Stałe i zmienne znakowe.

Stała znakowa: ‘znak’

Na przykład: ‘a’, ‘1’, ‘0’ ‘c’

Każdy znak jest reprezentowany w pamięci przez swój kod. Kody alfanumerycznych znaków ASCII – to liczby z przedziału [32, 127]. Liczby od 128 do 255 też są kodami znaków, ale znaki te mogą być różnie zdefiniowane – nie odpowiada to standardu ASCII.

Typ danych dla przechowywania pojedynczych znaków – char.

sizeof(char) -> 1 B

Na przykład,

```
char i = 32;    //i – to jest znak spacji – reprezentacje przez kod znaku
char i = ‘ ‘;    //to jest też znak spacji – reprezentacje przez znak
```

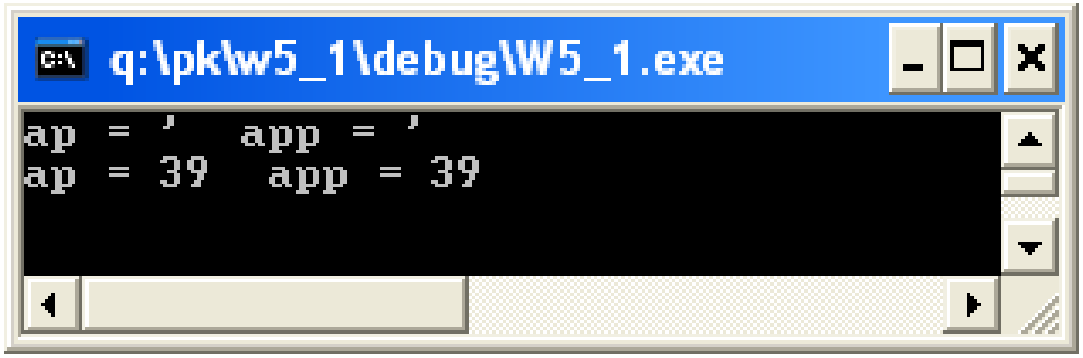
Apostrof jako znak należy przedstawiać jako \ ’ (za pomocą specjalną sekwencje znaków)

Może wystąpić konieczność uszycia typu int dla reprezentacji zmiennej znakowej. Na przykład:

```
int ch;
while((ch = getchar()) != EOF)
        ;
```

EOF = -1 – nie może być przedstawiona niezawodnie typem char

```
char ap = '\'';          // to jest znak apostrofa \’ – typ char
int app = '\'';          //  to jest kod znaku apostrofa \’ – typ int
printf("ap = %c  app = %c \n", ap, app);  //to jest wydruk znaków
printf("ap = %d  app = %d \n", ap, app); //to jest wydruk kodów znaków
```



Przykład w5_1

Tablice znakowe. Stałe tekstowe

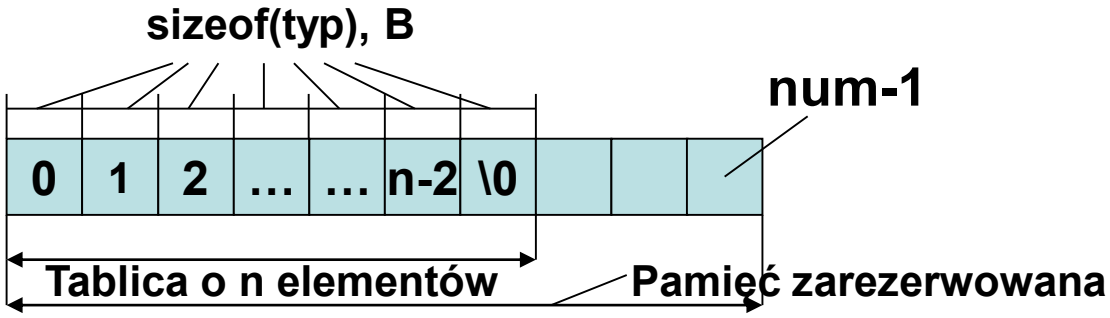
Tablice znakowe są używane dla zapisu ciągu znaków w pamięci komputera. Ciąg znaków (wiersz tekstowy) jest umieszczony w kolejnych bajtach pamięci komputera. Każdy znak tego ciągu – to jest odpowiedni element tablicy znakowej.

