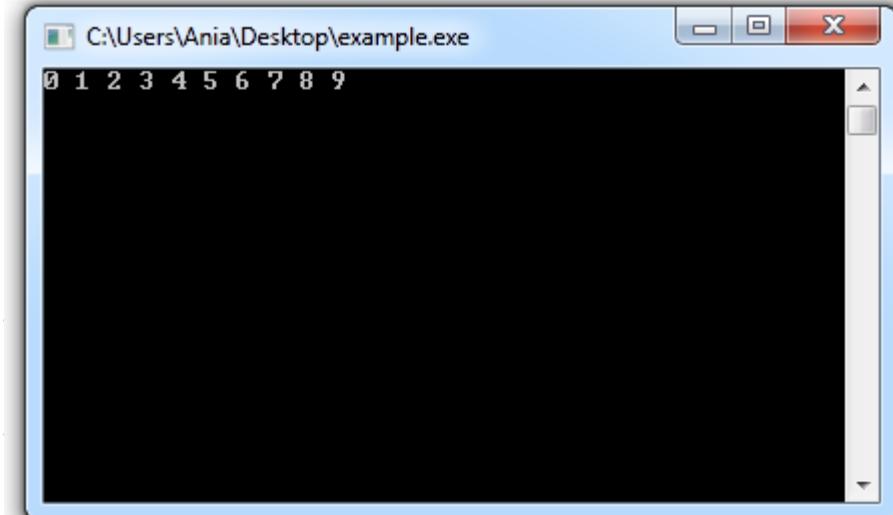
- ▶ każdy poziom funkcji rekurencyjnej posiada swoje własne zmienne
- ▶ każdemu wywołaniu funkcji odpowiada jeden powrót
- ▶ instrukcje w funkcji rekurencyjnej, znajdujące się **przed** miejscem, w którym wywołuje ona samą siebie, wykonywane są **w kolejności wykonywania poziomów**
- ▶ instrukcje w funkcji rekurencyjnej, znajdujące się **po** miejscu, w którym wywołuje ona samą siebie, wykonywane są **w kolejności odwrotnej do kolejności wykonywania poziomów**
- ▶ kod funkcji dla kolejnych wywołań nie jest zwielokrotniany
- ▶ funkcja rekurencyjna powinna posiadać element umożliwiający zakończenie sekwencji wywołań rekurencyjnych

Przykład 123a

```
#include <stdio.h>
/******************
void rekurencja(int);
/******************
int main(void) {

    rekurencja(0);
    getchar();
    return 0;
}
/******************
void rekurencja(int i) {

    if(i<10) {
        printf("%d ", i);
        rekurencja(i+1);
    }
}
```

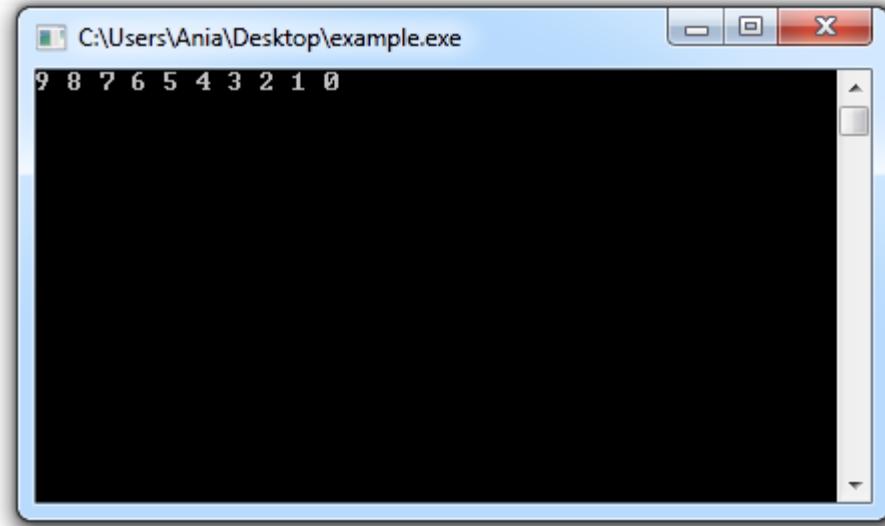


Przykład 123b

```
#include <stdio.h>
/******************
void rekurencja(int);
/******************
int main(void) {

    rekurencja(0);
    getchar();
    return 0;
}
/******************
void rekurencja(int i) {

    if(i<10){
        rekurencja(i+1);
        printf("%d ", i);
    }
}
```



Rekurencja końcowa

- ▶ W najprostszej postaci, wywołanie rekurencyjne znajduje się na końcu funkcji, tuż przed słowem *return* – mówi się wówczas o rekurencji końcowej (ang. *tail/end recursion*)
 - ▶ jest to najprostsza forma rekurencji, która działa tak, jak pętla
- ▶ Rekurencja zazwyczaj zużywa więcej pamięci, niż pętla:
 - ▶ otrzymuje oddzielny zestaw zmiennych
 - ▶ każde wywołanie umieszcza na stosie nową porcję danych
- ▶ Rekurencja jest zazwyczaj wolniejsza, ponieważ każde wywołanie funkcji zabiera czas

```

#include <stdio.h>
#include <time.h>

long long silnia (int n);
long long rsilnia (int n);

int main(void) {
    int i;
    time_t start, stop;

    start = clock();
    for (i=0; i<1000000; i++) silnia(120);
    stop = clock();

    printf("slinia iteracyjnie: %ld taktow\n", stop - start);

    start = clock();
    for (i=0; i<1000000; i++) rsilnia(120);
    stop = clock();

    printf("slinia rekurencyjnie: %ld taktow\n", stop - start);

    return 0;
}

long long silnia(int n){
    long long wyn;
    for(wyn = 1; n>1; n--)
        wyn*=n;
    return wyn;
}

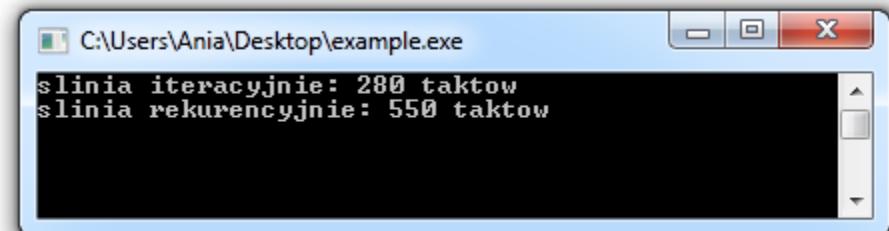
```

```

long long rsilnia(int n){
    long long wyn;
    if(n>0)
        wyn = n* rsilnia(n-1);
    else
        wyn = 1;
    return wyn;
}

```

Przykład 124



Przykład 125

```
#include <stdio.h>

/***********************/

void do_binar(int);

/***********************/

int main(void){

    int liczba;
    printf("Podaj liczbe calkowita (q konczy program) : \n");
    while(scanf("%d", &liczba) == 1) {
        printf("Odpowiednik dwojkowy: ");
        do_binar(liczba);
        printf("\n\n");
        printf("Podaj liczbe calkowita (q konczy program) : \n");
    }

    return 0;
}

/***********************/

void do_binar(int n){

    int r;
    r = n%2;

    if(n>=2)
        do_binar(n/2);
    putchar('0'+r);
    return;
}
```

```
C:\Users\Ania\Desktop\example.exe
Podaj liczbe calkowita (q konczy program):
5
Odpowiednik dwojkowy: 101
Podaj liczbe calkowita (q konczy program):
14
Odpowiednik dwojkowy: 1110
Podaj liczbe calkowita (q konczy program):
23
Odpowiednik dwojkowy: 10111
Podaj liczbe calkowita (q konczy program):
q
```

Rekurencyjne znajdowanie binarnej reprezentacji liczby dziesiętnej

Przykład 126

```
#include <stdio.h>

void rkopiuj(char *s1, char *s2);

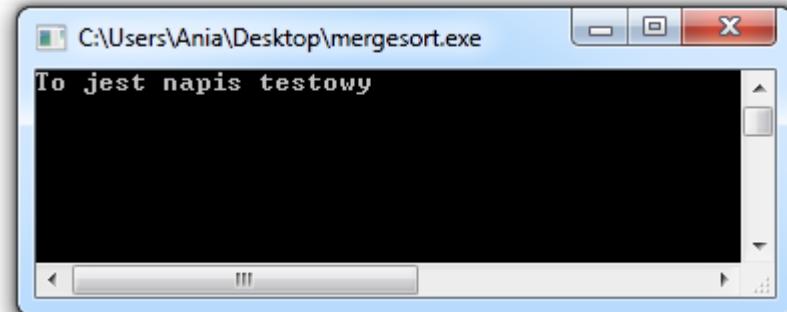
int main(void) {

    char nap[80];
    rkopiuj(nap, "To jest napis testowy");
    printf(nap);
    getchar();

    return 0;
}

void rkopiuj(char *s1, char *s2) {

    if(*s2) {                                /*jesli nie na koncu s2*/
        *s1++ = *s2++;
        rkopiuj(s1,s2);
    }
    else
        *s1 = '\0';                          /*znak null konczacy napis*/
}
```



Rekurencyjne kopiowanie napisów

Ciąg Fibonacciego

```
int FibRek(int n){  
    if (n < 2) return n;  
    else return FibRek(n-1)+FibRek(n-2);  
}
```

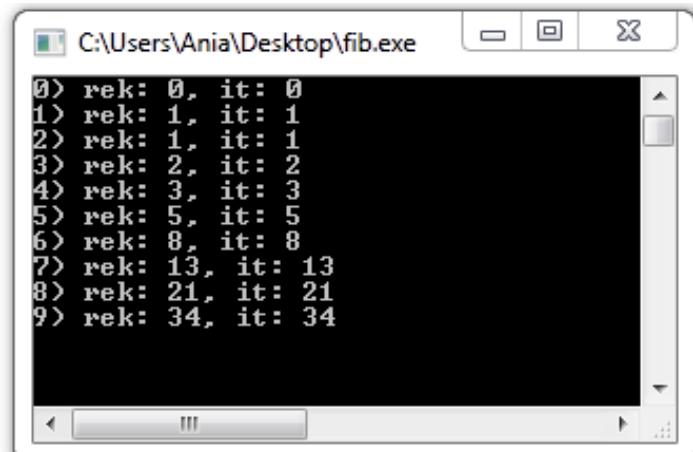
REKURENCYJNIE

```
int FibIt(int n){  
    int a = 0, b = 0, i, wynik = 0;  
  
    if (n<2) return n;  
    else{  
        for(i=2; i<=n; i++){  
  
            wynik = a + b;  
            a = b;  
            b = wynik;  
  
        }  
        return wynik;  
    }  
}
```

ITERACYJNIE

$$F_n := \begin{cases} 0 & \text{dla } n = 0; \\ 1 & \text{dla } n = 1; \\ F_{n-1} + F_{n-2} & \text{dla } n > 1. \end{cases}$$

```
int main(){  
    int i;  
    for(i=0; i<10; i++)  
        printf("%d) rek: %d,  
               it: %d\n", i,  
                      FibRek(i), FibIt(i));  
  
    return 0;  
}
```



Schemat Hornera

- ▶ **Schemat Hornera** – sposób obliczania wartości wielomianu dla danej wartości argumentu wykorzystujący minimalną liczbę mnożeń.

$$W(x) = a_0 + a_1x + a_2x^2 + a_3x^3 + \cdots + a_{n-2}x^{n-2} + a_{n-1}x^{n-1} + a_nx^n$$

$n(n+1)/2$ - mnożeń n - dodawań



$$W(x) = a_0 + x(a_1 + x(a_2 + \dots + x(a_{n-2} + x(a_{n-1} + xa_n))\dots)).$$

n - mnożeń n - dodawań

$$W_n(x) = \begin{cases} a_n & \text{dla } n = 0 \\ W_{n-1}(x) * x + a_n & \text{dla } n > 0 \end{cases}$$

Schemat Hornera

```
int HornerRek(int wsp[],int st, int x) {  
    if(st==0)  
        return wsp[0];  
  
    return x*HornerRek(wsp,st-1,x)+wsp[st];  
}
```

REKURENCYJNIE

```
int HornerIt(int wsp[],int st, int x) {  
    int i, wynik = wsp[0];  
  
    for(i=1;i<=st;i++)  
        wynik = wynik*x + wsp[i];  
  
    return wynik;  
}
```

ITERACYJNIE

Schemat Hornera

```
int main() {
    int *wsp, st, arg, i;

    printf("Podaj stopien wielomianu: ");
    scanf("%d", &st);

    wsp = (int*)malloc((st+1)*sizeof(int));           alokacja pamięci

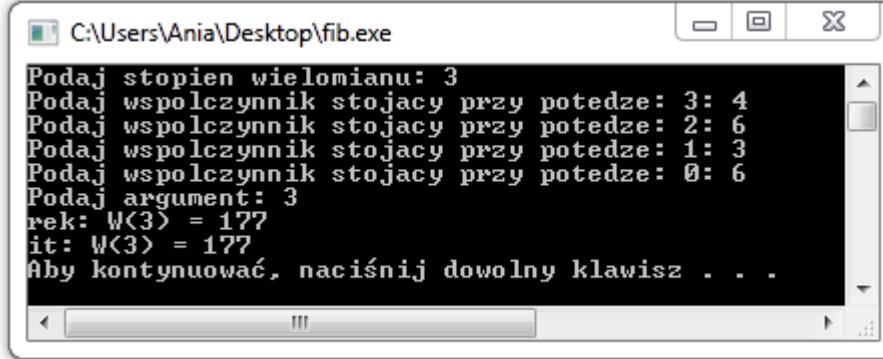
    //wczytanie współczynników
    for(i=0; i<=st; i++) {
        printf("Podaj współczynnik stojacy przy potedze: %d : ", st-i);
        scanf("%d", &wsp[i]);
    }

    printf("Podaj argument: ");
    scanf("%d", &arg);

    printf("rek: W(%d) = %d\n", arg, HornerRek(wsp,st,arg));
    printf("it: W(%d) = %d\n", arg, HornerIt(wsp,st,arg));

    free (wsp);                                     zwolnienie pamięci

    return 0;
}
```



Wyznacznik macierzy kwadratowej

$$\det(A_{[n \times n]}) = \begin{cases} a_{nn} & \text{dla } n = 1 \\ \sum_{k=1}^n a_{ik}(-1)^{i+k} \det(A_{ik}) & \text{dla } n > 1 \end{cases}$$

gdzie:

i - wiersz macierzy, względem którego następuje rozwinięcie

A_{ik} - macierz powstała przez skreślenie i -tego wiersza i j -tej kolumny w macierzy A

$$A = \begin{bmatrix} -2 & 2 & -3 \\ -1 & 1 & 3 \\ 2 & 0 & -1 \end{bmatrix},$$

$$\begin{aligned} \det(A) &= (-1)^{1+2} \cdot 2 \cdot \begin{vmatrix} -1 & 3 \\ 2 & -1 \end{vmatrix} + (-1)^{2+2} \cdot 1 \cdot \begin{vmatrix} -2 & -3 \\ 2 & -1 \end{vmatrix} + (-1)^{3+2} \cdot 0 \cdot \begin{vmatrix} -2 & -3 \\ -1 & 3 \end{vmatrix} \\ &= (-2) \cdot ((-1) \cdot (-1) - 2 \cdot 3) + 1 \cdot ((-2) \cdot (-1) - 2 \cdot (-3)) \\ &= (-2) \cdot (-5) + 8 = 18. \end{aligned}$$

```
double det(int n, int w, int * WK, double ** A) {  
    int i, j, k, m, *KK;  
    double s;  
  
    if(n == 1) // przypadek jednoelementowy  
        return A[w][WK[0]];  
    else {  
  
        KK = (int*) malloc ((n-1) * sizeof (int)); // wektor kolumn  
        s = 0; m = 1;  
  
        for(i = 0; i < n; i++) { // pętla obliczająca rozwinięcie  
            k = 0; // wektor kolumn dla rekurencji  
  
            for(j = 0; j < n - 1; j++) {  
                if(k == i) k++; // pominięcie bieżącej kolumny  
                KK[j] = WK[k++]; // przeniesienie pozostałych kolumn do KK  
            }  
  
            s += m * A[w][WK[i]] * det(n - 1, w + 1, KK, A);  
            m = -m; // kolejny mnożnik  
        }  
  
        free (KK);  
        return s;  
    }  
}
```

Przykład 127₍₁₎

Przykład 127₍₂₎

```
int main() {
    int n, i, j, *WK;
    double ** A;

    printf("Podaj rozmiar macierzy: ");
    scanf("%d", &n);                                dynamiczna alokacja i wypełnienie macierzy

    A = (double**) malloc (n * sizeof(double*)) ;

    for(i = 0; i < n; i++) {
        A[i] = (double*) malloc(n * sizeof(double));
        for(j = 0; j < n; j++) scanf("%lf", &A[i][j]);
    }

    WK = (int*) malloc (n*sizeof(int));
    for(i = 0; i < n; i++)
        WK[i] = i;

    printf("Wyznacznik wynosi: %lf", det(n, 0, WK, A));      zwalnianie pamięci

    free(WK);

    for(i = 0; i < n; i++) free(A[i]);

    free(A);
}
```

Sortowanie przez scalanie

▶ Algorytm scalania zbiorów uporządkowanych:

- ▶ porównaj ze sobą pierwsze elementy z każdego ze scalanych zbiorów;
- ▶ mniejszy element wstaw do nowego zbioru (wynikowego) usuwając go jednocześnie ze zbioru źródłowego;
- ▶ powtarzaj czynności 1 i 2, aż oba scalane zbiory będą puste.

▶ Algorytm sortowania przez scalanie:

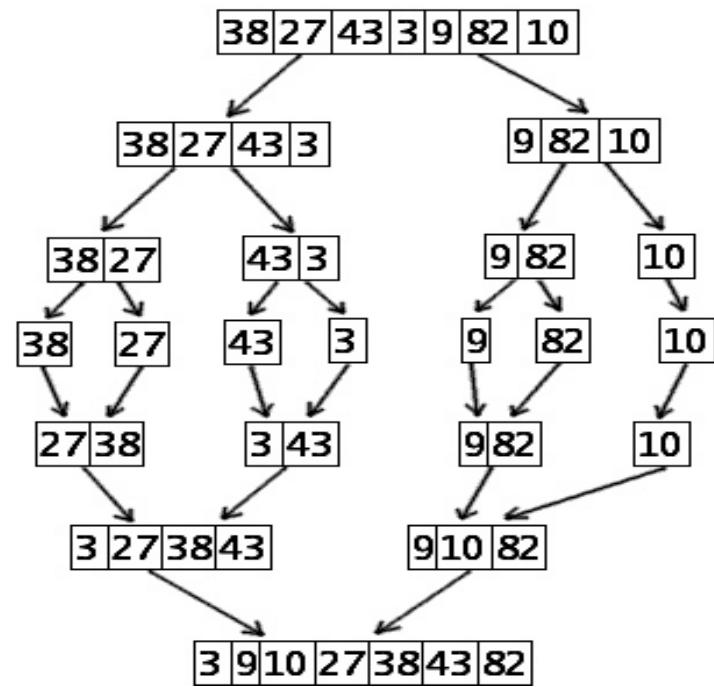
- ▶ jeśli zbiór zawiera więcej niż jeden element, to podziel go na dwie równe podzbiory (lub prawie równe, jeśli zbiór sortowany ma nieparzystą liczbę elementów);

- ▶ posortuj pierwszy podzbiór stosując ten sam algorytm;
- ▶ posortuj drugi podzbiór stosując ten sam algorytm;
- ▶ połącz dwa uporządkowane podzbiory w jeden zbiór uporządkowany.

rekurencja

Wynikiem scalenia dwóch uporządkowanych zbiorów:

{1 3 6 7 9} i {2 3 4 6 8} jest zbiór:
{1 2 3 3 4 6 6 7 8 9}.



