Przekazywanie struktur do funkcji

Przekazywanie składowych struktury

- Składowe przekazuje się tak jak zmienne proste:
  - przez wartość
  - przez adres
  - przez referencję

- Przykład 1. Przekazywanie przez wartość

```
#include <iostream.h>
struct Ulamek {
   long licznik;
   long mianownik;
  };
void DrukujUlamek(char *opis,long x) {
  cout << opis << x << endl;
}
int main() {
  struct Ulamek u;
  u.licznik=1; u.mianownik=2;
  DrukujUlamek("Licznik: ",u.licznik);
  DrukujUlamek("Mianownik: ",u.mianownik);
  return 0;
}
```

- Przykład 2. Przekazywanie przez adres

```
#include <iostream.h>
struct Ulamek {
   long licznik;
   long mianownik;
  };
void DrukujUlamek(char *opis, long *x) {
  cout << opis << *x <<endl;
}
int main() {
  struct Ulamek u;
  u.licznik=1; u.mianownik=2;
  DrukujUlamek("Licznik:",&u.licznik);
  DrukujUlamek("Mianownik:",&u.mianownik);
  return 0;
}
```

- Przykład 3. Przekazywanie przez referencję

```
#include <iostream.h>
#include <iostream.h>
struct Ulamek {
   long licznik;
   long mianownik;
  };
void DrukujUlamek(char *opis,long &x) {
  cout << opis << x << endl;
}
int main() {
  struct Ulamek u;
  u.licznik=1;  u.mianownik=2;
  DrukujUlamek("Licznik: ",u.licznik);
  DrukujUlamek("Mianownik: ",u.mianownik);
  return 0; }
```

Przekazywanie całych struktur

- Użycie struktury jako argumentu funkcji powoduje przekazanie struktury *przez wartość* i *nie jest zalecane* szczególnie w przypadku dużych struktur.

- Przykład 1. Przekazywanie przez wartość

```
#include <iostream.h>
struct Ulamek {
   long licznik;
   long mianownik;
  };
void DrukujUlamek(char *opis, Ulamek x) {
  cout << opis << x.licznik << '/' << x.mianownik << endl;
}
int main() {
  struct Ulamek u;
  u.licznik=1; u.mianownik=2;
  DrukujUlamek("Licznik/Mianownik ",u);
  return 0;
}
```

- Przykład 2. Przekazywanie przez adres

```
#include <iostream.h>
struct Ulamek {
   long licznik;
   long mianownik;
  };
void DrukujUlamek(char *opis, Ulamek *x) {
  cout << opis << x->licznik << '/' << x->mianownik << endl;
}
int main() {
  struct Ulamek u;
  u.licznik=1; u.mianownik=2;
  DrukujUlamek("Licznik/Mianownik: ",&u);
  return 0;
}
```

- Przykład 3. Przekazywanie przez referencję

```
#include <iostream.h>
#include <iostream.h>
struct Ulamek {
   long licznik;
   long mianownik;
  };
void DrukujUlamek(char *opis, Ulamek &x) {
  cout << opis << x.licznik << '\' << x.mianownik << endl;
}
int main() {
  struct Ulamek u;
  u.licznik=1;  u.mianownik=2;
  DrukujUlamek("Licznik/Mianownik: ", u);
  return 0;
}
```

Przykład:

```
#include <iostream.h>
struct Pomiary {
   char nazwa;
   int a,b;
 };
 void drukuj(Pomiary proba);
 int main() {
   Pomiary proba;
   proba.nazwa='A';
   proba.a=100;
   proba.b=20;
   drukuj(proba);
   return 0;
 }
// Przekazywanie przez wartość
 void drukuj(Pomiary proba) {
   cout << "Nazwa: "<< proba.nazwa
        <<" Wyniki: " << proba.a << '\t' << proba.b << endl;
 }
```

- *Uwaga*: należy unikać takiego przekazywania, ponieważ niepotrzebnie zwiększa się czas wykonywania oraz może wystąpić przepełnienie stosu.
- Zalecane jest posługiwanie się referencją lub wskaźnikiem do pierwszego elementu struktury.

```
#include <iostream.h>
struct Pomiary {
   char nazwa;
   int a,b;
 };
 void drukuj(Pomiary &proba);
 int main() {
   Pomiary proba;
   proba.nazwa='A';
   proba.a=100;
   proba.b=20;
   drukuj(proba);
   return 0;
 }
// Przekazywanie przez referencję
 void drukuj(Pomiary &proba) {
   cout << "Nazwa: "<< proba.nazwa
        <<" Wyniki: " << proba.a << '\t' << proba.b << endl;
 }
```

Zwracanie struktur z funkcji

- Funkcja może zwracać strukturę lub wskaźnik do struktury.
- Przykład 1:

```
#include <iostream.h>

struct Punkt {
  int x;
  int y;
};

Punkt wprowadz(char *tekst) {
  Punkt p;
  cout << tekst<<": ";
  cin >> p.x >> p.y;
  return p;
}
int main() {
  Punkt p1=wprowadz("Podaj wspolrzedne punktu p1");
  cout << "Punkt p1: " << p1.x << ',' << p1.y << endl;
  Punkt p2=wprowadz("Podaj wspolrzedne punktu p2");
  cout << "Punkt p2: " << p2.x << ',' << p2.y << endl;
  return 0;
}
```