Definicja tablicy znakowej:

char id_tablicy[num];

id_tablicy – imię tablicy (identyfikator)
num - maksymalna liczba znaków minus jeden

- **Każdy ciąg znaków kończy się symbolem \0 „koniec ciągu znaków”, który zajmuje 1 bajt – dla tego „minus jeden”. Przy przekroczeniu ilości znaków wartości num-1 występuje błąd – pisanie po pamięci (page fault).**



- **Elementy tablicy są ponumerowane od 0 do n-1:**

id_tablicy[0], id_tablicy[1], ..., id_tablicy[n-2], id_tablicy[n-1]

(id_tablicy[n-1] -> ‘\0’) Pamięć zarezerwowana na num elementów

- **Numery elementów tablicy będziemy nazywali indeksami.**

- **Para nawiasów kwadratowych – to jest operator [ ] . (id_tablicy[k] - odwołanie do elementu tablicy o indeksie k).**

- **Rozmiar tablicy w definicji – musi być stała.**

- **W każdym elemencie tablicy można zapisać jeden znak:**

```
id_tablicy[0] = ‘a’;
id_tablicy[1] = ‘b’;
.....
```

- **Ostatnim symbolem ciągu znaków mysi być ‘\0’**

Stałe tekstowe

Dowolny ciąg znaków, zapisany w postaci “to jest stała tekstowa” , jest stałą tekstową.

Jeśli w jednym z tych znaków ma być cudzysłów, to należy zapisać \”

Kompilator umieszcza tekst w odpowiedniej grupie bajtów (1 B na symbol). W końcu tekstu (stałą tekstową) jest dopisany symbol \0 .

Podany tekst ma 22 symboli (razem ze spacjami), więc w pamięci komputera będzie zajmował 23 B razem z \0 .

Inicjalizacia zmiennej znakowej

Przykład:

```
const char strrr[256] = “C:\\Documents and Settings\\MY_APPLIC”;
char str[ ]   = “abcd”;
size_t len, len1;
len   = sizeof(str);   //  len = 5 !          +  \0
len1 = strlen(str);   //  len1 = 4     size_t strlen(char *) – zwraca ilość
                      // znaków w wiersze tekstowym bez \0

str[2] = ‘  ‘;              //str --> ab d

strrr[6] = ‘j’;             //błąd – to jest const char
```

Przykład W5_2

Wskaźnik do tablicy

- **Definicja tablicy rezerwuje odpowiednią grupę bajtów w pamięci i przypisuje identyfikatorowi id_tablicy adres zerowego elementu.**
- **id_tablicy wskazuje do zerowego elementu tablicy.**
- **Nie wolno zmieniać wartości id_tablicy (to jest stała - wskaźnik na początek tablicy).**
- **id_tablicy nie jest L-walue, nie można używać po lewej strone instrukcji przypisania.**

Przykład:

```
char str1[ ] = "abcde";  //debuger information: str1 = 0x0012fe44 "abcde"
                         //str1[0] = 97 ‘a’,      &str1[0]  0x0012fe44 "abcde"
                         //str1[1]  = 98 ‘b’      &str1[1] 0x0012fe45 "bcde"
                         ..........................................................
```

Funkcja

char *gets(char *_buffer_);

- **Argument – identyfikator tablicy znakowej.**
- **Napisany na klawiaturze tekst (stream stdin) po wciśnięciu ENTER zostanie zapisany w tablice – jeden znak w jednym elemencie tablicy, i na końcu będzie dopisany \0 . Kopiowanie odbywa się dopóki nie spotkamy \n albo EOF. Symbol \n nie dopisuje się do *buffer*. Zwraca wskaźnik do *buffer* w przypadku powodzenia i *NULL* w przeciwnym przypadku.**

Fragment kodu:

```
char str[256];
char *ptr = gets(str);
                    //Przy wyjściu z gets str zawiera ciąg znaków,
                    //wprowadzonych z monitora, ptr – to jest wskaźnik
                    //do str:
```

```
str     0x0012fe60   "abcdef"   char [256]
ptr     0x0012fe60   "abcdef"   char *
```

Przykład W5_3

Fragment kodu:

```
char *ptr = NULL;          //ptr   0x00000000   <Bad Ptr> char *
{
        ptr = “abcdefg”;    //ptr   0x0041563c   "abcdefg" char *
}
```

//Uwaga! Pamięć jest wydzielona w stosie bloku. Po wejściu z //bloku ta informacja nie będzie chroniona.