Przykład 128₍₁₎

```
#include <stdio.h>
#define N 20
/*********************************************************/
void merge(int, int, int, int[]);
void mergesort(int, int, int[]);
/*********************************************************/
int main(){

    int i;
    int tab[N] = {29,28,27,25,1,2,3,5,6,24,22,20,18,8,10,11,17,15,13,12};

    printf("Zbior przed sortowaniem:\n");
    for (i=0; i<N; i++)
        printf("%d ", tab[i]);

    mergesort(0,N-1, tab);
}

printf("\nzbior po sortowaniu:\n");
for (i=0; i<N; i++)
    printf("%d ", tab[i]);

return 0;
}
```

wywołanie funkcji sortującej

Przykład 128₍₂₎

```
void merge(int pocz, int sr, int kon, int tab[N]) {  
  
    int i, j, q;  
    int t[N];  
  
    for (i=pocz; i<=kon; i++)  
        t[i]=tab[i];  
  
    i=pocz; j=sr+1; q=pocz;  
  
    while (i<=sr && j<=kon) {  
        if (t[i]<t[j])  
            tab[q++]=t[i++];  
        else  
            tab[q++]=t[j++];  
    }  
  
    while (i<=sr)  
        tab[q++]=t[i++];  
}
```

Kopiowanie danych do tablicy pomocniczej

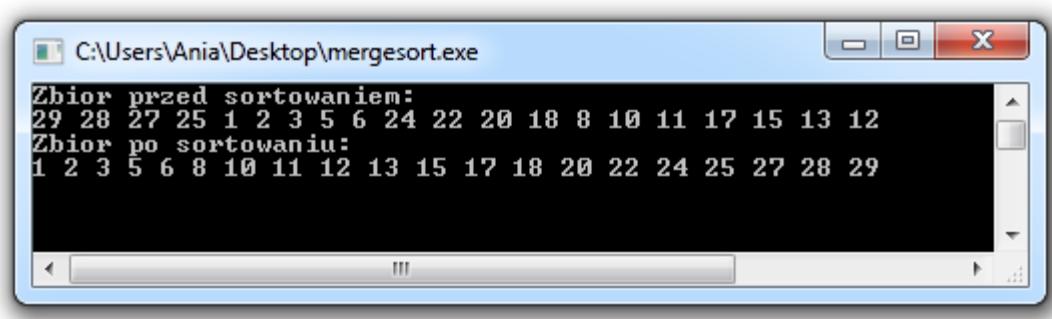
Przenoszenie danych z sortowaniem ze zbiorów pomocniczych do tablicy głównej

Przeniesienie nie skopiowanych danych ze zbioru pierwszego w przypadku, gdy drugi zbiór się skończył

Przykład 128(3)

```
void mergesort(int pocz, int kon, int tab[N]) {  
    int sr;  
  
    if (pocz<kon) {  
  
        sr=(pocz+kon)/2;  
        mergesort(pocz, sr, tab);  
        mergesort(sr+1, kon, tab);  
        merge(pocz, sr, kon, tab);  
    }  
}
```

sortowanie lewego podcięgu
sortowanie prawego podcięgu
scalanie podcięgów



Rekurencja wzajemna

- ▶ **Wzajemna rekurencja** (rekurencja pośrednia) dwóch funkcji występuje wówczas, kiedy pierwsza funkcja zawiera wywołanie funkcji drugiej, która ponownie wywołuje funkcję pierwszą

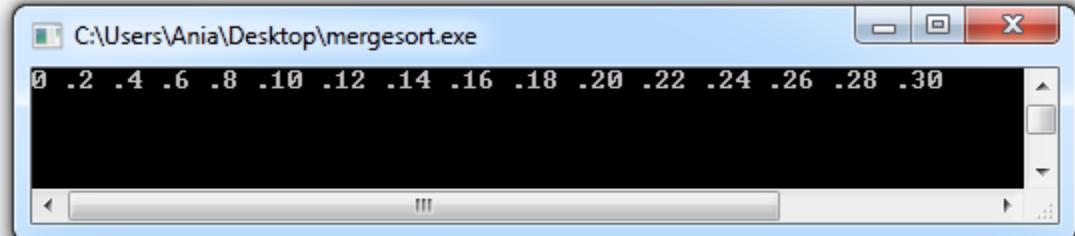
Przykład 130

```
#include <stdio.h>
/*****************/
void f1(int);
void f2(int);
/*****************/
int main(void) {
    f1(30);

    getchar();
    return 0;
}
/*****************/
void f1(int a) {
    if(a) f2(a-1);
    printf("%d ", a);
}

}
/*****************/
void f2(int b) {
    if(b) f1(b-1);
    printf(".");
}

}
```

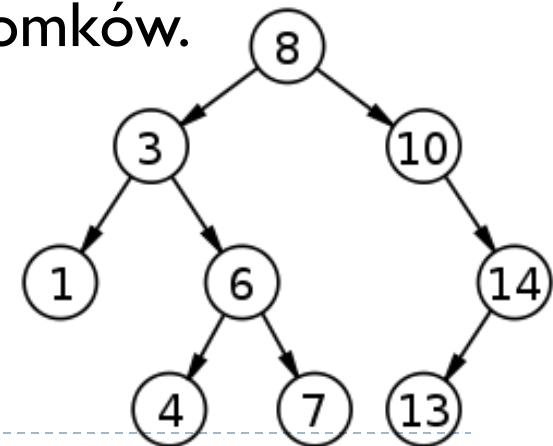


Drzewo binarne

- ▶ **Drzewo** (ang. tree) dynamiczna struktura zbudowana z węzłów (ang. nodes)
 - ▶ każdy węzeł może posiadać jednego **rodzica** (ang. parent node) oraz kilku **potomków** (ang. child node)
 - ▶ węzeł nieposiadający rodzica to **korzeń** lub **węzeł główny** (ang. root node)
 - ▶ węzeł nie posiadający potomków to **liść** (ang. leaf)
 - ▶ drzewo może posiadać wiele liści lecz tylko jeden korzeń.
- ▶ **Drzewo binarne** (ang. binary tree) to drzewo, w którym każdy węzeł może posiadać co najwyżej dwóch potomków.

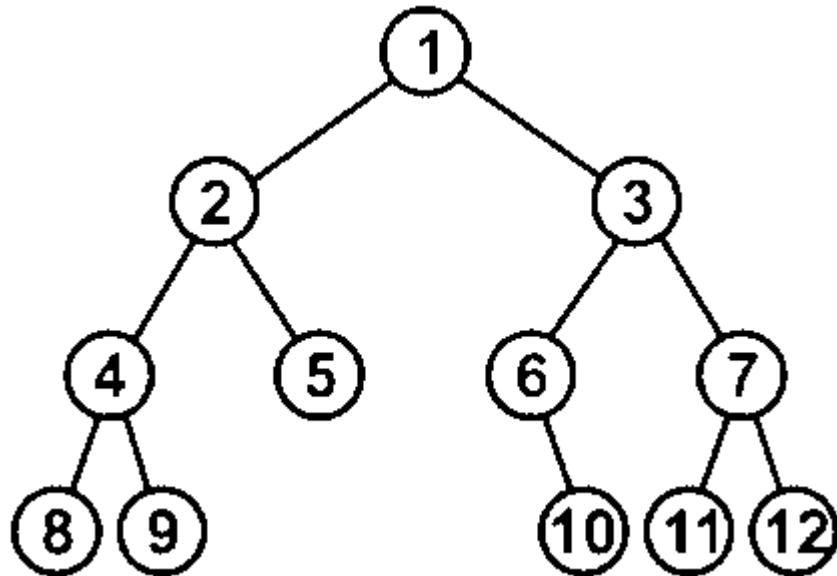
Struktura cykliczna

```
struct node
{  node * parent;
   node * left;
   node * right;
   ...
};
```



Sposób reprezentacji drzewa

- ▶ Jednym ze sposobów reprezentacji drzew binarnych jest **lista sąsiedztwa**:
 - ▶ $L[i] = \{\text{rodzic, lewy potomek, prawy potomek}\}$



w – wierzchołek

l – lewy potomek

r – rodzic

p – prawy potomek

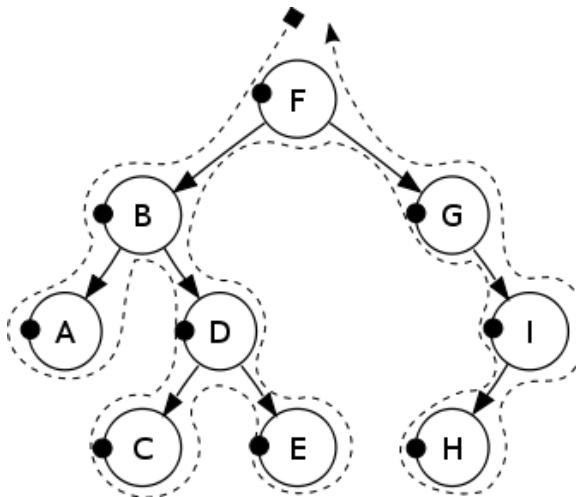
w	r	l	p
1	0	2	3
2	1	4	5
3	1	6	7
4	2	8	9
5	2	0	0
6	3	0	10
7	3	11	12
8	4	0	0
9	4	0	0
10	6	0	0
11	7	0	0
12	7	0	0

Przechodzenie drzewa

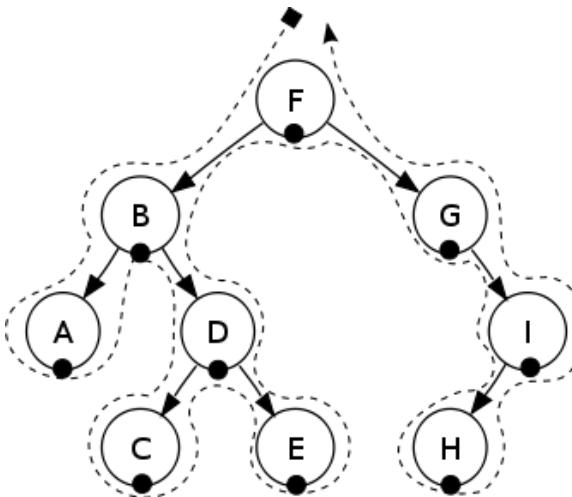
- ▶ **Przechodzenie drzewa** (przechodzenie po drzewie) to proces odwiedzania każdego wierzchołka drzewa
 - ▶ **przejście wzduż** (ang. *pre-order*)
 - ▶ **korzeń -> lewe poddrzewo -> prawe poddrzewo**
 - ▶ działanie jest wykonywane najpierw na rodzicu, następnie na synach
 - ▶ **przejście poprzeczne** (ang. *in-order*)
 - ▶ **lewe poddrzewo -> korzeń -> prawe poddrzewo**
 - ▶ najpierw wykonywane jest działanie na jednym z synów, następnie na rodzicu i na końcu na drugim synu.
 - ▶ **przejście wsteczne** (ang. *post-order*)
 - ▶ **lewe poddrzewo -> prawe poddrzewo -> korzeń**
 - ▶ działanie jest wykonywane najpierw na wszystkich synach, na końcu na rodzicu
 - ▶ **przejście wszerz BFS** (ang. *Bridth First Search*)
 - ▶ przechodzi przez kolejne węzły na poszczególnych poziomach drzewa binarnego

Przechodzenie drzewa

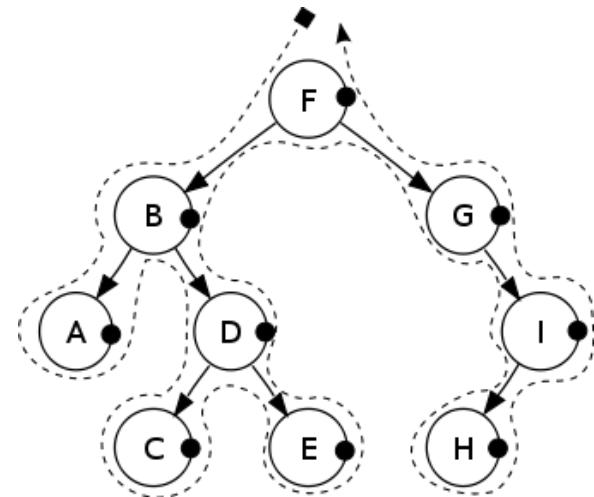
Pre-order:



In-order:



Post-order:



FBADCEGIH

preorder(v)

- przetwórz węzeł v
- if($L[v].left > 0$)
 preorder($L[v].left$)
- if($L[v].right > 0$)
 preorder($L[v].right$)
- koniec

ABCDEGFI

inorder(v)

- if ($L[v].left > 0$)
 inorder($L[v].left$)
- przetwórz węzeł v
- if($L[v].right > 0$)
 inorder($L[v].right$)
- koniec

ACEDBHIGF

postorder(v)

- if ($L[v].left > 0$)
 postorder($L[v].left$)
- if($L[v].right > 0$)
 postorder($L[v].right$)
- przetwórz węzeł v
- koniec

Przykład 129(1)

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
/*****
struct list_elem{
    int parent;
    int left;
    int right;
};
/*****
void preorder(int v, struct list_elem L[], int nr_preorder[], int *nr)
{
    nr_preorder[v] = (*nr)++;
    if(L[v].left)  preorder(L[v].left, L, nr_preorder, nr);
    if(L[v].right) preorder(L[v].right, L, nr_preorder, nr);
}
/*****
void inorder(int v, struct list_elem L[], int nr_inorder[], int *nr)
{
    if(L[v].left)  inorder(L[v].left, L, nr_inorder, nr);
    nr_inorder[v] = (*nr)++;
    if(L[v].right) inorder(L[v].right, L, nr_inorder, nr);
}
/*****
void postorder(int v, struct list_elem L[] , int nr_postorder[], int *nr)
{
    if(L[v].left)  postorder(L[v].left, L, nr_postorder, nr);
    if(L[v].right) postorder(L[v].right, L, nr_postorder, nr);
    nr_postorder[v] = (*nr)++;
}
```

Przykład 129₍₂₎

```
*****
struct list_elem* utworzListeSasiedztwa(int n, int m)
{
    int i, c,l,r;

    struct list_elem* L =
        (struct list_elem*)malloc((n+1)*sizeof(struct list_elem));

    for (i=1; i<=n; i++) {
        L[i].parent = 0;
        L[i].left = 0;
        L[i].right = 0;
    }

    for(i = 1; i <= m; i++) {
        printf("Podaj numer wierzcholka oraz prawego i lewego potomka: ");
        scanf("%d %d %d", &c, &l, &r);
        L[c].left = l; L[c].right = r;
        if(l) L[l].parent = c;
        if(r) L[r].parent = c;
    }

    return L;
}
*****
```

Przykład 129(3)

```
*****  
void drukujListeSasiedztwa(struct list_elem L[], int n)  
{  
    int i;  
    printf("-----\n");  
    for(i = 1; i <= n; i++)  
        printf("%d: | \t%d\t%d\t%d\n", i, L[i].parent, L[i].left, L[i].right);  
    printf("-----\n");  
}  
*****  
void drukujWierzcholki(int preorder[], int inorder[], int postorder[], int n)  
{  
    int i;  
    printf("-----\n");  
    printf("w\t pre\t in\t post\n");  
    printf("-----\n");  
    for(i = 1; i <= n; i++)  
        printf("%d\t | %d\t | %d\t | %d \t|\n",  
               i, preorder[i], inorder[i], postorder[i]);  
    printf("-----\n");  
}  
*****
```

Przykład 129₍₄₎

```
int main(void) {
    int n, m, i, nr;

    printf("Podaj liczbę wierzchołków w drzewie: ");
    scanf("%d", &n);

    printf("Podaj liczbę wierzchołków z potomkiem: ");
    scanf("%d", &m);

    int* nr_preorder = (int*)malloc((n+1)*sizeof(int));
    int* nr_inorder = (int*)malloc((n+1)*sizeof(int));
    int* nr_postorder = (int*)malloc((n+1)*sizeof(int));

    struct list_elem* L = utworzListeSasiedztwa(n, m);
    drukujListeSasiedztwa(L, n);

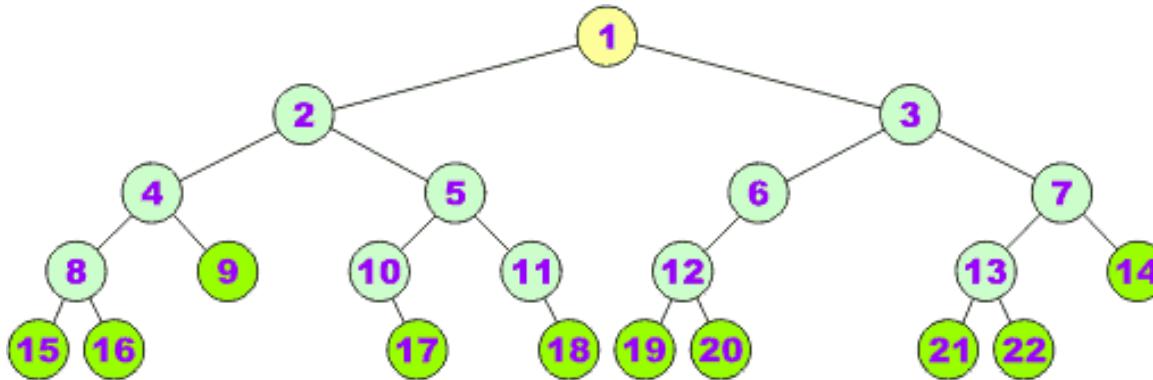
    nr = 1; preorder(1, L, nr_preorder, &nr);
    nr = 1; inorder(1, L, nr_inorder, &nr);
    nr = 1; postorder(1, L, nr_postorder, &nr);

    drukujWierzchołki(nr_preorder, nr_inorder, nr_postorder, n);

    free(nr_preorder);
    free(nr_inorder);
    free(nr_postorder);
    free(L);

    return 0;
```

Przykład 129 (wynik działania programu)



w	pre	in	post
1:	0	2	3
2:	1	4	5
3:	1	6	7
4:	2	8	9
5:	2	10	11
6:	3	12	0
7:	3	13	14
8:	4	15	16
9:	4	0	0
10:	5	0	17
11:	5	0	18
12:	6	19	20
13:	7	21	22
14:	7	0	0
15:	8	0	0
16:	8	0	0
17:	10	0	0
18:	11	0	0
19:	12	0	0
20:	12	0	0
21:	13	0	0
22:	13	0	0

Lista sąsiedztwa

w	pre	in	post
1	1	12	22
2	2	6	11
3	13	17	21
4	3	4	5
5	8	9	10
6	14	16	15
7	18	21	20
8	4	2	3
9	7	5	4
10	9	7	7
11	11	10	9
12	15	14	14
13	19	19	18
14	22	22	19
15	5	1	1
16	6	3	2
17	10	8	6
18	12	11	8
19	16	13	12
20	17	15	13
21	20	18	16
22	21	20	17

Ponumerowane wierzchołki

Przykład 131

```
clang-7 -pthread -lm -o main main.c
./main
F: 1 1 2 2 3 3 4 4 5 5 6 6 7 7 8 8 9 9 10 10 11 11 12 12 13
M: 0 0 1 1 2 2 3 3 4 4 5 5 6 6 7 7 8 8 9 9 10 10 11 11 12 12 13
```

```
#include <stdio.h>

int hofstaderFemale(int);
int hofstaderMale(int);

int hofstaderFemale(int n)
{
    return (n == 0) ? 1 : n - hofstaderMale(n - 1);
}

int hofstaderMale(int n)
{
    return (n == 0) ? 0 : n - hofstaderFemale(n - 1);
}

int main()
{
    int i;
    printf("F: ");
    for (i = 0; i < 25; i++)
        printf("%d ", hofstaderFemale(i));

    printf("\n");
    printf("M: ");
    for (i = 0; i < 25; i++)
        printf("%d ", hofstaderMale(i));
    return 0;
}
```

Sekwencja Hofstadera

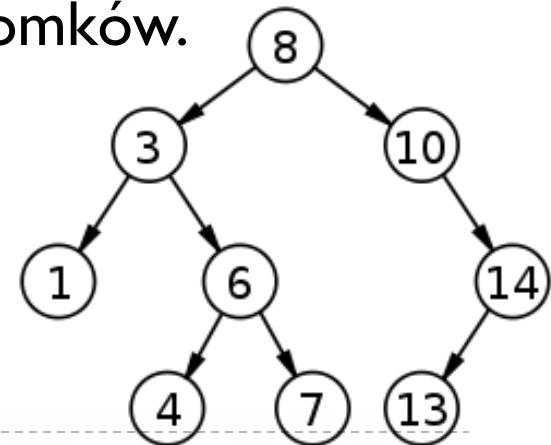
$$\begin{aligned} F(0) &= 1 \\ M(0) &= 0 \\ F(n) &= n - M(F(n-1)), n > 0 \\ M(n) &= n - F(M(n-1)), n > 0. \end{aligned}$$

Drzewo binarne

- ▶ **Drzewo** (ang. tree) dynamiczna struktura zbudowana z węzłów (ang. nodes)
 - ▶ każdy węzeł może posiadać jednego **rodzica** (ang. parent node) oraz kilku **potomków** (ang. child node)
 - ▶ węzeł nieposiadający rodzica to **korzeń** lub **węzeł główny** (ang. root node)
 - ▶ węzeł nie posiadający potomków to **liść** (ang. leaf)
 - ▶ drzewo może posiadać wiele liści lecz tylko jeden korzeń.
- ▶ **Drzewo binarne** (ang. binary tree) to drzewo, w którym każdy węzeł może posiadać co najwyżej dwóch potomków.