- Przykład 2: funkcja zwraca wskaźnik do punktu o najmniejszej współrzędnej x

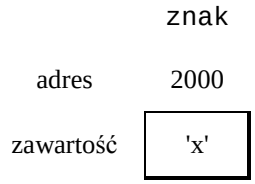
```
#include <iostream.h>
#include <math.h>   // dla abs()

struct Punkt {
  int x;
  int y;
};

Punkt *minX(Punkt *tablica, int ile)
{
  int p=0; // indeks punktu o najmniejszej współrzędnej x
  for (int i=1;i<ile;i++)
    if (abs(tablica[p].x) > abs(tablica[i].x)) p=i;
  return (tablica+p);
}
int main() {
  Punkt t[]={{1,2},{-3,5},{2,8}};
  Punkt *w=minX(t,sizeof(t)/sizeof(*t));
  cout << "Punkt o namniejszej wsp. x: "
       << w->x << ',' << w->y << endl;
  return 0;
}
```

Napisy

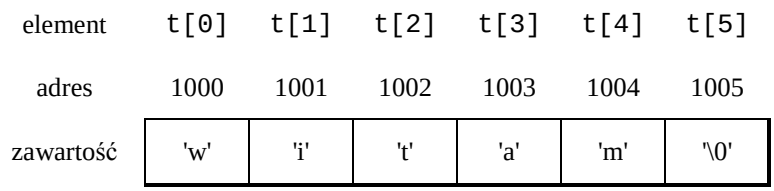
- Napis (*tekst*, *łańcuch* , ang. *string*) jest to grupa znaków traktowanych jako całość. Napis może zawierać litery, cyfry, znaki specjalne.
- Języki C/C++ nie posiadają wbudowanego podstawowego typu pozwalającego przechowywać napisy.
- W języku C n*apis* (ang. *string*) jest przechowywany jako ciąg znaków zakończonych znakiem pustym (znak o kodzie ASCII 0, `'\0'`, `NULL`).
- W języku C++ można korzystać z reprezentacji napisu:
  - jako ciągu znaków zakończonych znakiem pustym, mówimy wtedy o napisie w stylu języka C
  - jako obiektu klasy `string`, należącego do standardowego języka C++.

Napisy w stylu języka C

- Napis jest przechowywany w tablicy typu `char`.
- Napis kończy się znakiem pustym `'\0'` (znak o kodzie ASCII 0 - NULL). *Dzięki temu standardowe funkcje wiedzą, gdzie kończy się napis bez konieczności podawania jego długości*.
- Przykład:
  - Przechowywanie znaku w pamięci – zmienna typu `char`



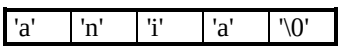
  - Przechowywanie napisu w pamięci – tablica typu `char`



Deklarowanie i inicjalizowanie napisów

- Do przechowywania napisu należy zadeklarować tablicę typu `char`. Przykład deklaracji:
  `char napis[5]; // pozwala przechować do 4 znaków i znak '\0'`
- Podczas deklaracji można inicjalizować tablicę:

```
char imie[5]="ania"; // zawsze o 1 więcej niż długość tekstu
char imie[]="ania";  // kompilator sam dobierze długość
```



- Kompilator automatycznie uzupełniani stałą tekstową (ciąg znaków w cudzysłowach `" "`) ogranicznikiem tekstu `'\0'` (symbol `NULL`).

Wczytywanie i wyświetlanie napisów

Metoda 1 - za pomocą operatorów << i >>

```
#include <iostream.h>
int main()
{
  char nazwisko[20];
  cout << "Podaj nazwisko: ";
  cin >> nazwisko;
  cout << "Nazywasz sie " << nazwisko << endl;
  return 0;
}
```

Wykonanie:

```
Podaj nazwisko: Kowalski
Nazywasz się Kowalski

Podaj nazwisko: Jan Kowalski
Nazywasz się Jan
```

Metoda 2 - za pomocą funkcji `getline()`

```
#include <iostream.h>
int main()
{
  char nazwisko[20];
  cout << "Podaj imie i nazwisko: ";
  cin.getline(nazwisko,20);
  cout << "Nazywasz sie " << nazwisko << endl;
  ile= cin.gcount();
  cout << "Wczytano: " << ile << " znakow" << endl;
  cout << "Dlugosc napisu: " << ile-1 << endl;
  return 0;
}
```

Wykonanie:

```
Podaj imie i nazwisko: Jan Kowalski
Nazywasz sie: Jan Kowalski
Wczytano 13 znakow
```

Uwagi:

- Funkcja `getline` wczytuje napis do bufora podanego jako pierwszy parametr.
- Wczytywanie domyślnie jest wykonywane do napotkania znaku nowej linii `'\n'`. Dzięki temu można wczytać cały wiersz tekstu, również spacje rozdzielające słowa.
- Funkcja zabezpiecza się przed przekroczeniem rozmiaru bufora, ponieważ odczyt jest przerywany po wczytaniu
  n-1 znaków, gdzie *n* oznacza drugi parametr funkcji `getline()`.
- Znak końca wiersza *nie jest* umieszczany w buforze podanym jako parametr. Jednakże jest on wliczany do przeczytanych znaków, stąd uwzględnia go funkcja `gcount()`.
- W miejsce znaku końca wiersza funkcja `getline` umieszcza znak końca napisu `'\0'`.