//Kod poprawny:

```
{
   ptr = (char *)malloc(256*sizeof(char)); //dynamiczne alokowanie pamięci w heap
   if(!ptr)
   {
        printf(“błąd \n”);    exit(1);
   }
   sprintf(ptr, “abcdefg”);
}
.....
if(ptr) free(ptr);    ptr = NULL;  //ptr jest już nie potrzebne – zwolni pamięć!
```

Funkcja

int sprintf(char *_buffer_, const char *_format_ [, _argument_] ...);

Formatuje i przechowuje ciąg znaków i liczby w buforze _buffer_ . Działa analogicznie funkcji printf, fprintf.

<u>Operacji ze wskaźnikami (przykłady):</u>

```
int i = 3;
char *ptr = NULL;
char str[ ] = “abcdef”;
char symb;
ptr = &str[0];       //wskazuje na element str[0]
.....................................
ptr++;               //wskazuje na element str[1] – przesuwa wartość wskaźnika
                     //o sizeof(char) w prawo;
ptr- -;              //wskazuje na element str[0]; - przesuwa wartość wskaźnika
                     //o sizeof(char) w lewo;
symb = *(ptr+i); //symb = ‘d’ (dla tego że ptr+i wskazuje do elementu o
                    //indeksie 3 tablicy str.
```

Dla wskaźnika o typie dowolnym:

```
int i[50];          //def. tablice typu int, 50 elementów
double a[200];      //def. tablice typu double, 200 elementów

//i - wskaźnik, który wskazuje na adres pierwszego bajta zerowego
// elementu tablicy, lub to samo:  &i[0]
//   i          0x0012fe7c    int [50]
//   &i[0]      0x0012fe7c    int *

   i++;          //Błąd!  i -  identyfikator tablicy!
   int *ptr_i = &i[0]; //tworzymy wskaźnik ptr_i i ustalamy jego do początku tabl.
   ptr_i++;    //wskazuje na i[1] o adresie &i[0] + sizeof(int)

   double *ptr_a = &a[0];
   ptr_a++;   //wskazuje do a[1] o adresie &a[0] + sizeof(double)

   ptr_a = &a[0];
   double bb = *(ptr_a+16); //ptr_a+16 wskazuje do elementu tablicy a o
                            //indeksie 16 (początek numeracji indeksów od 0 !)
```

Instrukcje switch, case

```
switch(wyrażenie_s)
{
        case wyrażenie stałe 1:
                ciąg instrukcji 1
        case wyrażenie stałe 2:
                ciąg instrukcji 2
 ..........................................
        case wyrażenie stałe n:
                ciąg instrukcji n
        default:
                ciąg instrukcji default
};
```

wyrażenie_s – wyrażenie o wartościach całkowitych
wyrażenie stałe i - wyrażenie o wartościach całkowitych

1. Obliczenie wartości *wyrażenie_s.*
2. Jeśli *wyrażenie_s* == *wyrażenie stałe k* – *wykonanie* ciągu instrukcji k, ciąg instrukcji k+1,..., ciąg instrukcji n, goto 3; jeśli *wyrażenie_s* != *wyrażenie stałe k* - ciąg instrukcji default, goto 3
3. Pierwszy operator poza blokiem switch

Instrukcje switch, case

```
switch(wyrażenie_s)
{
        case wyrażenie stałe 1:
                ciąg instrukcji 1
        case wyrażenie stałe 2:
                ciąg instrukcji 2
    ..........................................
        case wyrażenie stałe n:
                ciąg instrukcji n
};
```

default jest pominięty:

1. **Obliczenie wartości *wyrażenie_s.***
2. **Jeśli *wyrażenie_s* == *wyrażenie stałe k* – *wykonanie* ciąg instrukcji k, ciąg instrukcji k+1,..., ciąg instrukcji n; pierwszy operator poza blokiem switch. Jeśli *wyrażenie_s* != *wyrażenie stałe k* - pierwszy operator poza blokiem switch.**

Ostatnim operatorem ciągu instrukcji *k* może być break; break przerywa działanie switch i przekazuje sterowanie do pierwszego operatora poza blokiem switch

Przykłady

```
switch (c)
{
      case 'A':
      case 'a':
                  capa++;
                  lettera++;
            break;
      case 'f':
                  capa += 10;
            break;
      default: total++;
};
```

```
switch( i )
{
      case -1:
                        n++;
             break;
      case 0 :
                        z++;
            break;
      case 1 :
                        p++;
            break;
};
```