```
struct node
{
    struct node * parent;
    struct node * left;
    struct node * right;
    int data;
};
```

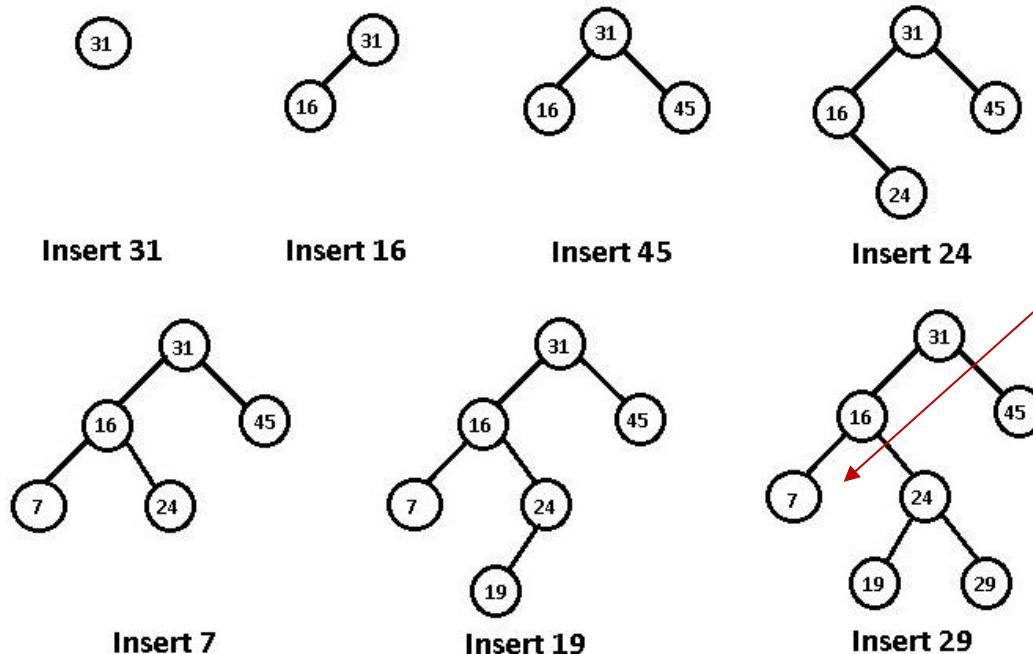
```
struct tree{
    struct node* root;
};
```



Drzewa binarne

Dodawanie elementów:

1. Po prawej stronie – element **większy** od rodzica
2. Po lewej stronie – element **mniejszy** od rodzica



Najmniejszy element:

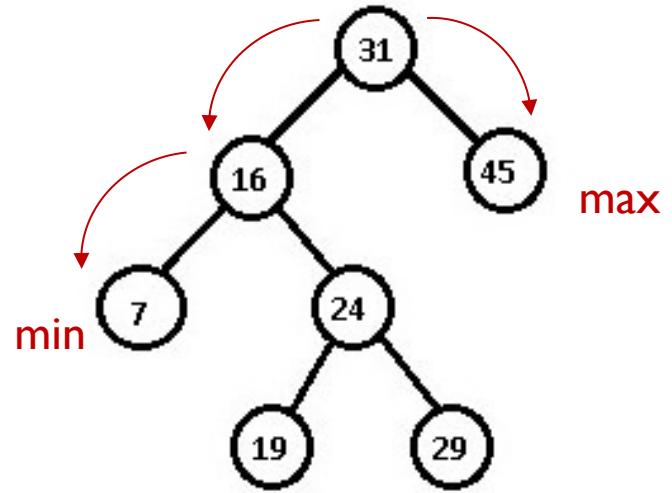
- Zawsze najbardziej „po lewo”

Największy element:

- Zawsze najbardziej „po prawo”

Przykład 132₍₁₎

```
struct node * minValueNode(struct tree* t){  
    struct node* temp = t->root;  
  
    while (temp->left)  
        temp = temp->left;  
    return temp;  
}  
  
struct node * maxValueNode(struct tree* t){  
    struct node* temp = t->root;  
  
    while (temp->right)  
        temp = temp->right;  
    return temp;  
}
```



```

void insertNode(struct tree* t, int data) {
    struct node *new = createNode(data);
    if(!t->root) {
        t->root = new;
        return;
    }
    struct node *temp = t->root;
    while (temp) {
        if (temp->data <= new->data) {
            if (!temp->right) {
                temp->right = new;
                break;
            }
            temp = temp->right;
        }
        if (temp->data > new->data) {
            if (!temp->left) {
                temp->left = new;
                break;
            }
            temp = temp->left;
        }
    }
}

```

w pustym drzewie element będzie korzeniem

Przykład 132(2)

Jeśli wartość nowego węzła jest większa od aktualnego, przejdź w prawo. Wstaw węzeł w miejsce napotkanego NULL

Inaczej przejdź w głąb drzewa

```

struct node* createNode(int data) {
    struct node * item
        = malloc(sizeof(struct node));
    item->left = NULL;
    item->right = NULL;
    item->data = data;
    return item;
}

```

```
int treeDepth(struct node* root) {  
    if (root) {  
        int dL = treeDepth(root->left);  
        int dR = treeDepth(root->right);  
        int height = dL > dR ? dL : dR;  
        return height + 1;  
    }  
    return 0;  
}
```

Przykład 132(3)

sprawdzenie wysokości (głębokości) drzewa

```
void printGivenLevel(struct node* root, int level)  
{  
    if (root == NULL)  
        return;  
    if (level == 1)  
        printf("%d ", root->data);  
    else if (level > 1) {  
        printGivenLevel(root->left, level-1);  
        printGivenLevel(root->right, level-1);  
    }  
}
```

drukowanie każdego poziomów osobno (rekurencja)

```
void printLevelOrder(struct node* root)  
{
```

```
    printf("\n***** \n");  
    int h = treeDepth(root);  
    int i;  
    for (i=1; i<=h; i++) {  
        printGivenLevel(root, i);  
        printf("\n");  
    }
```

drukowanie całego drzewa

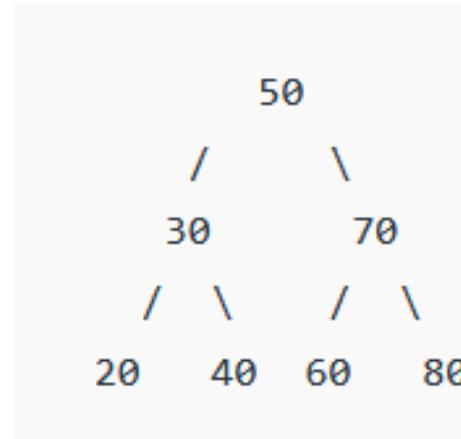
Przykład 132₍₄₎

```
int main()
{
    struct tree T;
    T.root = NULL;

    insertNode(&T, 50);
    insertNode(&T, 30);
    insertNode(&T, 20);
    insertNode(&T, 40);
    insertNode(&T, 70);
    insertNode(&T, 60);
    insertNode(&T, 80);

    printf("Tree height: %d \n", treeDepth(T.root));
    printLevelOrder(T.root);

    printf("\nMin: %d", minValueNode(&T)->data);
    printf("\nMax: %d\n", maxValueNode(&T)->data);
}
```



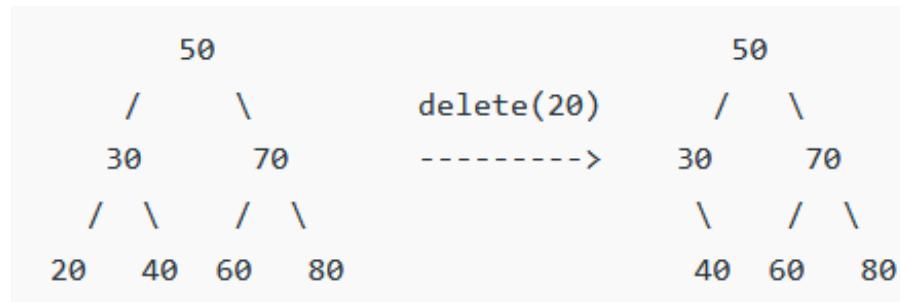
```
Tree height: 3
*****
50
30 70
20 40 60 80

Min: 20
Max: 80
```

Drzewa binarne

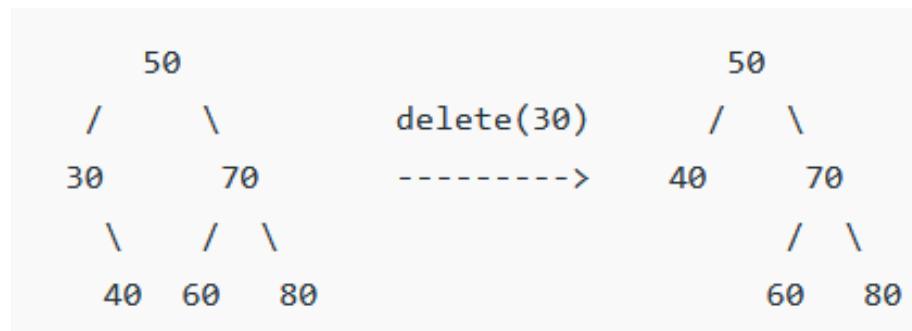
Usuwanie elementów:

I. Usuwany element jest liściem



Po prostu usuń element

2. Usuwany element ma jednego potomka

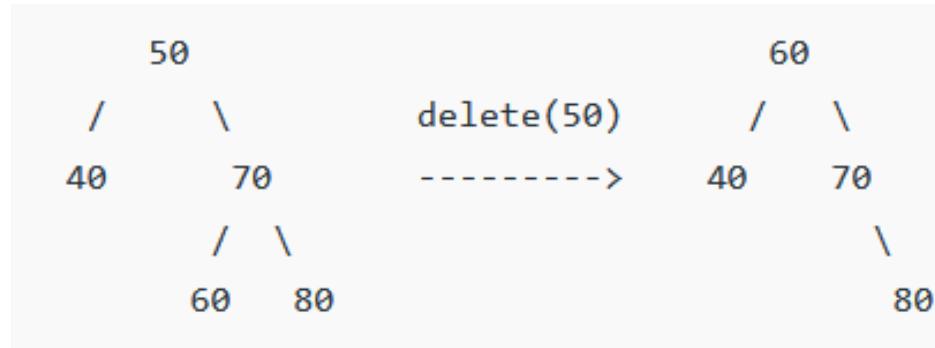


Przenieś potomka w miejsce
usuwanego rodzica

Drzewa binarne

Usuwanie elementów:

3. Usuwany element ma dwóch potomków



Znajdź następcę in order (minimum w prawym poddrzewie)
i przenieś go w miejsce usuwanego rodzica

```
struct node* deleteNode(struct node* root, int key)
{
    if (root == NULL)
        return root;

    if (key < root->data)
        root->left = deleteNode(root->left, key);

    else if (key > root->data)
        root->right = deleteNode(root->right, key);

    else {
        if (root->left == NULL) {
            struct node* temp = root->right;
            free(root);
            return temp;
        }
        else if (root->right == NULL) {
            struct node* temp = root->left;
            free(root);
            return temp;
        }
    }

    struct node* temp = minValueNode(root->right);

    root->data = temp->data;
    root->right = deleteNode(root->right, temp);

}
return root;
}
```

Przykład 132(5)

Wierzchołek z nie więcej niż jednym potomkiem

Wierzchołek z dwoma potomkami

Tree height: 3

Przykład 132(6)

```
int main()
{
    struct tree T;
    T.root = NULL;

    insertNode(&T, 50);
    insertNode(&T, 30);
    insertNode(&T, 20);
    insertNode(&T, 40);
    insertNode(&T, 70);
    insertNode(&T, 60);
    insertNode(&T, 80);

    printf("Tree height: %d \n", treeDepth(T.root));
    printLevelOrder(T.root);

    printf("\nMin: %d", minValueNode(&T) ->data);
    printf("\nMax: %d\n", maxValueNode(&T) ->data);

    deleteNode(T.root, 20);
    printLevelOrder(T.root);

    deleteNode(T.root, 30);
    printLevelOrder(T.root);

    deleteNode(T.root, 50);
    printLevelOrder(T.root);
}
```

50
30 70
20 40 60 80

Min: 20
Max: 80

50
30 70
40 60 80

50
40 70
60 80

60
40 70
80

Process finished with exit code 0

Wybrane funkcje biblioteczne

- ▶ Funkcje i makra standardowej biblioteki zadeklarowane są w standardowych nagłówkach:

- **<assert.h>** diagnostyka
- **<float.h>** ograniczenia określone przez implementację
- **<math.h>** funkcje matematyczne
- **<stdarg.h>** listy argumentów o zmiennej długości
- **<stdlib.h>** funkcje narzędziowe
- **<ctype.h>** wykrywanie przynależności znaków do określonej klasy
- **<limits.h>** ograniczenia określone przez implementację
- **<setjmp.h>** skoki odległe
- **<stddef.h>** standardowe definicje
- **<string.h>** operacje na ciągach znakowych
- **<errno.h>** raportowanie błędów
- **<locale.h>** ustawienia lokalne (międzynarodowe)
- **<signal.h>** sygnały
- **<stdio.h>** operacje wejścia-wyjścia
- **<time.h>** data i godzina

<assert.h>

diagnostyka

- ▶ Plik nagłówkowy **assert.h** definiuje makro preprocesora **assert()**
- ▶ Makro **assert()** umożliwia dodanie do programu mechanizmu diagnostycznego:

```
void assert(int wyrażenie)
```

- ▶ Jeżeli *wyrażenie* ma wartość zero (fałsz) w chwili wykonania polecenia, to makro **assert()** wypisuje komunikat:

```
Assertion failed: wyrażenie, file nazwa pliku, line numer
```

- ▶ oraz wywołuje funkcję **abort()**, aby zakończyć pracę programu.

Przykład 133

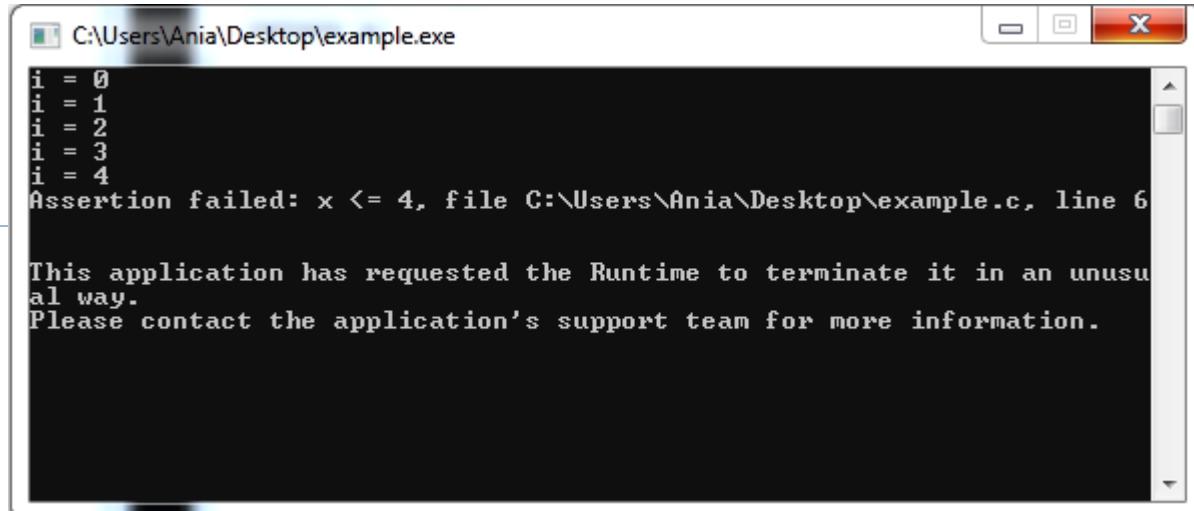
```
#include <stdio.h>
#include <assert.h>
```

```
int test_assert (int x)
{
    assert(x <= 4);
    return x;
}

int main (void)
{
    int i;

    for (i=0; i<=9; i++){
        test_assert( i );
        printf("i = %i\n", i);
    }

    return 0;
}
```



Przykład 134

```
#include <stdio.h>
#include <math.h>
#include <assert.h>

int main() {

    double x, y, z;
    puts("Podaj pare liczb (0 0 konczy program) : ");

    while(scanf("%lf %lf", &x, &y) == 2 && (x!=0 || y!=0))
    {
        z = x * x - y * y;
        assert(z >= 0);

        printf("Wynik wynosi %f\n", sqrt(z));
        puts("Kolejna para liczb: ");
    }

    return 0;
}
```

C:\Users\Ania\Desktop\example.exe

Podaj pare liczb (0 0 konczy program):
5 4
Wynik wynosi 3.000000
Kolejna para liczb:
4 5
Assertion failed: z >= 0, file C:\Users\Ania\Desktop\example.c, line 13

This application has requested the Runtime to terminate it in an unusual way.
Please contact the application's support team for more information.

<errno.h>

błędy

- ▶ Plik nagłówkowy **errno.h** umożliwia skorzystanie ze starszego mechanizmu raportowania błędów
- ▶ Tworzy miejsce w statycznej pamięci zewnętrznej, do którego dostęp następuje przez identyfikator **ERRNO**
 - ▶ niektóre funkcje biblioteczne umieszczają w tym miejscu wartość, aby zasygnalizować wystąpienie błędu.
- ▶ Makra:
 - ▶ **EDOM** - błąd zakresu argumentów przy wywoływaniu funkcji matematycznych (np. **sqrt (-1)**).
 - ▶ **ERANGE** - wynik działania funkcji matematycznych nie mieści się w typie wyniku (np. **exp (100000)** przekracza zakres **double**).
 - ▶ **EILSEQ** - napotkano błędna reprezentację wielobajtowego znaku.

Przykład 135

```
#include <stdio.h>
#include <errno.h>
#include <math.h>
#include <stdlib.h>

int main()
{
    errno = 0;
    double x, y;

    x = -10;
    y = sqrt(x);

    if(errno != 0) {
        fprintf(stderr, "sqrt error; program terminated. \n");
        exit(EXIT_FAILURE);
    }

    printf("sqrt(%.2f) = %.2f\n", x, y);

    return 0;
}
```

sqrt error; program terminated.

Process finished with exit code 1

Przykład 136