Związek między tablicą znakową a wskaźnikiem

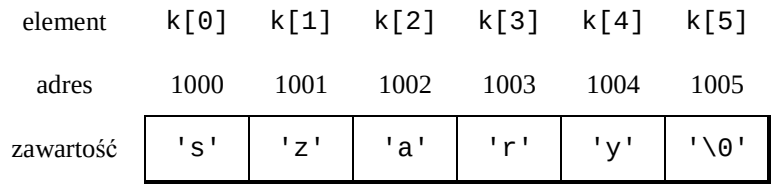
- Tak jak w zwykłej tablicy - nazwa tablicy jest adresem pierwszego elementu tablicy.
- Do elementów tablicy można się odwoływać za pomocą indeksów lub za pomocą operatora wyłuskania *.

Przykład 1. Zależność między tablicą i wskaźnikiem

- Dla tablicy `char imie[]="ania";`
  prawdziwa jest zależność: `imie[0] == *imie == 'a'`

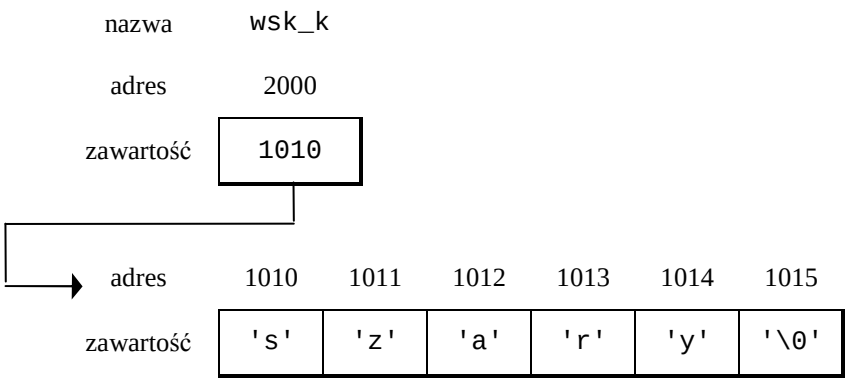
Przykład 2. Różnica między tablicą i wskaźnikiem

- W deklaracji napis może być przypisany zarówno tablicy znaków jak i zmiennej wskaźnikowej typu `char*`.
- Przypisanie napisu tablicy: `char k[]="szary";`



Działanie: Tworzona jest tablica, w każdym elemencie tej tablicy przechowywany jest kolejny znak, na końcu dopisywany jest symbol końca napisu `\0`.

- Przypisanie napisu wskaźnikowi: `char *wsk_k="szary";`



Działanie: Tworzona jest zmienna wskaźnikowa, w niej umieszczany jest adres obszaru pamięci, w którym zostanie umieszczony napis uzupełniony symbolem końca napisu `\0`.

- Podobieństwa i różnice
  - W obydwu przypadkach można korzystać z notacji tablicowej:

```
for (i=0; k[i]!= '\0';i++)
  cout << k[i];
cout << endl;
```

```
for (i=0; wsk[i]!=0; i++)
  cout << wsk[i];
cout << endl;
```

  - W obydwu przypadkach można korzystać z notacji wskaźnikowej:

```
for (i=0; *(k+i) != '\0';i++)
  cout << *(k+i);
cout << endl;
```

```
for (i=0; *(wsk+i)!= '\0';i++)
  cout << *(wsk_k+i));
cout << endl;
```

  - Tylko w notacji wskaźnikowej można zmieniać wartość zmiennej:

nie	`while(*(wsk) != '\0')`
	`  cout << *wsk_k++;`
	`cout << endl;`

Przykłady przypisywania napisów

```
char kolor[]="zielony"; //OK:
       // podczas inicjalizacji można przypisać tablicy wartość
char *wsk="zielony"; // OK:
   // podczas inicjalizacji można przypisać wskaźnikowi wartość


char kolor[10], *wsk1;
kolor="zielony";  // błąd:
            // nazwa tablicy jest wskaźnikiem stałym
            // przypisanie wartości tylko podczas inicjalizacji
wsk1="zielony";   // OK:
         // wskaźnik jest zmienną;
         // teraz wsk1 wskazuje na nowy napis gdzieś w pamięci


char kolor[10];
char *wsk;
kolor[6] = 'a';    // OK:
                   // przypisanie wartości elementowi tablicy
wsk1[6] = 'e';     // błąd:
                   // wskaźnik musi być zainicjalizowany

char *wsk="zielony";
wsk1[6] = 'e';     // OK:
                   // wsk1 wskazuje na napis: zielone
wsk1[8] = '1';     // błąd:
                   // wsk1 wskazuje krótszy napis

char kolor[]="zielony";
char *wsk1;
wsk1=kolor;
wsk1[6] = 'e';     // OK:
                   // wsk1 wskazuje na napis: zielone

char *wsk2;
*wsk2 = "niebieski"; // błąd:
                 // adres może być przypisany tylko wskaźnikowi
wsk2 = "niebieski"; // OK:
      // w tym przypadku wskaźnik może być nie zainicjalizowany
```