Przykłady

```
enum KIERUNEK_STUDIOW
{
      KIERUNEK_MATEMATYKA,     //0
      KIERUNEK_INFORMATYKA,    //1
      ILOSC_KIERUNKOW           //2
};
```

```
enum KIERUNEK_STUDIOW
{
      KIERUNEK_MATEMATYKA = 10,     //10
      KIERUNEK_INFORMATYKA = 20,    //20
      NASTEPNY_KIERUNEK             //21
};
```

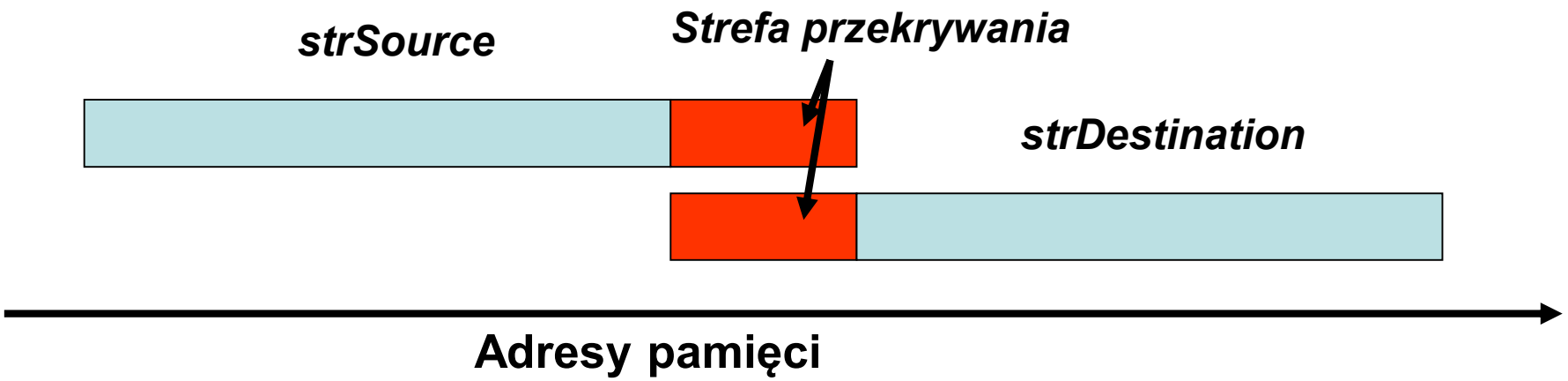
```
enum  KIERUNEK_STUDIOW key = KIERUNEK_INFORMATYKA;
char str[256] = ” ”;

switch(key)
{
   case KIERUNEK_MATEMATYKA: sprintf(str, “matematyka”);
      break;
   case KIERUNEK_INFORMATYKA: sprintf(str, “informatyka”);
      break;
   default:
            sprintf(str, “kierunek nie wybrany”);
};
```

Funkcje strcpy, strcat (#include <string.h>)

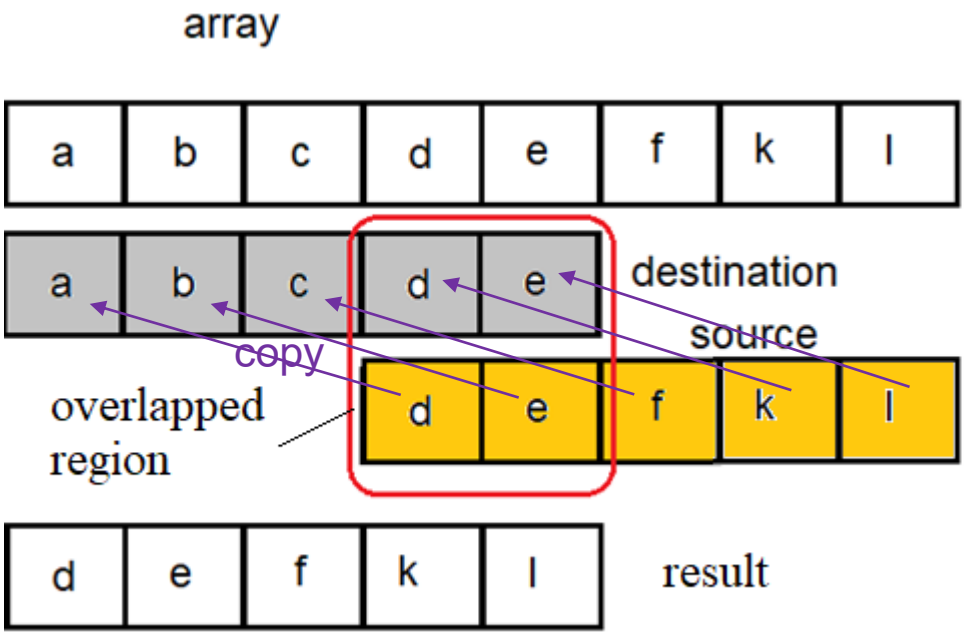
char *strcpy(char *strDestination, const char *strSource);