```
#include <stdio.h>
#include <errno.h>
#include <math.h>
#include <stdlib.h>

int main()
{
    errno = 0;
    double x, y;

    x = 10000;
    y = exp(x);

    if(errno == EDOM) {
        fprintf(stderr, "Argument poza zakresem \n");
        exit(EXIT_FAILURE);
    }
    if(errno == ERANGE) {
        fprintf(stderr, "Wartość zwracana poza zakresem \n");
        exit(EXIT_FAILURE);
    }
    printf("exp(% .2f) = % .2f\n", x, y);

    return 0;
}
```

Wartość zwracana poza zakresem

Process finished with exit code 1

<stdio.h> oraz <string.h> błędy - cd

- ▶ Plik nagłówkowy **stdio.h** zawiera funkcję

```
void perror(const char*)
```

- ▶ która wysyła komunikaty błędów systemowych do standardowego wyjścia błędów
 - ▶ Plik nagłówkowy **string.h** zawiera funkcję
- ```
char * strerror(int errnum)
```
- ▶ która zwraca wskaźnik do komunikatu o błędzie (zależnego od implementacji) odpowiadającego numerowi błędu **errnum**

# Przykład 137

```
#include <stdio.h>
#include <errno.h>
#include <math.h>
#include <stdlib.h>
#include <string.h>

int main()
{
 errno = 0;
 double x, y;

 x = -10;
 y = sqrt(x);

 if(errno != 0){
 perror("sqrt failure");
 puts(strerror(EDOM));
 exit(EXIT_FAILURE);
 }

 printf("sqrt(% .2f) = % .2f\n", x, y);

 return 0;
}
```

```
sqrt failure: Numerical argument out of domain
Numerical argument out of domain

Process finished with exit code 1
```

# <locale.h>

## ustawienia lokalne

- ▶ Zbiór ustawień lokalnych, kontrolujących m.in..
  - ▶ symbol używany jako separator w ułamkach dziesiętnych (78.5 vs. 78,5)
  - ▶ symbol waluty
  - ▶ zestaw znaków
  - ▶ formatowanie daty i czasu
- ▶ Poprzez zmianę ustawień lokalnych można adaptować program do różnych regionów

```
struct lconv *localeconv(void) ;
```

- ▶ Zwraca wskaźnik do struktury typu **struct lconv** zawierającej aktualne ustawienia lokalne

# <locale.h>

## ustawienia lokalne

```
char *setlocale(int category, const char *locale)
```

- ▶ ustawia wybrane parametry zgodnie z wartościami określonymi przez ustawienia w łańcuchu *locale*
- ▶ argument *category* decyduje o tym, które wartości ustawień zostaną zmienione

| Macro       | Description                                                                                                                             |
|-------------|-----------------------------------------------------------------------------------------------------------------------------------------|
| NULL        | Leave the locale unchanged and return a pointer to the current locale.                                                                  |
| LC_ALL      | Change all locale values.                                                                                                               |
| LC_COLLATE  | Change locale values for the collating sequence used by <code>strcoll()</code> and <code>strxfrm()</code> .                             |
| LC_CTYPE    | Change locale values for the character-handling functions and the multibyte functions.                                                  |
| LC_MONETARY | Change locale values for monetary-formatting information.                                                                               |
| LC_NUMERIC  | Change locale values for the decimal point symbol and non-monetary formatting used by formatted I/O and by string-conversion functions. |
| LC_TIME     | Change locale values for the time formatting used by <code>strftime()</code> .                                                          |

# Przykład 138

```
#include <locale.h>
#include <stdio.h>
#include <time.h>

int main () {
 time_t currtime;
 struct tm *timer;
 char buffer[80];

 time(&currtime);
 timer = localtime(&currtime);

 printf("Locale is: %s\n", setlocale(LC_ALL, "en_GB"));
 strftime(buffer,80,"%c", timer);
 printf("Date is: %s\n", buffer);

 printf("Locale is: %s\n", setlocale(LC_ALL, "de_DE"));
 strftime(buffer,80,"%c", timer);
 printf("Date is: %s\n", buffer);

 printf("Locale is: %s\n", setlocale(LC_ALL, "pl_PL"));
 strftime(buffer,80,"%c", timer);
 printf("Date is: %s\n", buffer);

 printf("Locale is: %s\n", setlocale(LC_ALL, "ja_JP.UTF-8"));
 strftime(buffer,80,"%c", timer);
 printf("Date is: %s\n", buffer);

 return(0);
}
```

```
Locale is: en_GB
Date is: 06 Jun 2021 00:07:18
Locale is: de_DE
Date is: So, 6. Jun 2021 00:07:18
Locale is: pl_PL
Date is: niedz., 6 cze 2021 00:07:18
Locale is: ja_JP.UTF-8
Date is: 2021年06月06日 00時07分18秒

Process finished with exit code 0
```

```

#include <locale.h>
#include <stdio.h>

int main () {
 struct lconv* loc;

 printf("Locale is: %s\n", setlocale(LC_ALL, "en_GB.UTF-8"));
 loc = localeconv();
 printf("Currency symbol is: %s (%s)\n",
 loc->currency_symbol,
 loc->int_curr_symbol);

 printf("Locale is: %s\n", setlocale(LC_ALL, "de_DE.UTF-8"));
 loc = localeconv();
 printf("Currency symbol is: %s (%s)\n",
 loc->currency_symbol,
 loc->int_curr_symbol);

 printf("Locale is: %s\n", setlocale(LC_ALL, "pl_PL.UTF-8"));
 loc = localeconv();
 printf("Currency symbol is: %s (%s)\n",
 loc->currency_symbol,
 loc->int_curr_symbol);

 printf("Locale is: %s\n", setlocale(LC_ALL, "ja_JP.UTF-8"));
 loc = localeconv();
 printf("Currency symbol is: %s (%s)\n",
 loc->currency_symbol,
 loc->int_curr_symbol);

 return(0);
}

```

## Przykład 139

```

Locale is: en_GB.UTF-8
Currency symbol is: £ (GBP)
Locale is: de_DE.UTF-8
Currency symbol is: € (EUR)
Locale is: pl_PL.UTF-8
Currency symbol is: zł (PLN)
Locale is: ja_JP.UTF-8
Currency symbol is: ¥ (JPY)

```

# <ctype.h>

## funkcje operujące na znakach

---

- ▶ **int isalnum (int c)** - sprawdza czy znak jest liczbą lub literą,
- ▶ **int isalpha (int c)** - sprawdza czy znak jest literą,
- ▶ **int isblank (int c) (C99)** - sprawdza czy znak jest znakiem odstępu oddzielającym wyrazy,
- ▶ **int iscntrl (int c)** - sprawdza czy znak jest znakiem sterującym,
- ▶ **int isdigit (int c)** - sprawdza czy znak jest cyfrą dziesiętną,
- ▶ **int isgraph (int c)** - sprawdza czy znak jest znakiem drukowalnym różnym od spacji,
- ▶ **int islower (int c)** - sprawdza czy znak jest małą literą,
- ▶ **int isprint (int c)** - sprawdza czy znak jest znakiem drukowalnym (włączając w to spację),
- ▶ **int ispunct (int c)** - sprawdza czy znak jest znakiem przestankowym,
- ▶ **int isspace (int c)** - sprawdza czy znak jest tzw. białym znakiem (spacja, '\f', '\n', '\r', '\t', '\v'),
- ▶ **int isupper (int c)** - sprawdza czy znak jest dużą literą,
- ▶ **int isxdigit (int c)** - sprawdza czy znak jest cyfrą szesnastkową,
- ▶ **int tolower (int c)** - sprawdza, czy dany znak znajduje się w zadanym przez locale zestawie dużych liter i jeśli tak przekształca go do odpowiedniej małej litery,
- ▶ **int toupper (int c)** - sprawdza, czy dany znak znajduje się w zadanym przez locale zestawie małych liter i jeśli tak przekształca go do odpowiedniej dużej litery

# Przykład 140

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

int main(){
 char ch;
 for(;;){
 ch = getchar();

 if(ch == EOF) break;

 if(isalnum(ch)) printf("%c jest znakiem alfanumerycznym\n", ch);
 if(isalpha(ch)) printf("%c jest litera\n", ch);
 if(iscntrl(ch)) printf("%c jest znakiem sterujacym\n", ch);
 if(isdigit(ch)) printf("%c jest cyfra\n", ch);
 if(isgraph(ch)) printf("%c jest znakiem drukowalnym (bez spacji)\n", ch);
 if(islower(ch)) printf("%c jest mala litera\n", ch);
 if(isupper(ch)) printf("%c jest mala litera\n", ch);
 if(isprint(ch)) printf("%c jest znakiem drukowalnym (ze spacja)\n", ch);
 if(ispunct(ch)) printf("%c jest znakiem przestankowym\n", ch);
 ifisspace(ch)) printf("%c jest znakiem odstepu\n", ch);
 if(isxdigit(ch)) printf("%c cyfra szesnastkowa\n", ch);
 printf("-----\n");
 while(getchar() != '\n');
 }
 return 0;
}
```

```
a
a jest znakiem alfanumerycznym
a jest litera
a jest znakiem drukowalnym (bez spacji)
a jest mala litera
a jest znakiem drukowalnym (ze spacja)
a cyfra szesnastkowa

P
P jest znakiem alfanumerycznym
P jest litera
P jest znakiem drukowalnym (bez spacji)
P jest mala litera
P jest znakiem drukowalnym (ze spacja)

9
9 jest znakiem alfanumerycznym
9 jest cyfra
9 jest znakiem drukowalnym (bez spacji)
9 jest znakiem drukowalnym (ze spacja)
9 cyfra szesnastkowa

```

```
jest znakiem drukowalnym (ze spacja)
jest znakiem odstepu

:
. jest znakiem drukowalnym (bez spacji)
. jest znakiem drukowalnym (ze spacja)
. jest znakiem przestankowym

@
@ jest znakiem drukowalnym (bez spacji)
@ jest znakiem drukowalnym (ze spacja)
@ jest znakiem przestankowym

\
\ jest znakiem drukowalnym (bez spacji)
\ jest znakiem drukowalnym (ze spacja)
\ jest znakiem przestankowym

^Z
```

## <math.h>

### operacje matematyczne(1)

---

- ▶ **sin(x)** sinus x (kąt podany w radianach),
- ▶ **cos(x)** cosinus x (kąt podany w radianach),
- ▶ **tan(x)** tangens x (kąt w radianach),
- ▶ **asin(x)** arcus sinus x (wynik w przedziale  $[-\pi/2, \pi/2]$ ),
- ▶ **acos(x)** arcus cosinus x (wynik w przedziale  $[0, \pi]$ ),
- ▶ **atan(x)** arcus tangens x (wynik w przedziale  $[-\pi/2, \pi/2]$ ),
- ▶ **atan2(x,y)** arcus tangens x/y (wynik w przedziale  $[-\pi, \pi]$ ),
- ▶ **sinh(x)** sinus hiperboliczny x,
- ▶ **cosh(x)** cosinus hiperboliczny x,
- ▶ **tanh(x)** tangens hiperboliczny x,

## <math.h>

# operacje matematyczne(2)

---

- ▶ **exp(x)** funkcja wykładnicza:  $e^x$ ,
- ▶ **exp2(x)** dwa do potęgi x
- ▶ **log(x)** logarytm naturalny:  $\ln(x)$ ,  $x > 0$ ,
- ▶ **log10(x)** logarytm o podstawie 10:  $\log_{10}(x)$ ,  $x > 0$ ,
- ▶ **pow(x,y)** potęgowanie:  $x^y$ ,
- ▶ **sqrt(x)** pierwiastek kwadratowy:
- ▶ **fabs(x)** wartość bezwzględna  $|x|$
- ▶ **ceil(x)** najmniejsza liczba całkowita nie mniejsza niż x,
- ▶ **floor(x)** największa liczba całkowita nie większa niż x,
- ▶ **trunc(x)** zaokrąglenie do liczby całkowitej w kierunku zera
- ▶ **round(x)** zaokrąglenie do najbliższej liczby całkowitej

## <math.h>

### operacje matematyczne(3)

- ▶ **cbrt(x)** pierwiastek sześcienny z x,
- ▶ **copysign(x)** kopiowanie znaku liczby,
- ▶ **fdim(x,y)** zwraca dodatnią różnicę pomiędzy argumentami,
- ▶ **fmax(x,y)** zwraca większą z wartości argumentów,
- ▶ **fmin(x,y)** zwraca mniejszą z wartości argumentów,
- ▶ **fmod (x,y)** reszta z dzielenia zmiennoprzecinkowego x przez y,
- ▶ **frexp** konwersja liczby zmiennopozycyjnej na część ułamkową i całkowitą
- ▶ **hypot(x,y)** pierwiastek kwadratowy z sumy kwadratów argumentów
- ▶ **llrint(x)** zaokrąglenie do najbliższej liczby całkowitej
- ▶ **llround(x)** zaokrąglenie do najbliższej liczby całkowitej w stronę na zewnątrz od zera
- ▶ **signbit(x)** sprawdza znak argumentu

# <time.h>

## funkcje daty i godziny

- ▶ **clock\_t clock (void)**

zwraca przybliżoną wartość czasu procesora zużytego przez program

- ▶ **char\* ctime (const time\_t \* tp)**

przekształca czas w formie podawanej przez funkcję *time()* na napis z czasem lokalnym

- ▶ **double difftime(time\_t t1, time\_t t2)**

zwraca różnicę między *t1*, a *t2* (*t1-t2*)

- ▶ **struct tm\*localtime(const time\_t \*tp)**

przekształca czas kalendarzowy zadany za pomocą wskaźnika do zmiennej zawierającej ilość sekund od 1 stycznia 1970 r. na czas lokalny w postaci wskaźnika do struktury *tm*.

- ▶ **size\_t strftime(char \*s, size\_t max, const char \*format, const struct tm \*t)**

przekształca dane o czasie zawarte w strukturze *t* na tekst, który umieszcza w tablicy *s* o ile ma on mniej niż *max* znaków.

- ▶ **time\_t time(time\_t \*tp)** odczytuje czas, który upłynął od dnia 1 stycznia 1970 roku godziny 0:00:00 czasu uniwersalnego. Liczba sekund, które upłynęły od tej chwili jest wartością funkcji.

```
struct tm {
 int tm_sec; /*sekundy od pełnej min.*/
 int tm_min; /*minuty od pełnej godz.*/
 int tm_hour; /*godzina 24h*/
 int tm_mday; /*dzień miesiąca*/
 int tm_mon; /*miesiąc (od zera)*/
 int tm_year; /*rok - 1900*/
 int tm_wday; /*dzień tyg. 0-niedziela*/
 int tm_yday; /*dzień roku (od zera)*/
 int tm_isdst; /*znacznik czasu letniego*/
};
```

# Przykład 141

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

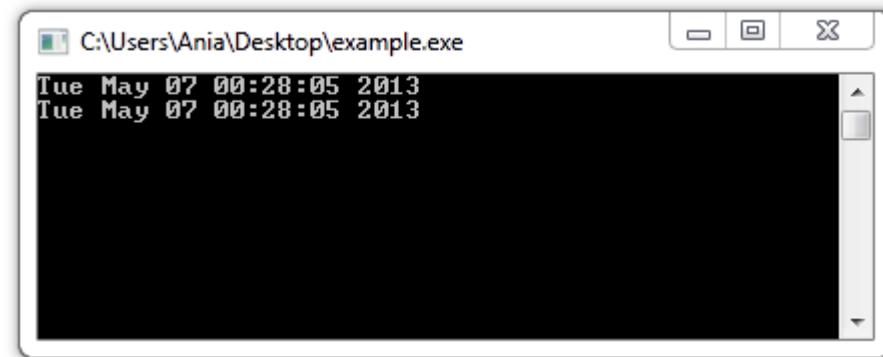
int main() {

 struct tm *wsk;
 time_t czas_lok;

 czas_lok = time(NULL) ;
 printf(ctime(&czas_lok)) ;

 wsk = localtime(&czas_lok) ;
 printf(asctime(wsk)) ;

 getchar();
 return 0;
}
```



# Przykład 142

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

int main(){

 struct tm *lokalna, *uniwersalna;
 time_t t;

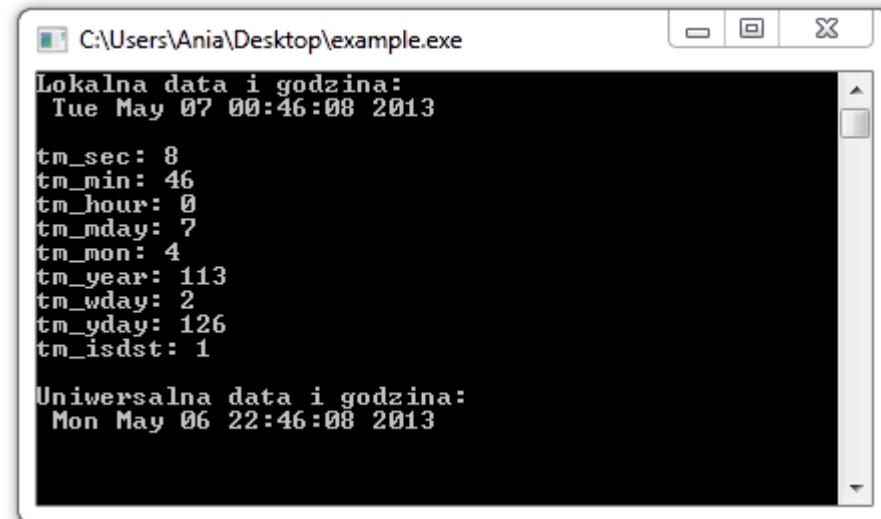
 t = time(NULL);

 lokalna = localtime(&t);
 printf("Lokalna data i godzina:\n %s\n", asctime(lokalna));

 printf("tm_sec: %d\n", lokalna->tm_sec);
 printf("tm_min: %d\n", lokalna->tm_min);
 printf("tm_hour: %d\n", lokalna->tm_hour);
 printf("tm_mday: %d\n", lokalna->tm_mday);
 printf("tm_mon: %d\n", lokalna->tm_mon);
 printf("tm_year: %d\n", lokalna->tm_year);
 printf("tm_wday: %d\n", lokalna->tm_wday);
 printf("tm_yday: %d\n", lokalna->tm_yday);
 printf("tm_isdst: %d\n", lokalna->tm_isdst);

 uniwersalna = gmtime(&t);
 printf("\nUniwersalna data i godzina:\n %s", asctime(uniwersalna));

 return 0;
}
```



```
C:\Users\Ania\Desktop\example.exe
Lokalna data i godzina:
Tue May 07 00:46:08 2013
tm_sec: 8
tm_min: 46
tm_hour: 0
tm_mday: 7
tm_mon: 4
tm_year: 113
tm_wday: 2
tm_yday: 126
tm_isdst: 1

Uniwersalna data i godzina:
Mon May 06 22:46:08 2013
```

# <stdio.h>

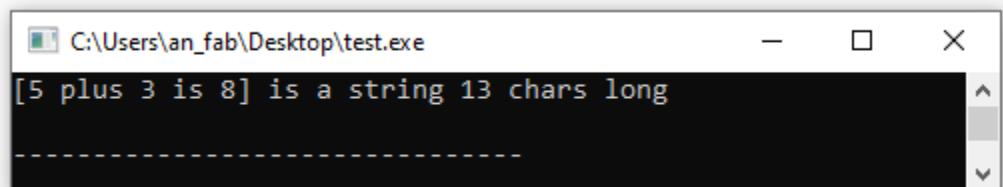
## operacje wejścia wyjścia

- ▶ Formatowanie łańcucha znakowego

```
int sprintf (char * str, const char * format, ...);
```

```
#include <stdio.h>

int main ()
{
 char buffer [50];
 int n, a=5, b=3;
 n=sprintf (buffer, "%d plus %d is %d", a, b, a+b);
 printf ("[%s] is a string %d chars long\n",buffer,n);
 return 0;
}
```