Przykłady różne

- Wyświetlanie napisu od końca

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

int main(){
  char napis[80], roboczy;
  int pierwszy,ostatni;

  cout << "Wpisz tekst:" << endl;
  cin.getline(napis,80);

  pierwszy=0;
  ostatni=strlen(napis)-1;

  while (ostatni>pierwszy) {
    roboczy=napis[pierwszy];
    napis[pierwszy++] = napis[ostatni];
    napis[ostatni--]=roboczy;
  }

  cout << "Napis od tylu:" << endl
       << napis << endl;
  return 0;
}
```

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

int main(){
 char napis[80], roboczy;
 char *pierwszy,*ostatni;

 cout << "Wpisz tekst: ";
 cin.getline(napis,80);

 pierwszy=napis;
 ostatni=napis + (strlen(napis)-1);

 while (ostatni>pierwszy) {
   roboczy=*pierwszy;
   *pierwszy++ = *ostatni;
   *ostatni--=roboczy;
 }
 cout << "Napis od tylu:" << endl
      << napis << endl;
 return 0;
}
```

Przekazywanie napisów do funkcji

Przykład 1

```
#include <iostream.h>

void drukuj(char tekst[])
{
 cout << tekst << endl;
}
int main()
{
  drukuj("Adam Nowak");
  drukuj("Piotr Janicki");
  return 0;
}
```

```
#include <iostream.h>

void drukuj(char *tekst)
{
 cout << tekst << endl;
}
int main()
{
  drukuj("Adam Nowak");
  drukuj("Piotr Janicki");
  return 0;
}
```

Przykład 2

```
#include <iostream.h>
```

```
int dlugosc_napisu(char *napis) {
  int dlugosc=0;
  while (*napis != '\0'){
      dlugosc++;
      napis++;
    }
  return (dlugosc);
}
```

```
int dlugosc_napisu(char *napis) {
  int dlugosc=0;
  if (napis)
    while (*napis++) dlugosc++;
  return (dlugosc);
}
```

```
int dlugosc_napisu(char *napis) {
  int dlugosc;
  if (napis)
    for(dlugosc=0; *napis++ != '\0';)
      dlugosc++;
  return (dlugosc);
}
```

```
int main() {
  char nazwisko[]="Nowak";
  cout << "Nazwisko " << nazwisko << " ma " << dlugosc_napisu(nazwisko)
       << " znaków " << endl;
  return 0;
}
```

Przykład 3

```
#include <iostream.h>

char *wielkie_litery(char *tekst){
  char *poczatek=tekst; // adres tekst[0]
  while (*tekst)
  {
    if ( (*tekst>='a') && (*tekst<='z'))
      *tekst=*tekst - 'a' +'A';
    tekst++ ;
  }
  return (poczatek);
}
int main (){
  cout << wielkie_litery ("Jan Nowak") << endl;
  return 0;
}
```

Przykład 4.

```
#include <iostream.h>

/* Funkcja dopasuj() zwraca wskaźnik do miejsca napisu,
   w którym znajduje się pierwsze wystąpienie znaku c.
   Jeżeli znak taki nie istnieje, to zwracany
   jest wskaźnik do znaku końca napisu */
char *dopasuj(char c, char *s);

int main() {
  char s[80], *p, znak;
  cout << "Wpisz tekst:"<< endl;
  cin.getline(s,80);
  cout << "Wpisz znak:"<<endl;
  cin >> znak;
  p=dopasuj(znak,s);
  if (*p) /* znak został znaleziony */
     cout << "Tekst od znaku " << endl << p << endl;
  else
     cout << "Znak nie został znaleziony." << endl;
  return 0;
}

char *dopasuj(char c, char *s)
{
  int licznik;
  licznik=0;
  while (c !=s[licznik] && s[licznik]) licznik++;
  return (&s[licznik]);
}
```

Funkcje biblioteczne działające na napisach

- Funkcje działające na napisach korzystają z pliku nagłówkowego `string.h`.

Prototyp funkcji	Opis funkcji
`size_t strlen(const char *s1)`	Zwraca długość napisu wskazywanego przez *`s1`* (bez znaku kończącego napis `'\0'`).
`char *strcpy(char *s1, const char *s2)`	Kopiuje napis z *`s2`* do tablicy znakowej *`s1`*. Zwraca *`s1`*. *Uwaga*: tablica znakowa *`s1`* musi mieć wystarczającą długość, aby zmieścił się w niej napis *`s2`*.
`char *strncpy(char *s1, const char *s2, size_t n)`	Kopiuje z *`s2`* co najwyżej *`n`* znaków do tablicy znakowej *`s1`*. Zwraca *`s1`*. *Uwaga*: znak kończący napis *`s2`* zostanie skopiowany tylko wtedy, kiedy *`n`* jest większe od długości *`s2`*, w przeciwnym wypadku trzeba samemu dopisać znak `'\0'`. Tablica znakowa *`s1`* musi mieć wystarczającą długość, aby zmieściło się w niej *`n`* znaków napisu *`s2`* i znak końca napisu.
`char *strcat(char *s1, const char *s2)`	Dołącza napis *`s2`* do napisu *`s1`*. Pierwszy znak *`s2`* nadpisuje znak końca *`s1`*. Zwraca *`s1`*. *Uwaga*: tablica znakowa *`s1`* musi mieć wystarczającą długość, aby zmieścił się w niej połączony napis.
`char *strncat(char *s1, const char *s2, size_t n)`	Dołącza co najwyżej *`n`* znaków napisu *`s2`* do napisu *`s1`*. Na końcu utworzonego napisu umieszcza `'\0'`. Zwraca *`s1`*. Uwaga: tablica znakowa *`s1`* musi mieć wystarczającą długość, aby zmieścił się w niej połączony napis.
`int strcmp(const char *s1, const char *s2)`	Porównuje napis *`s1`* z napisem *`s2`* i zwraca: • jeżeli oba napisy są równe • liczbę > 0, jeżeli napis *`s1`* jest większy od napisu *`s2`* • liczbę < 0, jeżeli napis *`s1`* jest mniejszy od napisu *`s2`*. Porównywanie jest wykonywane leksykograficznie.