Kopiuje ciąg znaków (string) *strSource*, zakończoną symbolem \0, w bufor (string), oznaczony jako *strDestination*. Zachowanie funkcji nie jest wyznaczone, jeśli bufory *strSource* i *strDestination* wzajemnie się przekrywają.

Zwraca wskaźnik na *strDestination*. Nie istnieje wartości, która wskazywała by na błędne zakończenie.



Typowy przykład




```
char str[] = "abcdefkl";
strcpy(str, str+3);
```

char *strcat(char *strDestination, const char *strSource);

Dopisuje ciąg znaków (string) *strSource* w koniec bufora (string) *strDestination* i zamyka symbolem \0. Zachowanie funkcji nie jest wyznaczone, jeśli bufory *strSource* i *strDestination* wzajemnie się przekrywają.

Zwraca wskaźnik na *strDestination*. Nie istnieje wartości, która wskazywała by na błędne zakończenie.

Przykłady: w kolorze czarnym są podane elementy tablic przekazywanych do funkcji *strcpy*. Markier \0 tablicy *str2* po wkopiowaniu do tablicy *str1* spowoduje że wszystkie elementy tablicy *str1* umieszczone sprawa od markiera, będą nieosiągalne.

char str1[] = “abcdefgk”;
char str2[] = “123456”; abcdefgk\0
strcpy(str1, str2); //wynik: 123456\0 → str1: 123456

char * str1[] = “abcdefgk”;
char * str2[] = “123456”; abcdefk\0
strcpy(str1, str2 + 4); //wynik: 123456\0 → str1: 56

char * str1[] = “abcdefgk”;
char * str2[] = “123456”; abcdefk\0
strcpy(str1 + 2, str2 + 4); //wynik: 123456\0 → str1: ab56

Przykład:

```
#include <string.h>
#include <stdio.h>
int main( void )
{
    char string[80];
    strcpy( string, "Hello world from " );
    strcat( string, "strcpy " );
    strcat( string, "and " );
    strcat( string, "strcat!" );
    printf( "string = %s\n", string );
}
```

Wydruk: ***string = Hello world from strcpy and strcat!***

Przykład: ***Lab_4 (wiersze tekstowe)***

Funkcje strcmp: (#include <string.h>)

int strcmp(const char *_string1_, const char *_string2_);

porównuje ciągi znaków _string1_ i _string2._ Zwraca RetVal typu int.

RetVal: < 0 - string1 < string2
= 0 - string1 == string2
> 0 - string1 > string2

string1 == string2, jeśli każdy znak z string1 jest równy odpowiedniemu znaku string2.

Relacji nierówności.
W C jest przyjęte że string1 > string2, jeśli pierwszy symbol string1 ma kod, większy od kodu pierwszego symbolu string2. Jeśli pierwsze symbolu są równe, porównujemy drugie. I tak dalej.

Przykład:

```
   char ss1[ ] = "abcdef";
   char ss2[ ] = "za";
   int ret = strcmp(ss1, ss2);  //ret = -1,  kod ‘a’ = 97, kod ‘z’ = 122 -> „a < z”
.....

   char ss1[ ] = "abcdef";
   char ss2[ ] = "Abcdef";
   int ret = strcmp(ss1, ss2);  //ret = 1, kod ‘a’ = 97,  kod ‘A’ = 65 -> „a > A”
.....

   char ss1[ ] = "abcde^";
    char ss2[ ] = "abcdef";
    int ret = strcmp(ss1, ss2);  //ret = -1, kod ‘^’ = 94, kod ‘f’ = 102 -> „^ < f”
.....

   char ss1[ ] = "ABCDE^";
   char ss2[ ] = "ABCDEF";
   int ret = strcmp(ss1, ss2);   //ret = 1, kod ‘^’ = 94, kod ‘F’ = 70 -> „^ > F”
```