# <stdio.h>

## operacje wejścia wyjścia

- ▶ Odczytywanie danych z łańcucha znakowego w określonym formacie

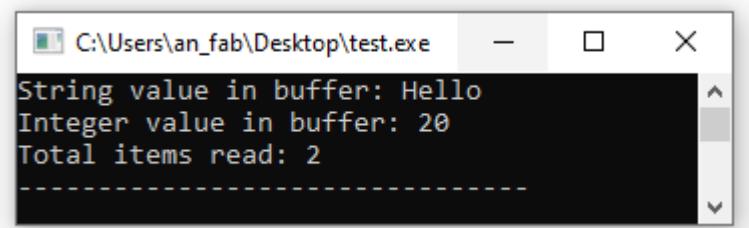
```
int sscanf (char * str, const char * format, ...);
```

```
#include<stdio.h>

int main() {
 char* buffer = "Hello 20";
 char string[10];
 int integer;
 int read;

 read = sscanf(buffer, "%s %d", string, &integer);

 printf("String value in buffer: %s", string);
 printf("\nInteger value in buffer: %d", integer);
 printf("\nTotal items read: %d", read);
 return 0;
}
```



# **<stdlib.h>**

## **funkcje narzędziowe**

---

- ▶ **abort** awaryjne zakończenie programu
- ▶ **abs** oblicza wartość bezwzględną
- ▶ **atexit** rejestracja funkcji wywoływanej po normalnym zakończeniu programu
- ▶ **atof** przekształcenie łańcucha na double
- ▶ **atoi** przekształcenie łańcucha na wartość całkowitą
- ▶ **atol** przekształcenie łańcucha na wartość całkowitą
- ▶ **bsearch** binarne przeszukiwanie posortowanej tablicy
- ▶ **calloc** przydziela i zwalnia pamięć dynamiczną
- ▶ **div** obliczanie ilorazu oraz reszty dzielenia liczb całkowitych
- ▶ **exit** normalne zakończenie programu
- ▶ **free** zwalnia pamięć dynamiczną
- ▶ **getenv** odczytanie zmiennej środowiska

# <stdlib.h>

## funkcje narzędziowe

---

- ▶ **labs** oblicza wartość bezwzględną
- ▶ **malloc** przydziela pamięć dynamiczną
- ▶ **mbtowc** przekształca ciąg wielobajtowy na znak szeroki
- ▶ **qsort** sortuje tablicę
- ▶ **rand** generator liczb losowych
- ▶ **realloc** przydziela i zwalnia pamięć dynamiczną
- ▶ **srand** generator liczb losowych
- ▶ **strtod** zamienia łańcuch znakowy na liczbę zmiennoprzecinkową
- ▶ **strtol** zamienia łańcuch znakowy na liczbę całkowitą długą
- ▶ **strtoul** zamienia łańcuch znakowy na liczbę całkowitą długą bez znaku
- ▶ **system** wywołanie polecenia w powłoce
- ▶ **wcstombs** konwertuje unikodowy łańcuch znaków na wielobajtowy łańcuch znaków

# <stdlib.h>

## Wyszukiwanie binarne bsearch()

```
void * bsearch(const void * klucz, const void * tab, size_t liczbaElementow,
size_t roz, int(* porownaj)(const void *, const void *));
```

- ▶ funkcja *bsearch* dokonuje wyszukiwanie binarne w posortowanej tablicy wskazywanej przez *tab* i zwraca wskaźnik do pierwszego napotkanego elementu zgodnego z kluczem wyszukiwania wskazanym przez *key*; liczba elementów w tablicy jest określona przez *liczbaElementów* a rozmiar pojedynczego elementu w bajtach wynosi *roz*
- ▶ funkcja wskazywana przez *porownaj* służy do porównywania elementów tablicy, jej postać jest następująca:

```
int nazwa-funkcji (const void *arg1, const void *arg2);
```

- ▶ funkcja musi zwracać wartość mniejszą niż 0, gdy *arg1* jest większe od *arg2*, 0 gdy *arg1* oraz *arg2* są równe oraz wartość większą od 0 jeżeli *arg1* jest większe od *arg2*
- ▶ tablica musi być posortowana w kierunku rosnącym

# Przykład 143

```
#include <stdio.h>
#include <ctype.h>
#include <stdlib.h>

char *alfabet = "abcdefghijklmnopqrstuvwxyz";

int por(const void *ch, const void *s);

int main(void){

 char ch, *p;

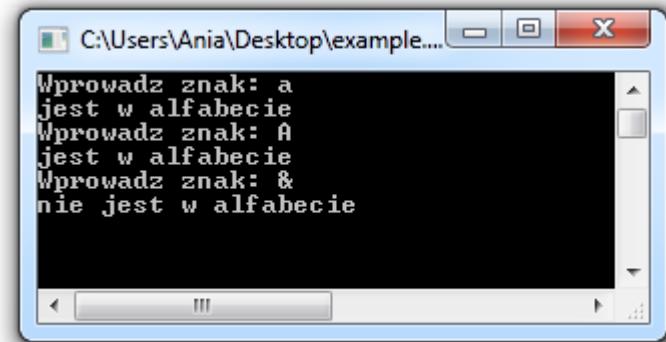
 do{
 printf("Wprowadz znak: ");
 scanf("%c%c", &ch);
 ch = tolower(ch);
 p=bsearch(&ch, alfabet, 26, 1, por);

 if(p) printf("jest w alfabecie\n");
 else printf("nie jest w alfabecie\n");
 }while(p);

 return 0;
}

int por(const void *ch, const void *s){

 return *(char*)ch - *(char *)s;
}
```



## <stdlib.h>

### Funkcje `exit()` i `atexit()`

---

- ▶ Funkcja `exit()` powoduje automatyczne zakończenie programu
- ▶ Funkcja `atexit()` pozwala określić funkcje, które powinny zostać wywołane razem z funkcją `exit()` przy zakończeniu programu
  - ▶ należy jej przekazać adres funkcji
  - ▶ standard ANSI gwarantuje, że lista funkcji wywoływanych razem z `exit()` może zmieścić przynajmniej 32 funkcje – każda z tych funkcji musi zostać zarejestrowana w oddzielnym wywołaniu `atexit()`
  - ▶ ostatnia zarejestrowana funkcja jest wywoływana jako pierwsza
  - ▶ wszystkie funkcje powinny być typu `void` i nie mieć argumentów

# Przykład 144

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

void koniec(void);
void wpadka(void);

int main(void){

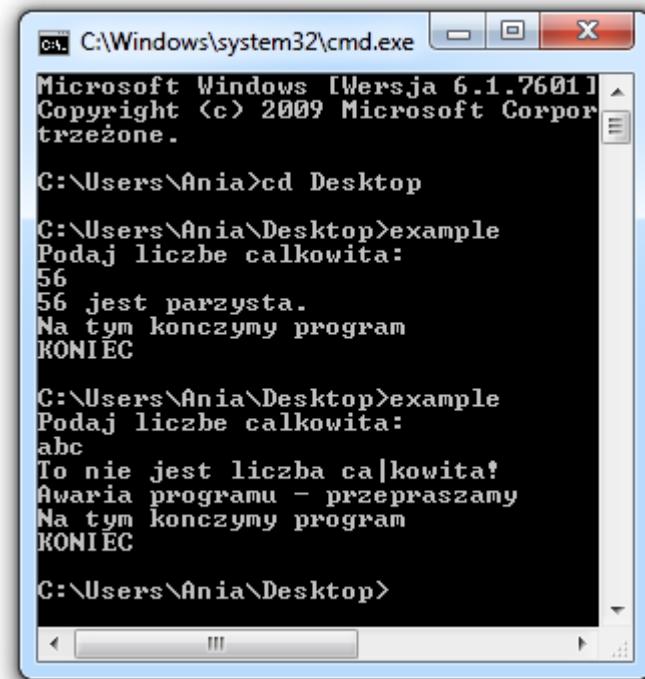
 int n;
 atexit(koniec);

 puts("Podaj liczbe całkowita: ");
 if(scanf("%d", &n) != 1){
 puts("To nie jest liczba całkowita!");
 atexit(wpadka);
 exit(EXIT_FAILURE); //ew. EXIT_SUCCESS
 }

 printf("%d jest %s.\n", n, (n%2 == 0) ? "parzysta" : "nieparzysta");
 return 0;
}

void koniec(void){
 puts("Na tym konczymy program");
 puts("KONIEC");
}

void wpadka(void){
 puts("Awaria programu - przepraszamy");
}
```



The screenshot shows a Windows Command Prompt window titled 'C:\Windows\system32\cmd.exe'. The window displays the following text:

```
Microsoft Windows [Wersja 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. Wszelkie prawa zastrzeżone.

C:\Users\Ania>cd Desktop
C:\Users\Ania\Desktop>example
Podaj liczbe całkowita:
56
56 jest parzysta.
Na tym konczymy program
KONIEC

C:\Users\Ania\Desktop>example
Podaj liczbe całkowita:
abc
To nie jest liczba całkowita!
Awaria programu - przepraszamy
Na tym konczymy program
KONIEC

C:\Users\Ania\Desktop>
```

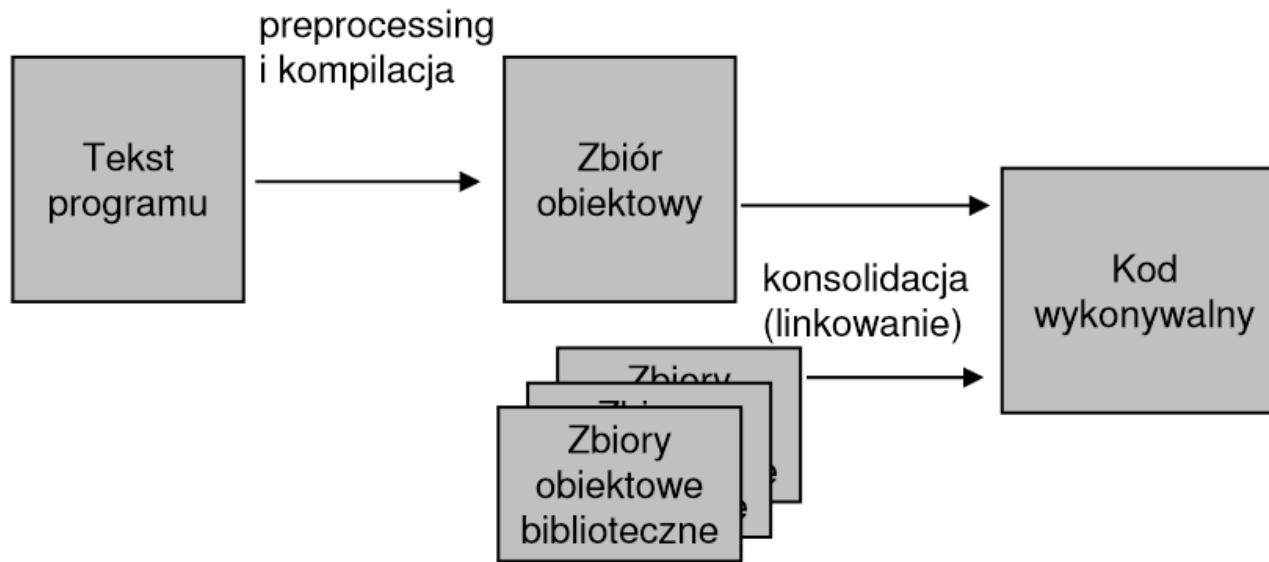
# Programowanie hybrydowe

---

- ▶ **Programowanie hybrydowe** (ang. *hybrid programming*) polega na tworzeniu programu, którego moduły napisane są w różnych językach programowania.
- ▶ **Zastosowania:**
  - ▶ używanie gotowych modułów, ze starszych programów w nowym oprogramowaniu, pisany w innym języku;
  - ▶ realizacja w języku niższego poziomu operacji niedostępnych lub nieefektywnych w języku wysokiego poziomu (np. język C łączony z asemblerem, języki wyższych generacji łączone z językiem C);
  - ▶ korzystanie z modułów bibliotecznych;
  - ▶ łączenie pracy programistów specjalizujących się w różnych językach programowania.

# Tłumaczenie kodu programu

- ▶ Tłumaczenie kodu programu napisanego w języku wysokiego poziomu



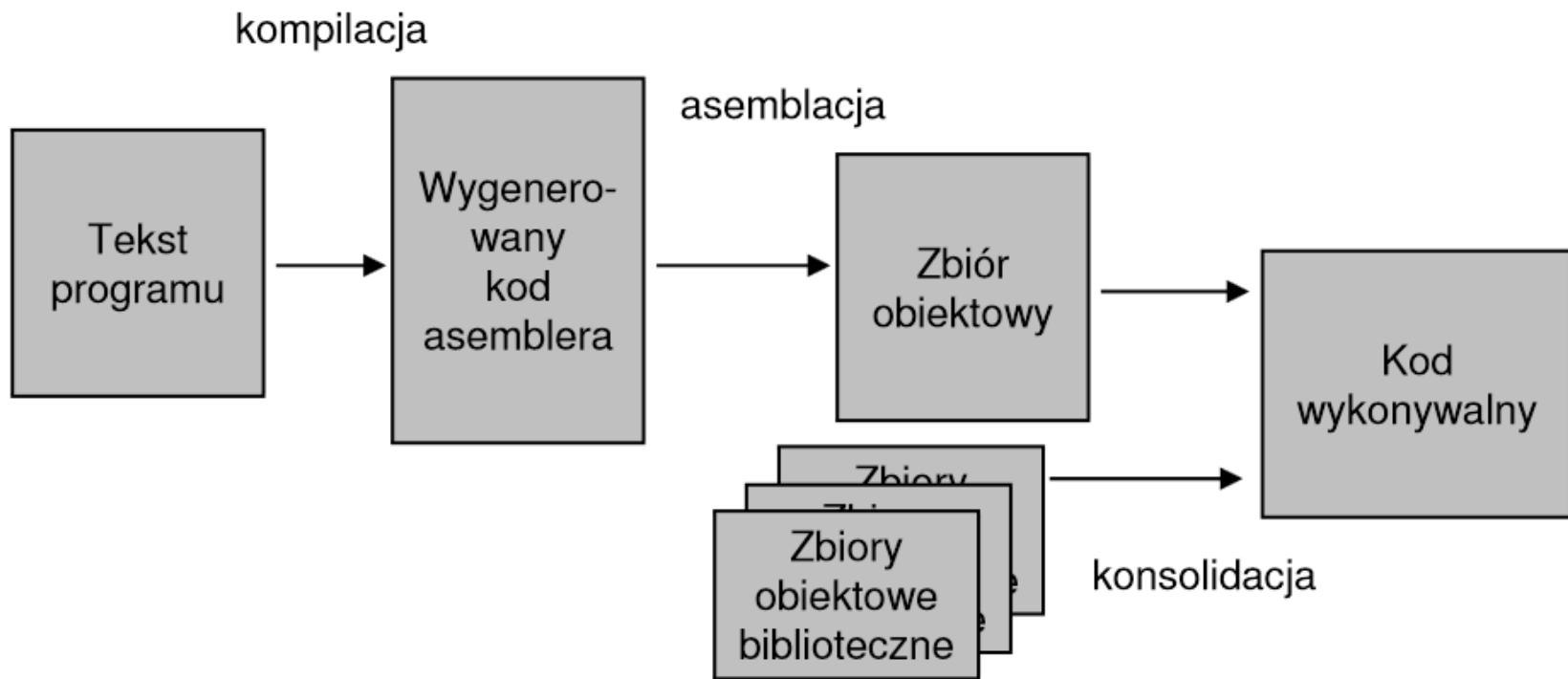
## Zbiory obiektowe:

- zawierają kod wykonywalny, w którym nie zostały zdefiniowane odwołania do obiektów zewnętrznych: zmiennych i procedur definiowanych w innych modułach;
- zawierają informację o tych odwołaniach i udostępnianych przez siebie obiektach.

Dzięki ww. informacjom konsolidator może wiązać obiekty z odwołaniami w innych modułach.

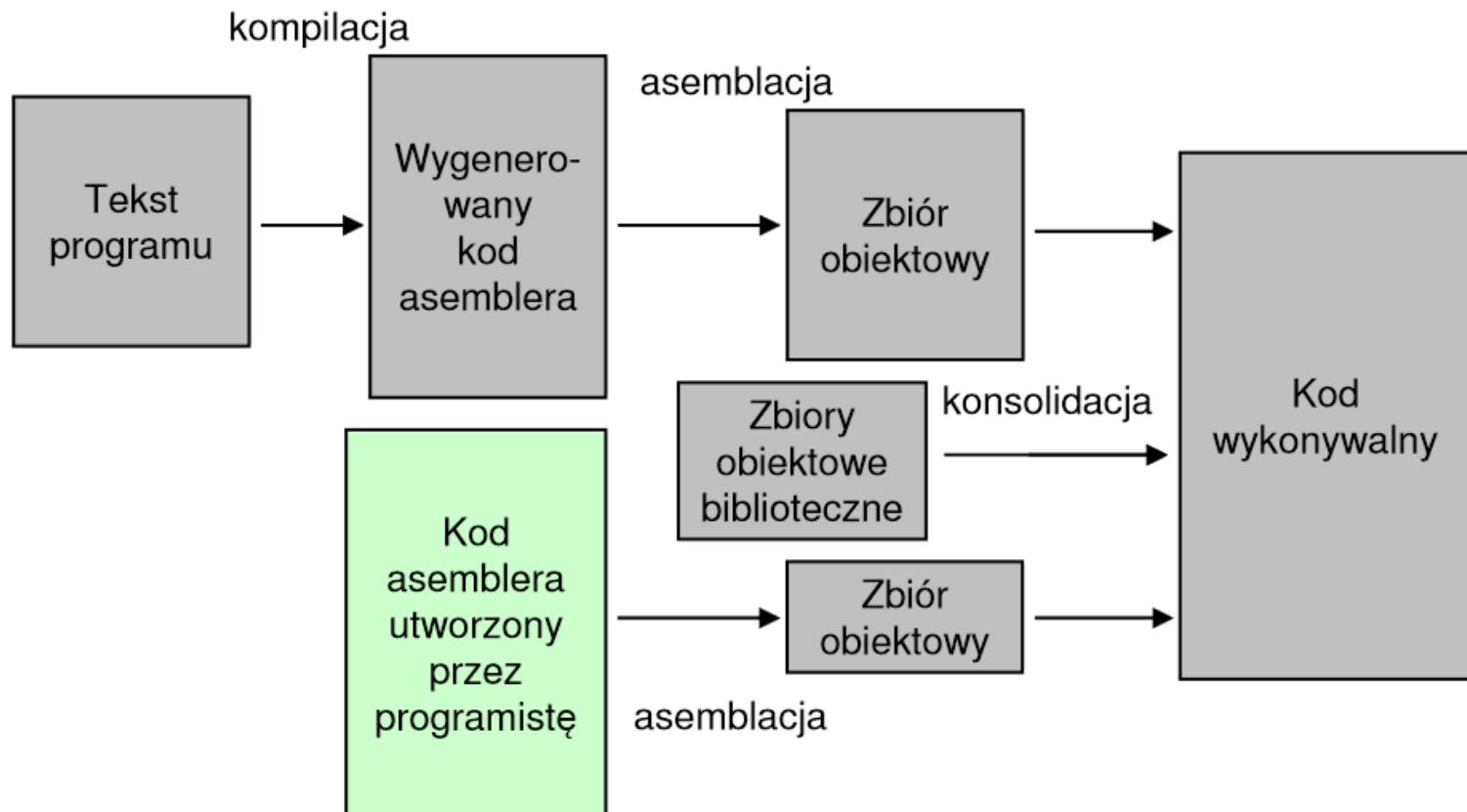
# Tłumaczenie kodu programu

- ▶ Tłumaczenie kodu programu – wersja z asemblacją



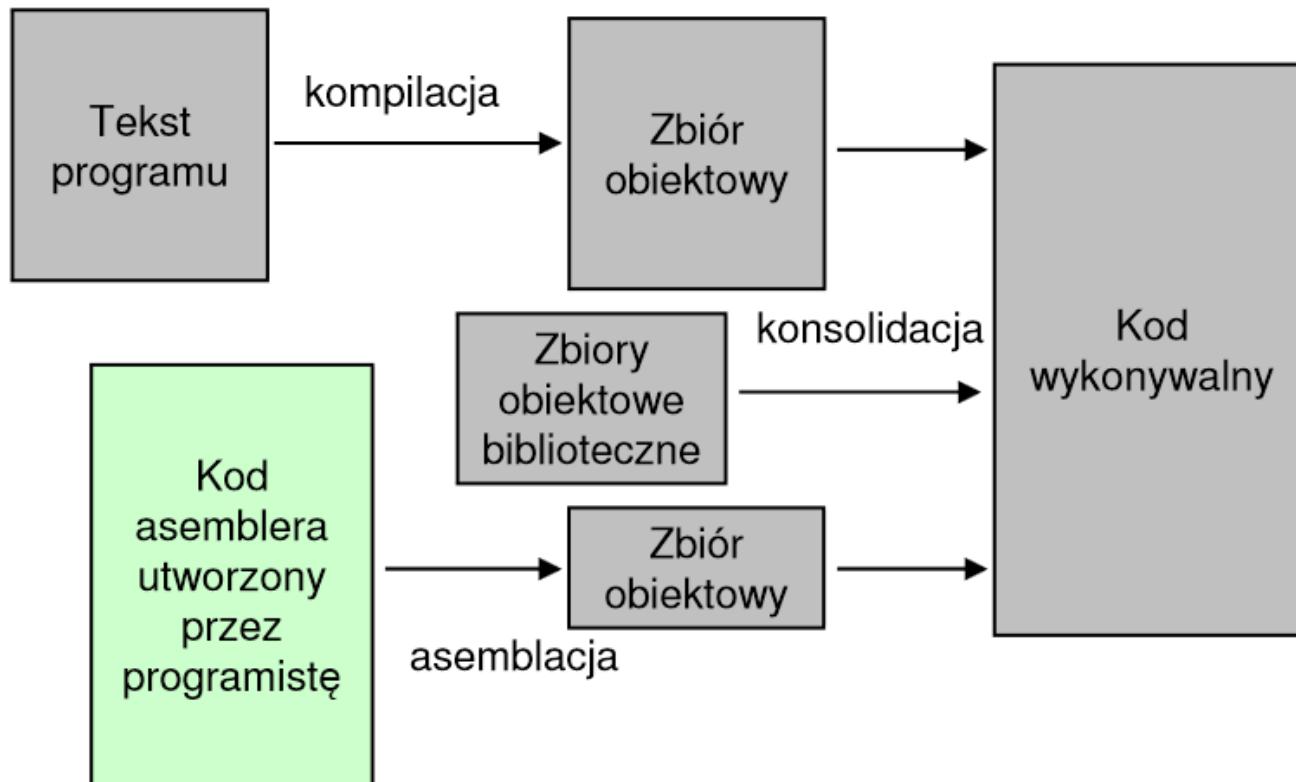
# Tłumaczenie kodu programu

- ▶ Łączenie programowania w języku wysokiego poziomu i w asemblerze (programowanie hybrydowe)



# Tłumaczenie kodu programu

- ▶ Łączenie programowania w języku wysokiego poziomu i w asemblerze (programowanie hybrydowe) – z pominięciem postaci asemblerowej



# Wstawki asemblerowe w języku C

- ▶ Niektóre kompilatory języka C posiadają wbudowany asembler, co pozwala na bezpośrednie dołączanie kodu asemblerowego do treści funkcji w języku C.
- ▶ Technika ta, określana terminem wstawki asemblerowe, może być zalecana wówczas, gdy długość dołączanego kodu asemblerowego nie przekracza kilkudziesięciu linii.

Jedna instrukcja:

```
asm <instrukcja języka asembler>;
```

Grupa instrukcji:

```
asm {
 <instrukcja języka asembler>;
 ...
 <instrukcja języka asembler>;
};
```

```
int jakas_funkcja (int n)
{
 ...
etykleta1:
 asm mov cx, n;
 asm mov [si], 0;
 asm inc si;
 asm loop etykleta1;
 ...
 return 0;
};
```

# Wstawki assemblerowe w języku C

## ▶ Albo: funkcja **asm()**

- ▶ pojedyncze instrukcje umieszcza się w cudzysłowie i kończy znakiem końca linii: "\n"
- ▶ pierwszej linii funkcji **asm()** określa się typ składni dla instrukcji procesora.

```
asm(".intel_syntax \n" // rodzaj składni
 "MOV ESI, [EBP+8] \n" // wskaznik
 "ADD ESI, 4 \n" // wskaznik+1
 "MOV EAX, [ESI] \n" // *(wskaznik+1)
...
);
```

# Przykład łączenia kodów C i ASM

## Program w języku assembler:

```
.text
.global f1
f1:
pushl %ebp
movl %esp, %ebp
movl $4, %eax
movl $1, %ebx
movl $tekst, %ecx
movl $len, %edx
int $0x80
popl %ebp
Ret
.data
tekst:
.string "Hello
world\n"
len = . - tekst
```

## Program główny w języku C:

```
extern void f1 (void);

int main () {
 f1();
 return 0;
}
```

# Przykład łączenia kodów C i ASM

Program w języku asembler:

```
.text
.global _funkcja
_funkcja:
 pushl %ebp
 movl %esp, %ebp
 movl 8(%esp), %eax /* kopiujemy pierwszy argument do %eax */
 addl 12(%esp), %eax /* do pierwszego argumentu w %eax dodajemy
 drugi argument */
 popl %ebp
 ret
/* ... i zwracamy wynik dodawania... */
```

Program główny w języku C:

Program główny w języku C:

```
#include <stdio.h>
extern int funkcja (int a, int b);
int main () {
printf ("2+3=%d\n", funkcja(2,3));
return 0;
}
```

# Program główny w języku C

//Plik: driver.c

```
#include "cdecl.h"

int PRE_CDECL asm_main(void) POST_CDECL;

int main() {
 int ret_status;
 ret_status = asm_main();
 return ret_status;
}
```

wywołanie funkcji  
napisanej w  
assemblerze

# Struktura podprogramu w assemblerze (wywołanego z C)

---

```
;Plik: first.asm
%include "asm_io.inc"
segment .data
prompt1 db "Wprowadź liczbę: ", 0
...
segment .bss ; dane niezainicjalizowane
input1 resd 1
...
segment .text
global _asm_main
_asm_main:
 enter 0,0
 pusha
 ...
 popa
 mov eax,0
 leave
 ret
```

# Przykład łączenia kodów C i MATLAB

- ▶ Przykład I: wyświetlanie napisu z pliku mex

```
#include <stdio.h>
#include "mex.h"

void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray
*prhs[])
{
 printf("Hello World!\n");
}

//kompilacja mex hello_mex.c
//wynik hello_mex.mexw64 albo hello_mex.mexw32
```

hello\_mex.c

```
function run_mex plik.m
 hello_mex;
end
```

# Przykład łączenia kodów C i MATLAB

## ▶ Przykład 2: pobieraniełańcucha do pliku mex

```
#include <stdio.h>
#include "mex.h"
#include "matrix.h"

void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[]) {

 char *buf;
 int buflen;
 int status;

 buflen = (mxGetM(prhs[0]) * mxGetN(prhs[0])) + 1;
 buf = malloc(buflen);
 status = mxGetString(prhs[0], buf, buflen);
 printf("%s\n",buf);
 free(buf);
}
//kompilacja mex hello2_mex.c
//wynik hello2_mex.mexw64 albo hello2_mex.mexw32
```

hello2\_mex.c

```
function run_mex2
 hello2_mex('Hello World second time');
end
```

plik.m

# Przykład łączenia kodów C i MATLAB

## ▶ Przykład 3: pobieranie i działania na tablicy w pliku mex

```
#include <stdio.h>
#include "mex.h"

void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[]) {

 double *xData, avg, fac;
 int i, j, rows, cols;
 //pobranie tablicy danych xData[M x N]
 xData = (double *)mxGetPr(prhs[0]);
 rows = mxGetM(prhs[0]);
 cols = mxGetN(prhs[0]);
 //pobieranie danej skalarnej
 fac=(double)mxGetScalar(prhs[1]);
 //obliczenie średnich w wierszach tablicy ze skalą fac
 //tablica MATLAB 2D ułożona kolumnami w pamięci operacyjnej
 for(j=0;j<cols;j++) {
 avg=0;
 for(i=0;i<rows;i++)
 avg += xData[j*rows+i];
 avg = fac*avg/cols;
 printf("Srednia skalowana w kolumnie %d, is %.4g",j,avg);
 }
}
```

avg\_mex.c

# Przykład łączenia kodów C i MATLAB

- ▶ Przykład 3: pobieranie i działania na tablicy w pliku mex

```
function run_mex3
 data=rand(512,512);
 fac=0.7;
 avg_mex(data,fac);
end
```

plik.m

# Przykład łączenia kodów C i MATLAB

## ▶ Przykład 4: zwracanie zmiennych do środowiska MATLAB

```
//plik "array_mex.c", wynik array_mex.mexw64 albo array_mex.mexw32
#include <stdio.h>
#include "mex.h"

void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[]){
 double fac, *xData, *outArray;
 int i,j, rows, cols;
 //pobranie tablicy danych xData[M x N]
 xData = (double *)mxGetPr(prhs[0]);
 rows = mxGetM(prhs[0]);
 cols = mxGetN(prhs[0]);
 //pobieranie danej skalarnej jako 2 parametru funkcji
 fac=(double)mxGetScalar(prhs[1]);
 //alokuj pamięć dla MATLAB
 plhs[0] = mxCreateDoubleMatrix(rows, cols, mxREAL);
 //mxReal - liczby rzeczywiste double
 //odczytaj wskaźnik do alokowanej pamięci
 outArray = mxGetPr(plhs[0]);
 //przeskaluj wartości tablicy wejściowej
 for(j=0;j<cols;j++){
 for(i=0;i<rows;i++){
 outArray[j*rows+i] = fac*xData[j*rows+i];
 }
 }
} //kompilacja mex array_mex.c
```

array\_mex.c

# Przykład łączenia kodów C i MATLAB

- ▶ Przykład 4: wracanie zmiennych do środowiska MATLAB

```
function run_mex4
 data=rand(512,512);
 fac=0.7;
 data2=array_mex(data,fac);
endkompilacja mex array_mex.c
```

# Przykład łączenia kodów C i MATLAB

## ▶ Przykład 5: korzystanie z funkcji MATLAB w C (inwersja macierzy)

```
//plik "inv_mex.c",
//kompilacja mex inv_mex.c, wynik inv_mex.mexw64 albo inv_mex.mexw32

#include <stdio.h>
#include "mex.h"

void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[]){
 mxArray *lhs[1]={NULL};
 double *xData;

 if(nrhs!=1 || nlhs!=1 || !mxIsDouble(prhs[0])){
 mexErrMsgTxt("Niepoprawne argumenty funkcji");
 }
 xData = (double *)mxGetPr(prhs[0]);
 rows = mxGetM(prhs[0]);
 cols = mxGetN(prhs[0]);

 if(rows!=cols){
 mexErrMsgTxt("Macierz nie jest kwadratowa");
 }
 else{ //wywołanie funkcji inv(xData) ze środowiska MATLAB
 mexCallMATLAB(1, lhs, 1, prhs, "inv");
 }
 plhs[0]=lhs[0];
```

# Przykład łączenia kodów C i MATLAB

- ▶ Przykład 5: korzystanie z funkcji MATLAB w C (inwersja macierzy)

```
function run_mex5
 data=eye(16) ;//
 data2=inv_mex(data) ;
 data
end
```

# Przykład łączenia kodów C i Python

## ▶ Wywoływanie funkcji C w języku python

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

```
int add(int a, int b) {
 return a + b;
}
```

```
int multiply(int a, int b) {
 return a * b;
}
```

example.c

Kompilacja kodu C do biblioteki współdzielonej:

```
gcc -shared -o
example.so example.c
```



```
import ctypes

Załaduj bibliotekę C
lib = ctypes.CDLL('./example.so')

Ustaw typy argumentów dla funkcji add
lib.add.argtypes = [ctypes.c_int, ctypes.c_int]

Ustaw typ wyniku dla funkcji add
lib.add.restype = ctypes.c_int

Wywołaj funkcję add
result = lib.add(5, 3)
print(f"Wynik dodawania: {result}")

Ustaw typy argumentów dla funkcji multiply
lib.multiply.argtypes = [ctypes.c_int, ctypes.c_int]

Ustaw typ wyniku dla funkcji multiply
lib.multiply.restype = ctypes.c_int

Wywołaj funkcję multiply
result = lib.multiply(5, 3)
print(f"Wynik mnożenia: {result}")
```

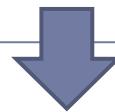
example.py

# Przykład łączenia kodów C i Python

## ▶ Przekazywanie tablic z Pythona do C (I)

```
#include <stdio.h> array_sum.c

int array_sum(int* array, int length) {
 int sum = 0;
 for (int i = 0; i < length; i++) {
 sum += array[i];
 }
 return sum;
}
```



Kompilacja kodu C do biblioteki współdzielonej:

```
gcc -shared -o array_sum.so array_sum.c
```

# Przykład łączenia kodów C i Python

## ▶ Przekazywanie tablic z Pythona do C (2)

```
import ctypes

Załóż bibliotekę C
lib = ctypes.CDLL('./array_sum.so')

Utwórz tablicę w Pythonie
array = [1, 2, 3, 4, 5]
length = len(array)

Utwórz tablicę C typu int
c_array = (ctypes.c_int * length)(*array)

Ustaw typy argumentów dla funkcji array_sum
lib.array_sum.argtypes = [ctypes.POINTER(ctypes.c_int), ctypes.c_int]

Ustaw typ wyniku dla funkcji array_sum
lib.array_sum.restype = ctypes.c_int

Wywołaj funkcję array_sum
result = lib.array_sum(c_array, length)
print(f"Suma elementów tablicy: {result}")
```

array\_sum.py

# Przykład łączenia kodów C i Python

---

- ▶ Wywoływanie funkcji Python w języku C

script.py

```
def greet(name):
 return f"Hello, {name}!"
```

```
#include <Python.h>

int main() {
 Py_Initialize(); // Initialize the Python interpreter

 // Add the current directory to Python's sys.path
 PyRun_SimpleString("import sys");
 PyRun_SimpleString("sys.path.append('./')");

 // Load the Python script
 PyObject *pName = PyUnicode_DecodeFSDefault("script");
 PyObject *pModule = PyImport_Import(pName);

 // Get the Python function
 PyObject *pFunc = PyObject_GetAttrString(pModule, "greet");

 // Create a tuple to hold the arguments for the Python function
 PyObject *pArgs = PyTuple_New(1);

 // Add the string "World" to the tuple
 PyTuple_SetItem(pArgs, 0, PyUnicode_FromString("World"));

 // Call the Python function and get the result
 PyObject *pValue = PyObject_CallObject(pFunc, pArgs);

 // Print the result
 printf("Greeted: %s\n", PyUnicode_AsUTF8(pValue));

 Py_FinalizeEx(); // Finalize the Python interpreter
 return 0;
}
```

# Przykład łączenia kodów C i Python

## ▶ Wywoływanie funkcji Python w języku C

```
def sum(a, b):
 return a + b
```

script.py

```
#include <Python.h>

int main() {
 Py_Initialize(); // Initialize the Python interpreter
 // Add the current directory to Python's sys.path
 PyRun_SimpleString("import sys");
 PyRun_SimpleString("sys.path.append('..')");

 // Load the Python script
 PyObject *pName = PyUnicode_DecodeFSDefault("script");
 PyObject *pModule = PyImport_Import(pName);

 // Get the Python function
 PyObject *pFunc = PyObject_GetAttrString(pModule, "sum");

 // Create a tuple to hold the arguments for the Python function
 PyObject *pArgs = PyTuple_New(2);
 PyTuple_SetItem(pArgs, 0, PyLong_FromLong(3)); // Add the integer 3 to the tuple
 PyTuple_SetItem(pArgs, 1, PyLong_FromLong(4)); // Add the integer 4 to the tuple

 // Call the Python function and get the result
 PyObject *pValue = PyObject_CallObject(pFunc, pArgs);

 // Print the result
 printf("Sum: %ld\n", PyLong_AsLong(pValue));

 Py_FinalizeEx(); // Finalize the Python interpreter
 return 0;
}
```

# Łączenie kodów C i Java

---

- ▶ **Java Native Interface (JNI)** - technologia pozwalająca na wywoływanie z poziomu Javy funkcji zaimplementowanych w C/C++.
- ▶ **Podstawowe etapy:**
  1. przygotowanie kodu źródłowego;
  2. komplikacja;
  3. generowanie pliku nagłówkowego;
  4. implementacja metody natywnej;
  5. utworzenie biblioteki dynamicznej;
  6. uruchomienie programu.

Łączenie Javy i C++ nie odbywa się na poziomie leksykalnym - poprzez fizyczne wstawianie kodu programu, lecz na poziomie kodu wykonywanego - poprzez dynamiczne łączenie skompilowanego kodu w trakcie wykonania.

# Łączenie kodów C i Java

- ▶ Przygotowanie kodu źródłowego

```
class HelloWorld{
 native public void sayHello();

 public static void main(String[] args) {
 new HelloWorld().sayHello();
 }
}
```

HelloWorld.java

Informacja dla kompilatora, że ma do czynienia z metodą natywną - taką, której implementacja została wykonana w innym języku programowania i zostanie dostarczona w osobnej bibliotece ładowanej dynamicznie.

# Łączenie kodów C i Java

- ▶ Kompilacja

```
javac HelloWorld.java
```



```
Exception in thread "main"
java.lang.UnsatisfiedLinkError: sayHello
 at HelloWorld.sayHello(Native Method)
 at HelloWorld.main(HelloWorld.java:6)
```

- ▶ Generowanie pliku nagłówkowego

# Łączenie kodów C i Java

## ▶ Generowanie pliku nagłówkowego

`javah -jni` → `HelloWorld.h`

```
/* DO NOT EDIT THIS FILE - it is machine generated */
#include <jni.h>
/* Header for class HelloWorld */
```

```
#ifndef _Included_HelloWorld
#define _Included_HelloWorld
#ifndef __cplusplus
extern "C" {
#endif
```

```
/*
 * Class: HelloWorld
 * Method: sayHello
 * Signature: ()V
 */
```

```
JNIEXPORT void JNICALL Java_HelloWorld_sayHello(JNIEnv *, jobject);
```

```
#ifdef __cplusplus
}
#endif
#endif
```

Makra potrzebne do wyeksportowania funkcji z biblioteki dynamicznej i zapewnienia odpowiedniego protokołu wywołania

Standardowe argumenty przekazywane wszystkim funkcjom implementującym metody natywne

`HelloWorld.h`

# Łączenie kodów C i Java

- ▶ Implementacja metody natywnej

```
JNIEXPORT void JNICALL Java_HelloWorld_sayHello(JNIEnv *, jobject);
```

```
#include <jni.h>
```

Plik nagłówkowy, który zawiera deklaracje funkcji i typów JNI. Musi on być włączany przez każdy moduł implementujący funkcje natywne.

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

```
#include "HelloWorld.h"
```

Wcześniej wygenerowany plik nagłówkowy.