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

Porównuje leksykograficznie *n* znaków napisu *s1* z napisem *s2* i zwraca:

- jeżeli oba napisy są równe
- liczbę > 0, jeżeli napis *s1* jest większy od napisu *s2*
- liczbę < 0, jeżeli napis *s1* jest mniejszy od napisu *s2*.

Porównywanie jest wykonywane leksykograficznie.

`char *strchr(char*s, int c)`

Znajduje *pierwsze* wystąpienie znaku *c* w napisie *s*. Zwraca wskaźnik odnalezionego znaku lub NULL, jeśli znak nie występuje w napisie.

`char *strrchr(char*s, int c)`

Znajduje *ostatnie* wystąpienie znaku *c* w napisie *s*. Zwraca wskaźnik odnalezionego znaku lub NULL, jeśli znak nie występuje w napisie.

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

Sprawdza czy napis *s1* zawiera napis *s2*; zwraca wskaźnik do pierwszego wystąpienia napisu *s2* w tekście *s1*, lub NULL, jeśli *s2* nie występuje w *s1*

Przykład 1.

```
#include <iostream.h>
#include <string.h>
int main() {
  char x[]="Adam Kowalski";
  char y[50], z[25];
  cout << "Napis w tablicy x:"<< endl << x <<endl;
  strcpy(y,x);  // kopiowanie do tablicy y
  cout << "Napis w tablicy y:"<< endl << y <<endl;
  strncpy(z,x,4); // nie kopiuje znaku końca napisu
  z[4]='\0';      // trzeba samemu dopisać znak końca
  cout << "Napis w tablicy z:"<< endl << z <<endl;
  return 0;
}
```

Przykład 2.

```
#include <iostream.h>
#include <string.h>
int main() {
  char napis[80];
  char *p_wsk, *k_wsk;
  char znak;
  cout << "Wpisz tekst:"<<endl;
  cin.getline(napis,80);
  p_wsk=napis;
  k_wsk=p_wsk+(strlen(napis)-1);
  while (k_wsk > p_wsk) {
    znak= *p_wsk;
    *p_wsk++ = *k_wsk;
    *k_wsk-- = znak;
  }
  cout << "Odwrocony tekst:" << endl << napis << endl;
  return 0;
}
```

Przykład 3

```
#include <iostream.h>
#include <string.h>
int main() {
  int i;
  char alfabet[27], dopisz[2];
  dopisz[1]=NULL;
  strcpy(alfabet,"a");
  for (i='b';i<='z';i++) {
    dopisz[0]=i;
    strcat(alfabet,dopisz);
  }
  cout << "Alfabet" << endl << alfabet<<endl;
  return 0; }
```

Dynamiczne przydzielanie pamięci na napisy

- Napisy przechowywane są w tablicach. Jeśli nie znamy najdłuższego napisu, to nie możemy z góry zarezerwować odpowiednio dużej tablicy. Musimy posłużyć się techniką dynamicznego przydzielania pamięci.

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

int main(){
  char bufor[256]; // duży bufor dla wszystkich wczytywanych napisów

  cin.getline(bufor,256);          // wczytanie tekstu
  int rozmiar=strlen(bufor+1);     // długość wczytanego napisu
  char *tekst1=new char[rozmiar];  // przydzielenie pamięci na wczytany
napis
  strcpy(tekst1,bufor);            // skopiowanie z bufora we właściwe
miejsce
  cout << "tekst1=\""<<tekst1<<"\""<<endl;

  cin.getline(bufor,256);          // wczytanie nowego tekstu
  // dalszy ciąg programu
  return 0;
}
```

Wyszukiwanie

- Podstawowe funkcje pozwalające przeszukiwać napisy to:

`char *strchr(char*s, int c)` – szuka *pierwszego* wystąpienia znaku `c` w napisie `s`.

`char *strrchr(char*s, int c)` - szuka *ostatniego* wystąpienia znaku `c` w napisie `s`.

`char *strstr(const char *s1, const char *s2)` – szuka napisu `s2` w `s1`

`char *strtok(char *s1, const char *s2)` – wyszukuje w napisie `s1` ciągi znaków rozdzielone znakami separatorów zdefiniowanych w napisie `s2`

Korzystając z tych funkcji można budować inne, bardziej złożone.

- Przykład: znaleźć ostatnie wystąpienie napisu `s2` w `s1`; funkcja ma zwracać wskaźnik do miejsca, w którym rozpoczyna się ostatnie wystąpienie napisu `s2`, jeśli napis `s1` nie zawiera napisu `s2`, funkcja ma zwrócić `NULL`

```
char *strrstr( char  *s1, char  *s2 )
{
  char *ostatni = NULL;
  char *biezacy;

// szukaj tylko wtedy, kiedy napis s2 nie jest pusty
  if( *s2 != '\0' ){
// znajdź pierwsze wystąpienie napisu
    biezacy = strstr( s1, s2 );
// powtarzaj wyszukiwanie
    while( biezacy != NULL ){
      ostatni = biezacy;
      biezacy = strstr( ostatni + 1, s2 );
    }
  }
  return ostatni;
}
```

- Przykład: podzielić tekst na wyrazy, każdy z nich wydrukować w oddzielnej linii; separatorem wyrazu jest spacja, tabulacja, powrót karetki, znak nowego wiersza. Wykorzystać funkcję biblioteczną `strtok()`.

```
void drukuj_wyrazy( char *tekst )
{
// separatory: spacja, tabulacja, powrót karetki, nowy wiersz
  static char odstep[] = " \t\r\n";
  char *wyraz;

   for( wyraz = strtok( tekst, odstep ); wyraz != NULL;
        wyraz = strtok( NULL, odstep ) )
     cout << "Nastepny wyraz: "<< wyraz << endl ;
}
```

Komentarz: Funkcja `strtok()` działa następująco: jeśli pierwszy argument jest różny od `NULL`, funkcja znajduje pierwszy wyraz (ang.*token*) oddzielony od reszty tekstu jednym z podanych separatorów w drugim argumencie, zapamiętuje jego pozycję i zwraca wskaźnik do początku znalezionego wyrazu. Jeśli pierwszy argument jest równy `NULL`, funkcja korzystając z zapamiętanej pozycji poprzedniego wyrazu, zwraca wyraz następny i zapamiętuje jego pozycję. Funkcja zwróci `NULL`, gdy nie znajdzie więcej wyrazów.

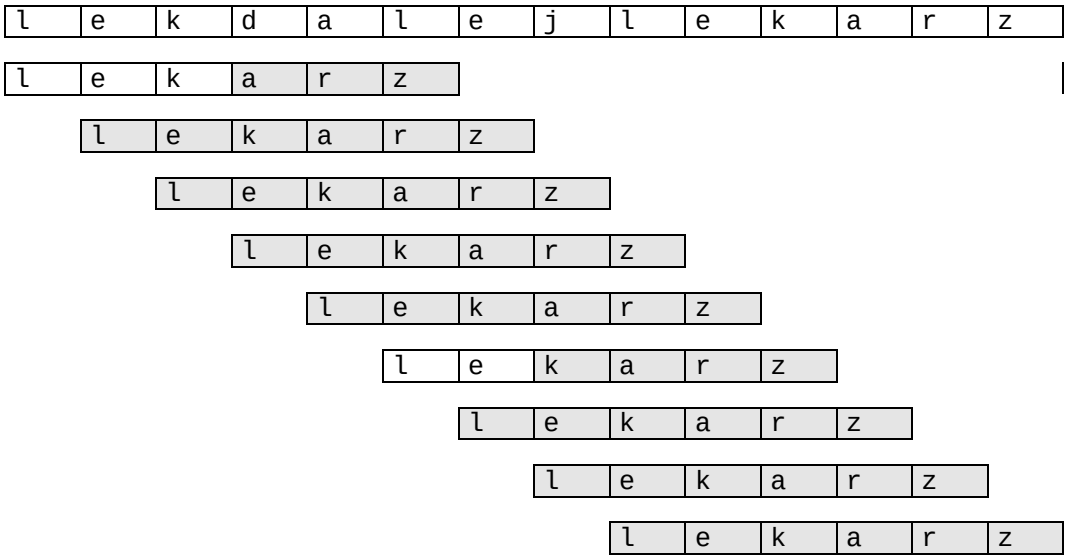
- Przykład: Napisać własną funkcję, która znajduje pierwsze wystąpienie wyrazu w podanym tekście.

  Sformułowanie zadania: Dany jest ciąg znaków (*tekst*) o długości *n*. Szukamy w nim ciągu znaków (*wzorzec*) o długości *m*.

  Tego typu problemy nazywane są *problemami wyszukiwania wzorca* (ang. *pattern matching*).

  Najprostszym algorytmem rozwiązującym to zadanie jest *algorytm naiwny* (ang. *straightforward searching*). Badamy każdą pozycję *k* tekstu i sprawdzamy, czy kolejne *m* znaków jest zgodnych ze wzorcem.

  Przykład: szukamy w tekście słowa *lekarz*




```
// wersja A - tablice
int SzukajA(char *tekst, int n, char *wzorzec, int m)
// Zwraca pozycję, od której rozpoczyna się wzorzec,
// jeżeli wzorca nie znaleziono zwracane jest -1
{
  int i, j, k;
  for (k = 0; k<n; k++) {
    for (j = 0, i = k; j<m; j++, i++)
      if (wzorzec[j] != tekst[i]) break;
      if (j == m) return k; // Wzorzec rozpoczyna się na pozycji k
  }
  return -1;
}
```

Komentarz: wersja A jest najprostsza, każdy znak w tekście może być początkiem wzorca.