```
JNIEXPORT void JNICALL
Java_HelloWorld_sayHello(JNIEnv * env, jobject self)
{
 printf("Hello World!\n");
}
```

HelloWorld.c

# Łączenie kodów C i Java

- ▶ Utworzenie biblioteki dynamicznej:
  - ▶ Należy skompilować plik z implementacją `HelloWorld.cpp` i utworzyć bibliotekę dynamiczną.
  - ▶ Sposób realizacji tego kroku jest zależny od kompilatora i systemu operacyjnego.

## Linux/GNU gcc

```
c++ -I:j2sdk/include -I:j2sdk/include/linux -shared -o libHelloWorld.so HelloWorld.c
```

## Win32/Borland bcc32

```
bcc32 -IC:\j2sdk\include -IC:\j2sdk\include\win32 -tWD -eHelloWorld.dll HelloWorld.c
```

## Win32/Microsoft VC++

```
cl /IC:\j2sdk\include /IC:\j2sdk\include\win32 /GX /LD HelloWorld.cpp /FeHelloWorld.dll
```

# Łączenie kodów C i Java

- ▶ Uruchomienie programu
  - ▶ żądanie załadowania przez VM przygotowanej biblioteki.

```
static {
 System.loadLibrary("HelloWorld");
}
```

- ▶ poinformowanie SO gdzie biblioteki należy szukać (ustalając ścieżkę środowiskową)

```
bash, sh, ksh
 export LD_LIBRARY_PATH=lib

tcsh, csh
 setenv LD_LIBRARY_PATH lib

ms-dos, cmd.exe dla Win32
 set PATH=%path%;lib
```

# Wywoływanie kodu Java z programów C

```
public class JNIHelloWorld2 {
 int i = 0;

 public JNIHelloWorld2() {
 this.i = 10;
 }

 public static void napisz(int n) {
 System.out.println("Hello C World number "+n);
 }

 public static void napisz2(String tekst) {
 System.out.println(tekst);
 }

 public static double pomnoz(double a, double b) {
 return a*b;
 }

 public static void pokazWiadomosc() {
 javax.swing.JOptionPane.showMessageDialog(null, "Hello C World!");
 }

 public void dynamiczna() {
 System.out.println("i="+i);
 }
}
```

## Krok I – kod w języku Java

plik **JNIHelloWorld2.java**

# Wywoływanie kodu Java z programów C

## Krok 2 – komplikacja klasy Java

```
javac JNIHelloWorld2.java → JNIHelloWorld2.class
```

# Wywoływanie kodu Java z programów C

## Krok 3 – kod programu natywnego C/C++

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

#ifndef _WIN32
#define PATH_SEPARATOR ';'
#else
#define PATH_SEPARATOR ':'
#endif

int main() {
 JavaVMOption options[1];
 JNIEnv *env;
 JavaVM *jvm;
 JavaVMInitArgs vm_args;
 int status;

 jclass cls;
 jmethodID mid;
 jdouble mult;
```

1.

plik: **JNIHelloWorld2.c**

```
options[0].optionString = "-Djava.class.path=. ";
memset(&vm_args, 0, sizeof(vm_args));
vm_args.version = JNI_VERSION_1_2;
vm_args.nOptions = 1;
vm_args.options = options;
```

2.

# Wywoływanie kodu Java z programów C

```
status = JNI_CreateJavaVM(&jvm, (void**)&env, &vm_args);
```

3.

```
if (status != JNI_ERR) {
 printf("JVM successfully started\n\n");
 cls = (*env)->FindClass(env, "JNIHelloWorld2");
```

```
if (cls != 0) {
 mid = (*env)->GetStaticMethodID(env, cls, "napisz", "(I)V");
 if (mid != 0)
 (*env)->CallStaticVoidMethod(env, cls, mid, 5);
 else
 printf("Error loading method.\n");
```

```
mid = (*env)->GetStaticMethodID(env, cls, "napisz2", "(Ljava/lang/String;)V");
if (mid != 0) {
 jstring str = (*env)->NewStringUTF(env, "Hello C World!");
 (*env)->CallStaticVoidMethod(env, cls, mid, str);
} else
 printf("Error loading method.\n");
```

```
mid = (*env)->GetStaticMethodID(env, cls, "pomnoz", "(DD)D");
if (mid != 0) {
 mult = (*env)->CallStaticDoubleMethod(env, cls, mid, 2.0, 5.0);
 printf("2*5=%f\n", mult);
} else
 printf("Error loading method.\n");
```

plik: **JNIHelloWorld2.c**

# Wywoływanie kodu Java z programów C

```
mid = (*env)->GetStaticMethodID(env, cls, "pokazWiadomosc", "()V");
if(mid != 0)
 (*env)->CallStaticVoidMethod(env, cls, mid);
else printf("Error loading method.\n");
```

3.

plik: **JNIHelloWorld2.c**

```
mid = (*env)->GetMethodID(env, cls, "<init>", "()V");
if(mid != 0)
 jobject obj = (*env)->NewObject(env, cls, mid);

mid = (*env)->GetMethodID(env, cls, "dynamiczna", "()V");
if(mid != 0)
 (*env)->CallVoidMethod(env, obj, mid);
else printf("Error loading method.\n");
} else {
 printf("Error loading constructor method.\n");
}
} else printf("Error loading Java class.\n");
}

(*jvm)->DestroyJavaVM(jvm);
 return 0;
} else {
 printf("Error starting JVM\n");
 return -1;
}
```

---

► Koniec

# Wskaźniki

## Wskaźnik (ang. *pointer*)

- zmienna zawierająca adres innej zmiennej;
- może obejmować kilka komórek pamięci (w zależności od typu zmiennej na którą wskazuje).

WSK – wskaźnik

**WSK = 2460**

wskaźnik WSK został ustawiony tak, aby wskazywał na komórkę pamięci o adresie 2460

Zawartość komórki wskazywanej przez wskaźnik WSK: **11100101**

Adresy komórek w pamięci są wartościami całkowitymi, z tego też powodu wskaźniki przyjmują wyłącznie wartości całkowite.

| ADRESY | KOMÓRKI PAMIĘCI |
|--------|-----------------|
| ...    | ...             |
| 2453   | 01001100        |
| 2454   | 00000010        |
| 2455   | 00000000        |
| 2456   | 10110110        |
| 2457   | 01000010        |
| 2458   | 00000001        |
| 2459   | 00000000        |
| 2460   | 11100101        |
| 2461   | 00001011        |
| 2462   | 10110010        |
| ...    | ...             |

# Deklaracja wskaźników

**TYP \*nazwa\_zmiennej\_wskaznikowej;**

- ▶ TYP – typ podstawowy wskaźnika; może być typem podstawowym języka C, lub też typem złożonym (np. strukturą).

```
int *pi; /*pi jest wskaźnikiem do zmiennej całkowitej*/
char *pc; /*pc jest wskaźnikiem do zmiennej znakowej */
float *pf; /*pf jest wskaźnikiem do zmiennej typu float*/
```

- ▶ Typ wskaźnika, musi być zgodny z typem zmiennej, na którą wskazuje.

# Ustawianie wskaźników

**TYP zmienna = wartość;**

**TYP \*wskaźnik = &zmienna;**

- ▶ Dopóki wskaźnik nie zostanie ustawiony (tzn. nie zostanie mu przypisany adres zmiennej), wskaźnik jest pusty (na nic nie wskazuje).
- ▶ Operator & (ampersand) zwraca adres zmiennej, którą poprzedza.

```
int i = 10;
int *pi = &i;
```

Ustawianie wskaźnika  
na etapie deklaracji

```
float f = 3.14;
float *pf;
pf = &f;
```

Ustawianie wskaźnika  
po deklaracji

# Operator wyłuskania

**TYP zmienna = wartość;**

**TYP \*wskaźnik = &zmienna;**

**TYP wart = \*wskaźnik;**

- ▶ Operator **&** (ampersand) zwraca adres zmiennej, którą poprzedza.
- ▶ Operator **\*** (operator wyłuskania, pośredniości, dereferencji) zwraca wartość zmiennej przechowywanej pod adresem, który poprzedza.

```
int x=1, *wsk;
wsk = &x
```



**x ⇌ \*wsk**  
**wsk ⇌ &x**

# Przykład 1

```
#include <stdio.h>

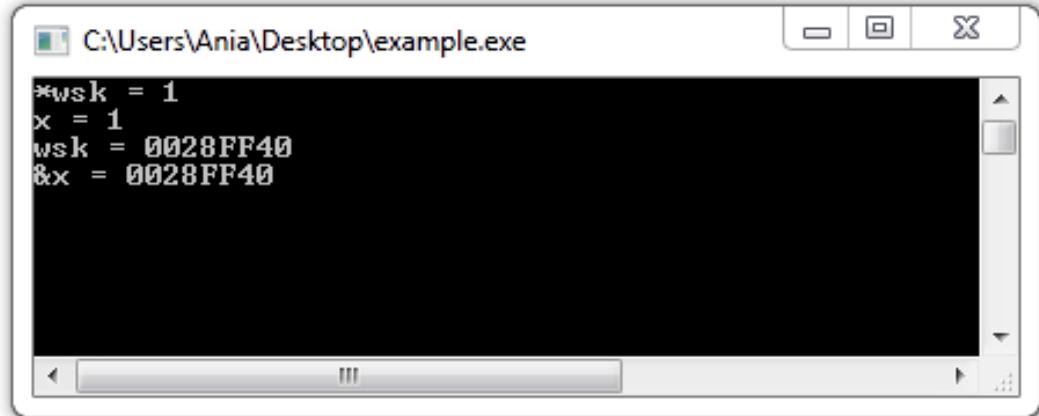
int main(void)
{
 int *wsk, x=1;

 wsk = &x; // przypisanie wsk adresu x

 /*wyświetlenie wartości zmiennej x*/
 printf("*wsk = %d\n", *wsk); // odwołanie niejawne
 printf("x = %d\n", x); // odwołanie jawne

 /*wyświetlenie adresu zmiennej x*/
 printf("wsk = %p\n", wsk);
 printf("&x = %p\n", &x);

 return 0;
}
```



# Zmiana wartości zmiennej poprzez wskaźnik

**TYP zmienna = wartość;**

**TYP \*wskaźnik = &zmienna;**

**TYP wart = \*wskaźnik;**

**\*wskaźnik = nowa\_wartość;**

```
float f;
float *pf = &f;
*pf = 10.8;
```

Nadanie wartości zmiennej za pośrednictwem wskaźnika

```
float f = 3.14;
float *pf = &f;
*pf = 10.8
```

Zmiana wartości zmiennej za pośrednictwem wskaźnika

```
float f = 3.14;
float *pf;
*pf = 10.8
```

BŁAD!  
Wskaźnik pf na nic nie wskazuje !

# Ćwiczenie

---

```
int x, a = 1, b = 2;
```

```
int *p;
```

**Obliczyć kolejne wartości x:**

1. p = &a; x = \*p; // x=1
2. \*p = 0; x = a; // x=0
3. p = &b; x \*= \*p; // x=0\*2=0
4. x = \*p; // x=2
5. x = \*p/2; // x=2/2=1

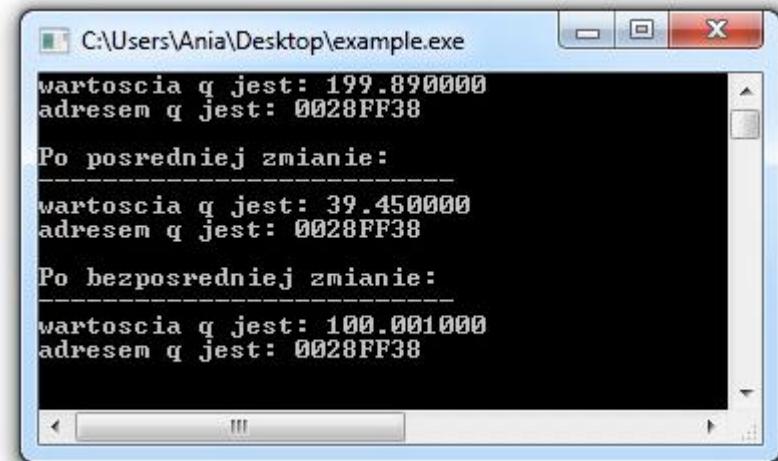
## Przykład 2

```
int main(void)
{
 double q, *p = &q;
 *p = 199.89; /*ustawienie wartości q
printf("wartoscia q jest: %lf\n", q);
printf("adresem q jest: %p\n\n", p);

*p = 39.45; /*zmiana wartości q za pośrednictwem wskaźnika*/
printf("Po posredniej zmianie:\n-----\n");
printf("wartoscia q jest: %lf\n", q);
printf("adresem q jest: %p\n\n", p);

q = 100.001; /*bezpośrednia zmiana wartości q*/
printf("Po bezposredniej zmianie:\n-----\n");
printf("wartoscia q jest: %lf\n", q);
printf("adresem q jest: %p\n", p);

return 0;
}
```



# Zgodność typów

`unsigned int zm = 65530;`

| Komórka 1       | Komórka 2       |
|-----------------|-----------------|
| 1 1 1 1 1 1 1 1 | 1 1 1 1 1 0 1 0 |

= (unsigned int) 65530

`unsigned char *wsk = &zm;`

przy próbie odczytania `*wsk` przyjmie wartość 255

| Komórka 1       |
|-----------------|
| 1 1 1 1 1 1 1 1 |

= (unsigned char) 255

`unsigned long int *wsk2 = &zm;`

przy próbie odczytania `*wsk2` przyjmie wartość 4294639360

| Komórka 1       | Komórka 2       | Komórka 3       | Komórka 4       |
|-----------------|-----------------|-----------------|-----------------|
| 1 1 1 1 1 1 1 1 | 1 1 1 1 1 0 1 0 | 1 1 1 1 1 1 1 0 | 0 0 0 0 0 0 0 0 |

# Przykład 3

---

```
#include <stdio.h>

int main(void)
{
 int *p;
 double q, tymcz;
 tymcz = 1234.34;

 p = &tymcz; //BŁĄD
 q = *p; //BŁĄD – kopiowane tylko 2 bajty

 printf("%f", q);

 return 0;
}
```

# Arytmetyka wskaźników

---

- ▶ Ze wskaźnikami można stosować tylko cztery operatory arytmetyczne, tj.: **+**, **++**, **-**, **--**
- ▶ **Do wskaźników dodawać i odejmować można tylko wartości całkowite.**
- ▶ Arytmetyka wskaźników wykonywana jest względem typu podstawowego wskaźnika – po każdorazowym zwiększeniu o jednostkę, będzie on wskazywał na następny element pamięci (w sensie zdefiniowanym przez typ podstawowy) występujący za elementem obecnie wskazywanym.
- ▶ **Wykonanie operacji arytmetycznej na wskaźniku zmienia wskazywany adres.**
- ▶ Wskaźników nie można mnożyć, dzielić oraz brać reszty z dzielenia całkowitego.

# Przykład 4a

```
#include <stdio.h>

int main(void) {

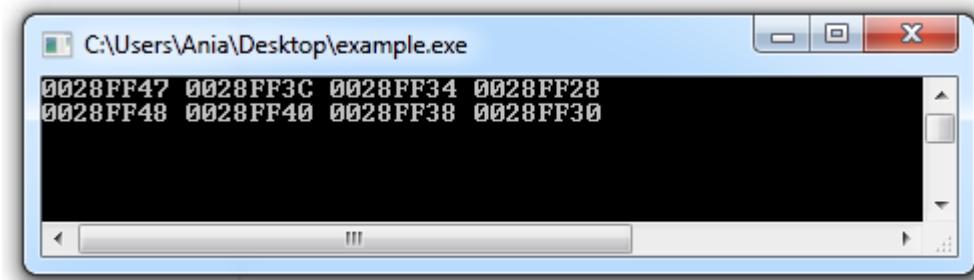
 char ch, *cp = &ch;
 int i, *ip = &i;
 float f, *fp = &f;
 double d, *dp = &d ;

 printf("%p %p %p %p\n", cp, ip, fp, dp);

 cp++; ip++; fp++; dp++;

 printf("%p %p %p %p\n", cp, ip, fp, dp);

 return 0;
}
```



# Przykład 4b

```
include <stdio.h>

int main(void) {

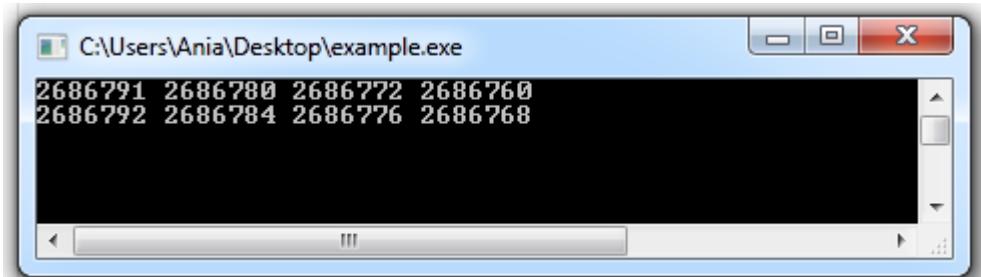
 char ch, *cp = &ch;
 int i, *ip = &i;
 float f, *fp = &f;
 double d, *dp = &d ;

 printf("%d %d %d %d\n", cp, ip, fp, dp);

 cp++; ip++; fp++; dp++;

 printf("%d %d %d %d\n", cp, ip, fp, dp);

 return 0;
}
```



# Arytmetyka wskaźników

- ▶ Zwiększenie wskazywanego obiektu a zwiększenie wskaźnika do obiektu 😊

Zwiększenie wskaźnika do obiektu: **\*wsk++**

- ▶ Zwiększenie wskaźnika powoduje jego przesunięcie na następną komórkę w pamięci.

Zwiększenie wskazywanego obiektu : **(\*wsk)++**

- ▶ Aby zwiększyć wskazywany obiekt należy najpierw go wyłuskać.