```
// wersja B – zastąpienie tablicy wskaźnikami
int SzukajB(char *tekst, int n, char *wzorzec, int m) {
  char *t, *kt, *p, *kw, *q;
  t = tekst;  kt = tekst+n;  kw = wzorzec + m ;
  while(t < kt) {
    p = wzorzec;  q = t;
    while(++p < kw && *p == *++q) { cout << p << endl << q << endl << endl;
}
    if (p == kw && !*p) return t - tekst;
    t++;
  }
  return -1;
}
```

Komentarz: wersja B – zastosowanie wskaźników, tablica jest przeglądana sekwencyjnie, zwiększymy dzięki temu szybkość.

```
// Wersja C -
int SzukajC(char *tekst, int n, char *wzorzec, int m)
{
  char *t, *kt, *p, *kw, *q;
  char pierwszy = *wzorzec;

  t = tekst;  kt = tekst+n-m+1;  kw = wzorzec + m;
  while(1) {
    // Szukamy pierwszego znaku wzorca
    while (t < kt && *t != pierwszy) t++;
    if (t >= kt) break;
    // Znaleziono pierwszy znak wzorca, rozpocznij sprawdzanie
    p = wzorzec;  q = t;
    while(++p < kw && *p == *++q)
      {  }
    if (p == kw) return t - tekst;
    t++;
  }
  return -1;
}
```

Komentarz: W poprzednich wersjach przyjęto, że każdy znak w tekście może być początkiem wzorca, mimo, że pewne części tekstu można by było przeskoczyć, gdyż wiadomo, że nie będą one odpowiadały wzorcowi. W wersji C przyjęto, że algorytm można podzielić na dwie części:

- poszukiwanie w tekście pierwszego znaku, od którego rozpoczyna się wzorzec
- dopiero wtedy rozpoczęcie dokładnego sprawdzania

Dodatkowo uwzględniono fakt, że jeżeli w tekście pozostanie mniej znaków do sprawdzania, niż ma ich wzorzec, nie ma sensu kontynuowanie sprawdzania (koniec tekstu `kt=tekst+n –m +1`)

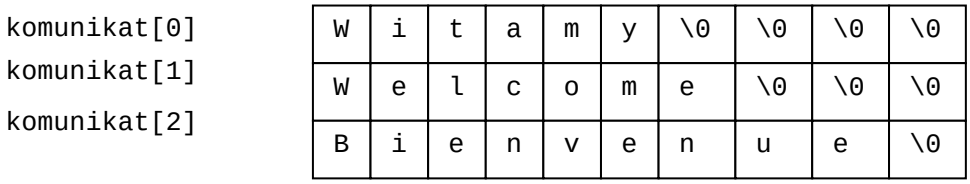
Tablice napisów

- Tablica napisów pozwala korzystać z różnych napisów wskazując je za pomocą indeksu.

Tablica wskaźników do napisów

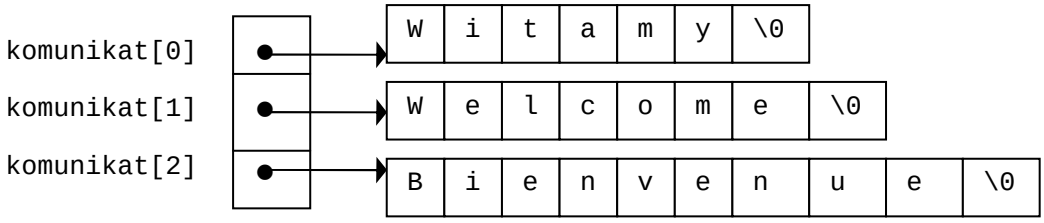
- Napisy można przechowywać w tablicy dwuwymiarowej

```
char komunikat[3][10] ={"Witamy",
                        "Welcome",
                        "Bienvenue"};
```



- Można również przechowywać napisy w postaci tablic jednowymiarowych, zaś dostęp do nich zapewnić za pomocą wskaźnika przechowywanego w tablicy wskaźników.

```
char *komunikat[3]={"Witamy",
                    "Welcome",
                    "Bienvenue"};
```



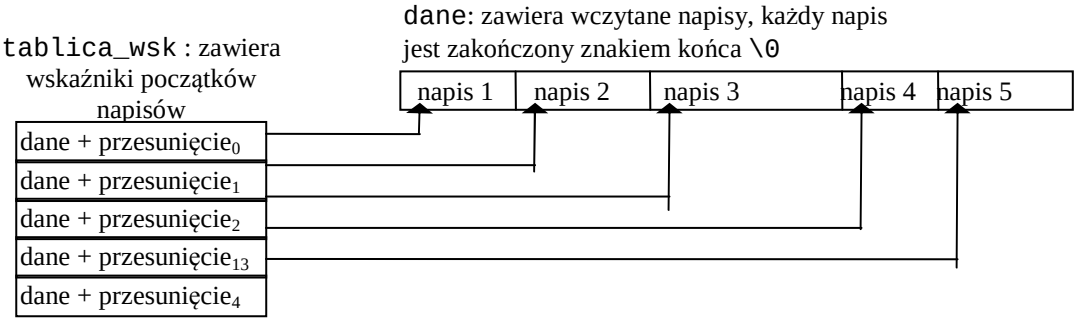
- Dostęp do napisu :

```
komunikat[1]="WElcome";
*(komunikat[1]+1)='e';
cout << komunikat[1];
```

- Przykład:

```
void blad(int numer)
{
  static char *komunikat[]={
  "Blad otwarcia pliku\n",
  "Blad odczytu\n",
  "Blad zapisu\n"
  };
  cout << komunikat[numer];
}
```

- Przykład wykorzystania tablicy wskaźników: program wczytuje nazwiska do bufora umieszczając je jedno za drugim. Uwaga: program *nie sprawdza* przepełnienia tablicy z nazwiskami.




```
#include <iostream.h>

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

int main()
{
  char *tablica_wsk[5], dane[100];
  int pozycja=0, i;
  for (i=0;i<5;i++)
  {
    cout << "Podaj nazwisko nr " << (i+1) <<":";
    cin.getline(dane+pozycja,100-i);
    tablica_wsk[i]=dane+pozycja;
    pozycja += strlen(tablica_wsk[i])+1;
  }
  cout << "Wczytane nazwiska:" << endl;
  for (i=0;i<5;i++)
    cout << "Nazwisko nr " << (i+1) << ":" << tablica_wsk[i] << endl;
  return 0;
}
```

Przekazywanie tablicy wskaźników do funkcji

- Przykład 1: program wczytuje 5 nazwisk zmiennej długości, umieszcza je w tablicy jeden za drugim, następnie je sortuje metodą sortowania bąbelkowego.
  Funkcja `sortowanie()` jako pierwszy argument otrzymuje tablicę wskaźników do początków nazwisk. Drugim argumentem jest liczba elementów w tablicy czyli liczba napisów do sortowania.
  Prototyp funkcji `sortowanie()` :

```
void sortowanie(char *napis[], int ile)
```

lub

```
void sortowanie(char **napis, int);
```

```
#include <iostream.h>
#include <string.h>
void sortowanie(char**, int);
int main()
{
  char *wsk_nazwiska[5], tbl_nazwiska[50];
  int wsk_rob=0;
  int i;

  for (i=0;i<5;i++)
  {
    cout << "Podaj nazwisko :";
    cin.getline(tbl_nazwiska+wsk_rob,10);
    wsk_nazwiska[i]=tbl_nazwiska+wsk_rob;
    wsk_rob += strlen(wsk_nazwiska[i])+1;
  }
  cout << "Przed sortowaniem: "<<endl;
  for (i=0;i<5;i++)
    cout << "Nazwisko: " << wsk_nazwiska[i] << endl;
  sortowanie(wsk_nazwiska, 5);
  cout << "Po sortowaniu: "<<endl;
  for (i=0;i<5;i++)
    cout << "Nazwisko: " << wsk_nazwiska[i] << endl;
  return 0;
}

void sortowanie(char *napis[], int ile)
{
  char *robocza;
  int i,j;

  for (i=0; i< ile-1; i++)
    for (j=(ile-1);i<j;j--)
    if (strcmp(napis[j-1],napis[j]) > 0)
    {
      robocza=napis[j-1];
      napis[j-1]=napis[j];
      napis[j]=robocza;
    }
}
```

```
///for(i=0; i<29000; i++)
        {
        for (j=i+1; j<30000; j++)
         if (tab[i]>tab[j]) {l=tab[i]; tab[i]=tab[j]; tab[j]=l;}
         }
```

Przykład 2: funkcja sprawdza, czy w tablicy znajduje się podane słowo i zwraca indeks do tego elementu tablicy, jeśli słowo nie zostało znalezione zwraca -1.

Wersja A: funkcja ma przetwarzać tablicę postaci:

```
      char const *tablica[]={
      "warszawa",
      "krakow",
      "gniezno",
      "poznan"
      };

int szukaj_slowa( char const *tablica[],
                  int const rozmiar, char const * const slowo )
{
  char const **wt;  // zmienna robocza

  for( wt = tablica; wt < tablica + rozmiar; wt++ )
  // jeśli słowo zgadza się z elementem tablicy,
  // zwróć indeks elementu
    if( strcmp( slowo, *wt ) == 0 )
      return wt - tablica;
  // nie znaleziono
  return -1;
}
```

Wersja B: funkcja ma przetwarzać tablicę postaci:

```
      char const *tablica[]={
      "warszawa",
      "krakow",
      "gniezno",
      "poznan",
      NULL
      };
```

Koniec tablicy jest oznaczony za pomocą wskaźnika pustego. Pozwoli on znaleźć koniec tablicy i nie trzeba do funkcji przesyłać rozmiaru tablicy.

```
int szukaj_slowa( char const *tablica[], char const * const slowo )
{
  char const **wt;  // zmienna robocza

  for( wt = tablica; *wt != NULL ; wt++ )
  // jeśli słowo zgadza się z elementem tablicy,
  // zwróć indeks elementu
    if( strcmp( slowo, *wt ) == 0 )
      return wt - tablica;
  // nie znaleziono
  return -1;
}
```