# Przykład 5

```
#include <stdio.h>

int main(void) {

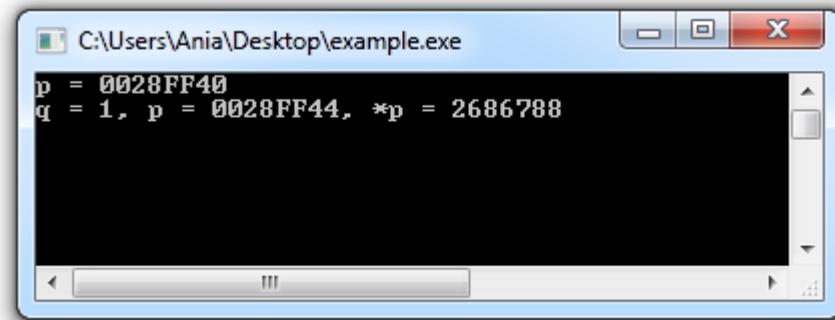
 int *p, q;
 p = &q;
 q = 1;

 printf("p = %p\n", p);

 *p++;

 printf("q = %d, p = %p, *p = %d", q, p, *p);

 return 0;
}
```



# Przykład 6

```
#include <stdio.h>

int main(void) {

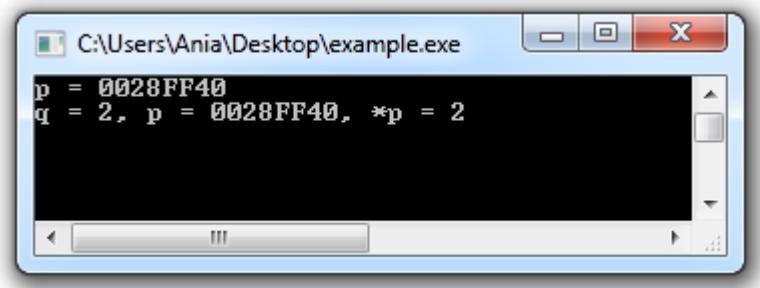
 int *p, q;
 p = &q;
 q = 1;

 printf("p = %p\n", p);

 (*p)++;

 printf("q = %d, p = %p, *p = %d", q, p, *p);

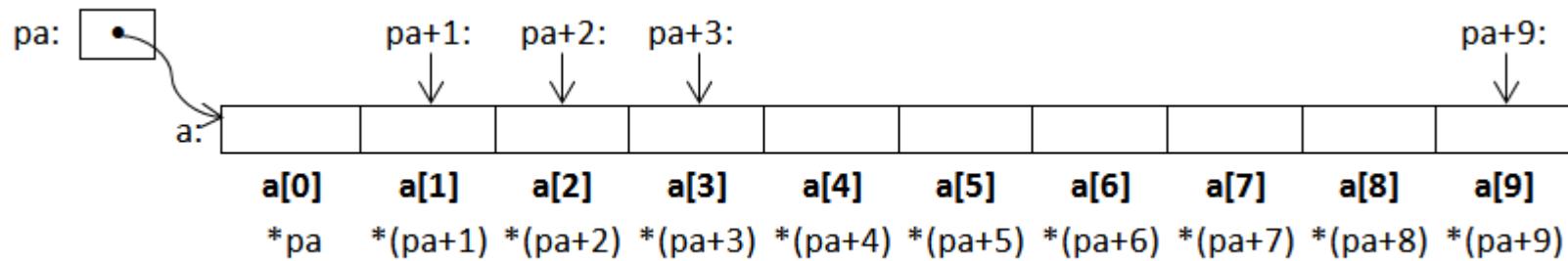
 return 0;
}
```



# Wskaźniki i tablice (1)

- W języku C tablice i wskaźniki są ścisłe związane i można je stosować wymiennie.

```
int a[10];
int *pa;
pa = &a[0]; // pa wskazuje na pierwszy element tablicy a
```

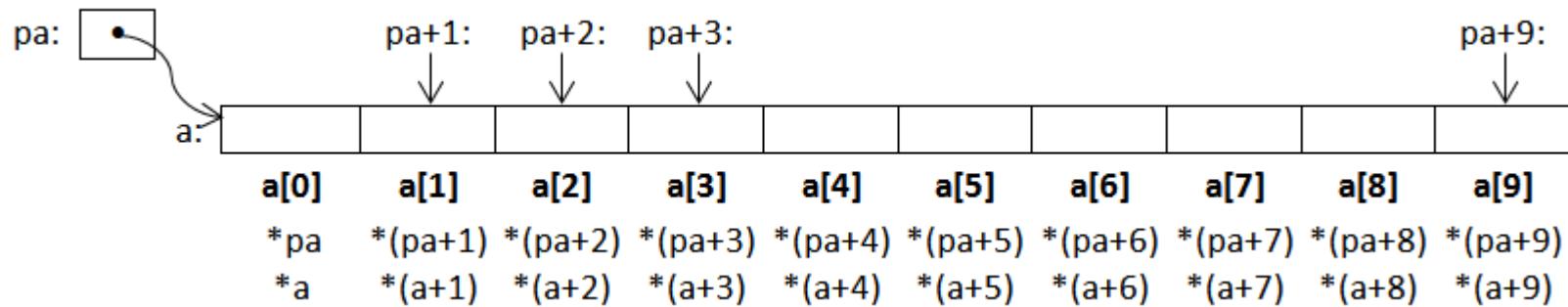


pa – adres pierwszego elementu tablicy  $\Leftrightarrow \&a[0]$   
\*pa – wartość pierwszego elementu tablicy  $\Leftrightarrow a[0]$   
pa+i – adres i-tego elementu tablicy  $\Leftrightarrow \&a[i]$   
\*(pa+i) – wartość i-tego elementu tablicy  $\Leftrightarrow a[i]$

# Wskaźniki i tablice (2)

- ▶ W języku C nazwa tablicy jest adresem jej pierwszego elementu.

```
int a[10];
int *pa;
pa = a; // pa wskazuje na pierwszy element tablicy a
```



a – adres pierwszego elementu tablicy  $\Leftrightarrow \&a[0]$   
\*a – wartość pierwszego elementu tablicy  $\Leftrightarrow a[0]$   
a+i – adres i-tego elementu tablicy  $\Leftrightarrow \&a[i]$   
\*(a+i) – wartość i-tego elementu tablicy  $\Leftrightarrow a[i]$

# Wskaźniki i tablice (3)

- ▶ Różnica między nazwą tablicy a wskaźnikiem

```
int a[10];
int *pa;
pa = a; // pa wskazuje na pierwszy element tablicy a
```

```
pa = a;
pa ++;
```

**Wyrażenia poprawne**, gdyż wskaźnik jest zmienną

```
a = pa;
a ++;
```

**Wyrażenia błędne**, gdyż nazwa tablicy nie jest zmienną

## Przykład 7a

```
#include <stdio.h>

int main(void) {

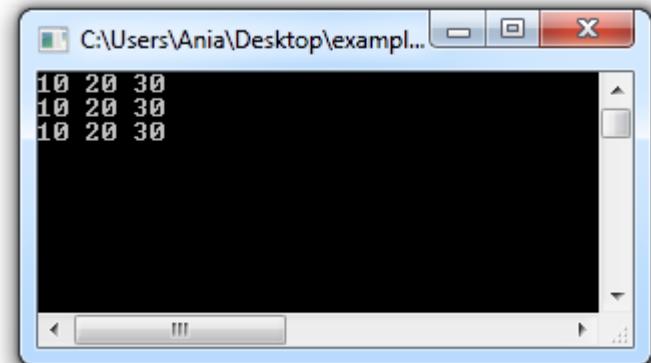
 int a[10] = {10, 20, 30, 40, 50, 60, 70, 80, 90, 100};
 int *pa = a; //przypisanie pa adresu tablicy

 /*wypisanie pierwszego, drugiego i trzeciego elementu tablicy*/
 printf("%d %d %d\n", a[0], a[1], a[2]);

 /*to samo z wykorzystaniem wskaźnika*/
 printf("%d %d %d\n", *pa, *(pa+1), *(pa+2));

 /*i jeszcze raz to samo */
 printf("%d %d %d\n", *a, *(a+1), *(a+2));

 return 0;
}
```



## Przykład 7b

```
#include <stdio.h>

int main(void) {

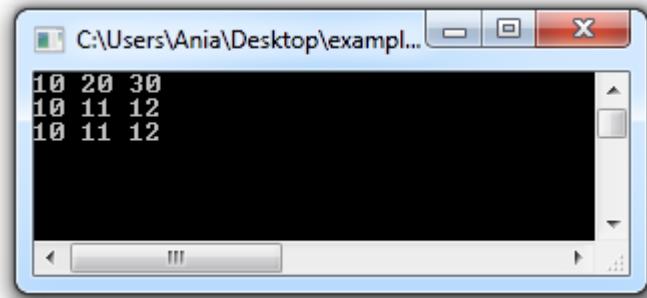
 int a[10] = {10, 20, 30, 40, 50, 60, 70, 80, 90, 100};
 int *pa = a; //przypisanie pa adresu tablicy

 /*wypisanie pierwszego, drugiego i trzeciego elementu tablicy*/
 printf("%d %d %d\n", a[0], a[1], a[2]);

 /*Błąd - brak nawiasów*/
 printf("%d %d %d\n", *pa, *pa+1, *pa+2);

 /*i jeszcze raz to samo */
 printf("%d %d %d\n", *a, *a+1, *a+2);

 return 0;
}
```



Błąd!!  
najpierw wyłuskanie, potem  
zwiększenie wartości!

## Wskaźniki i tablice (4)

---

- ▶ Dodanie wartości 1 (jeden) do wskaźnika powoduje dodanie jednej jednostki pamięci danego typu
  - ▶ W przypadku tablic oznacza to przesunięcie do następnego elementu tablicy (Przykład 8)
- ▶ W wyniku zastosowania operatora wyłuskania do adresów kolejnych komórek tablicy, można odczytać ich zawartość (Przykład 9a i 9b)
- ▶ Wskaźniki wskazujące na tablice można indeksować jak tablice (Przykład 10)

# Przykład 8

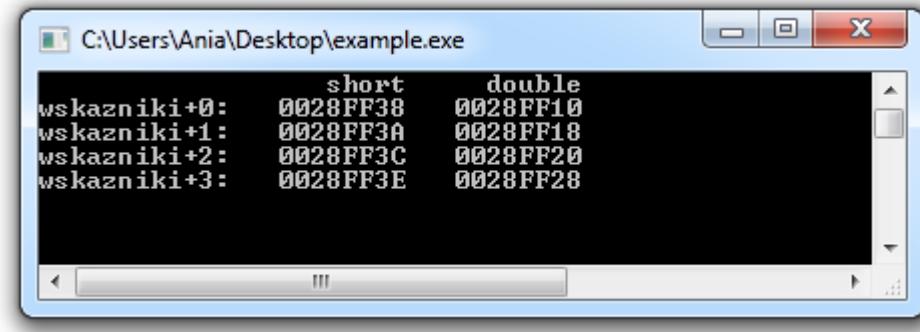
```
#include <stdio.h>
#define ROZMIAR 4

int main(void)
{
 short daty[ROZMIAR], *wski;
 double rachunki[ROZMIAR], *wskf;
 wski = daty;
 wskf = rachunki;
 short indeks;

 printf("%23s %10s\n", "short", "double");

 for (indeks = 0; indeks < ROZMIAR; indeks++)
 printf("wskazniki+%d: %10p %10p\n", indeks, wski+indeks,
 wskf+indeks);

 return 0;
}
```



# Przykład 9a

```
#include <stdio.h>

int main(void)
{
 int i;
 int a[10] = {10, 20, 30, 40, 50, 60, 70, 80, 90, 100};
 int *pa = a; //przypisanie pa adresu tablicy

 for (i=0; i<10; i++)
 printf("%d\t", *(pa+i));

 return 0;
}
```



# Przykład 9b

```
#include <stdio.h>

int main(void)
{
 int i;
 float a[10]={1.1, 2.2, 3.3, 4.4, 5.5, 6.6, 7.7, 8.8, 9.9, 10.1};
 float *pa = a; //przypisanie pa adresu tablicy

 for (i=0; i<10; i++)
 printf("%f\t", *(pa+i));

 return 0;
}
```



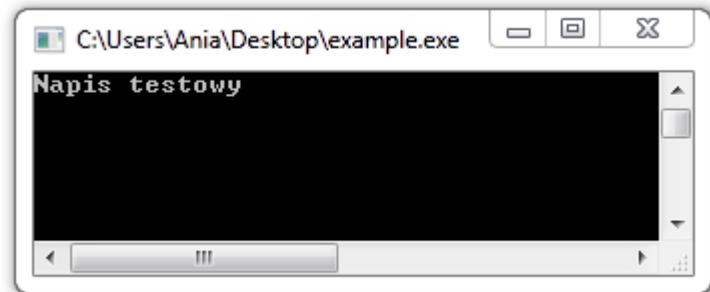
# Przykład 10

```
#include <stdio.h>

int main(void)
{
 char nap[]="Napis testowy";
 char *p = nap;
 int i;

 /*powtarzaj do napotkania NULL*/
 for(i=0; p[i]; i++)
 printf("%c",p[i]);

 return 0;
}
```

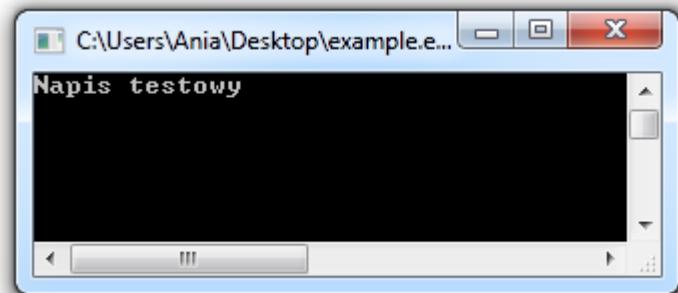


# Przykład 11

```
#include <stdio.h>

int main(void)
{
 char *p = "Napis testowy";
 printf(p);

 getchar();
 return 0;
}
```



# Przykład 12

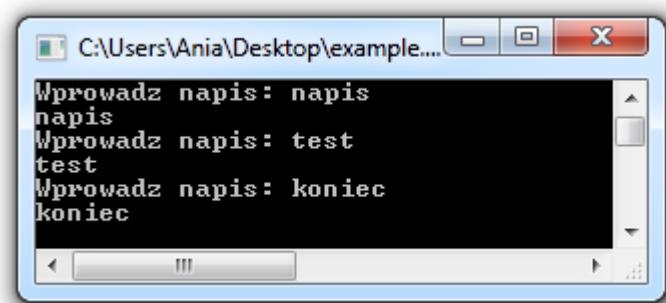
```
#include <stdio.h>
#include <string.h>

/*program wczytuje napisy do czasu pojawienia się słowa
"koniec"*/

int main(void)
{
 char *p = "koniec";
 char nap[80];

 do{
 printf("Wprowadz napis: ");
 fscanf("%79s",nap);
 puts(nap);
 }while(strcmp(p,nap) != 0);

 return 0;
}
```



# Wskaźnik jako parametr funkcji

- ▶ Wskaźniki, mogą być argumentami funkcji

**TYP NazwaFunkcji (TYP\* wsk);**

- ▶ Argumenty w postaci wskaźników pozwalają na dostęp oraz modyfikację obiektów w funkcji wywołującej

```
void swap(int x, int y)
{
 int temp;
 temp = x;
 x = y;
 y = temp;
}
...
int a=5, b=3;
swap(a,b);

// a=5, b=3
```

```
void swap(int *x, int *y)
{
 int temp;
 temp = *x;
 *x = *y;
 *y = temp;
}
...
int a=5, b=3;
swap(&a, &b);

// a=3, b=5
```

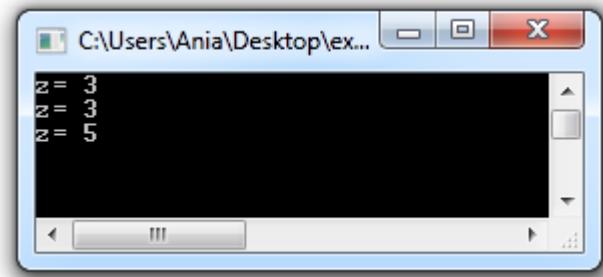
# Przykład 13

```
#include <stdio.h>

void func_var (int zmienna) {
 zmienna = 4;
}

void func_pointer (int *zmienna) {
 (*zmienna) = 5;
}

int main (void) {
 int z=3;
 printf ("z= %d\n", z);
 func_var (z); printf ("z= %d\n", z);
 func_pointer (&z);
 printf ("z= %d\n", z);
 return 0;
}
```



# Funkcja z parametrami tablicowymi

- ▶ Język C interpretuje **TYP tab []** tak samo jak **TYP \*tab**

```
int sumuj(int *tab, int n);
int sumuj(int *, int);
int sumuj(int tab[], int n);
int sumuj(int [], int);
```

# Przykład 15

```
#include <stdio.h>
#define ROZMIAR 10

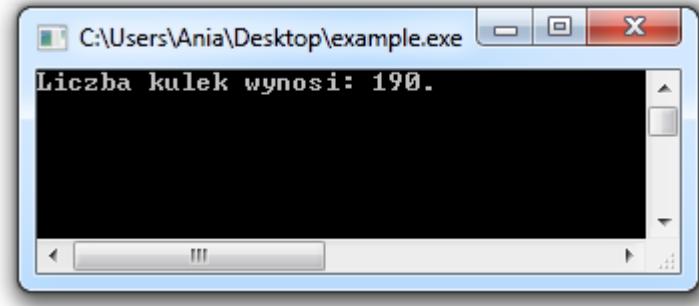
long sumujw(int *początek, int* koniec);

int main(void) {
 int kulki[ROZMIAR] = {20,10,5,39,4,16,19,26,31,20};
 long wynik;
 wynik = sumujw(kulki, kulki+ROZMIAR);
 printf("Liczba kulek wynosi: %ld.\n", wynik);
 return 0;
}

long sumujw(int *początek, int* koniec){
 long suma = 0;

 while(początek<koniec) {
 suma += *początek; /*dodaje do sumy*/
 początek++; /*przesuwa wskaźnik do następnego elementu*/
 /* suma += *początek ++*/
 }

 return suma;
}
```



# Ochrona zawartości tablicy

- ▶ Aby uniemożliwić zmianę wartości tablicy przez funkcję, w deklaracji argumentu formalnego należy użyć słowa kluczowego **const**

**TYP nazwa\_funkcji(const TYP\* wsk);**

```
long sumuj(const int *tab, int n); /*prototyp*/

long sumuj(const int *tab, int n) /*definicja*/
{
 int i;
 long suma = 0;
 for(i=0; i<n; i++)
 {
 suma+=(* (tab+i))++;
 }
 return suma;
}
```

Błąd !!

# Wskaźnik void

- ▶ Wskaźnik do danych nieokreślonego typu.

```
void* wsk;
```

- ▶ Może pokazywać na dane dowolnego typu oraz można mu przypisać wskaźnik dowolnego typu.

```
void *wskv;
int *wski;
float *wskf

wskv = wski; // operacja poprawna
wskv = wskf; // operacja poprawna
```

- ▶ Aby przypisać wskaźnik void do wskaźnika innego typu, należy dokonać **rzutowania typów**

```
wski = wskv; // BŁĄD !!
wski = (int*)wskv; // operacja poprawna
wskf = (float*)wskv; // operacja poprawna
```

# Przykład 31

```
#include <stdio.h>

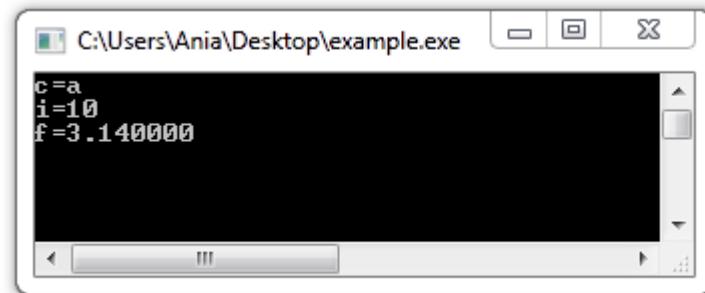
int main() {
 void *wskv;
 char c = 'a';
 int i=10;
 float f=3.14;

 wskv = &c;
 printf("c=%c\n", *(char*)wskv);

 wskv = &i;
 printf("i=%d\n", *(int*)wskv);

 wskv = &f;
 printf("f=%f\n", *(float*)wskv);

 return 0;
}
```



rzutowanie typów

wskv = &c;  
printf("c=%c\n", \*(char\*)wskv);

wskv = &i;  
printf("i=%d\n", \*(int\*)wskv);

wskv = &f;  
printf("f=%f\n", \*(float\*)wskv);

# Przykład 78<sub>(1)</sub>

```
#include <stdio.h>

void dr_nap(const char *p);

int main(void) {
 char nap[80];
 printf("Wprowadz napis: ");
 fgets(nap, 79, stdin);

 dr_nap(nap);

 getchar();

 return 0;
}

void dr_nap(const char *p) {
 while(*p) putchar(*p++);
}
```

"obietnica", że funkcja nie będzie modyfikowała argumentu

## Przykład 78(2)

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

void dr_nap(const char *p);

int main(void) {
 char nap[80];

 printf("Wprowadz napis: ");
 fgets(nap, 79, stdin);

 dr_nap(nap);

 return 0;
}

void dr_nap(const char *p) {
 while(*p) {
 *p = toupper(*p); //BŁĄD
 putchar(*p++);
 }
}
```

| Line | File                                                                   | Message                                                                     |
|------|------------------------------------------------------------------------|-----------------------------------------------------------------------------|
| 23   | C:\Users\Ania\Desktop\example.cpp<br>C:\Users\Ania\Desktop\example.cpp | In function `void dr_nap(const char*)':<br>assignment of read-only location |