Podstawy programowania 1

Łańcuchy znaków

Arkadiusz Chrobot

Katedra Systemów Informatycznych

27 listopada 2024

Plan

1. Operacje na pojedynczych znakach
2. Typ danych dla ciągów znaków
3. Inicjacja łańcuchów
4. Wprowadzanie i wypisywanie ciągu
5. Operacje na łańcuchach
6. Konwersje
7. Zakończenie

Operacje na pojedynczych znakach

Funkcja `getchar()`

Język C udostępnia zbiór funkcji, które nazywamy *standardową biblioteką*. Wśród nich są funkcje, które wykonują określone operacje na pojedynczych znakach reprezentowanych za pomocą wartości kodu ASCII i przechowywanych w zmiennych typ `char`. Język C umożliwia też pracę ze znakami zapisanymi w innych kodach, np. należących do rodziny kodów UTF, ale ten temat nie będzie poruszany na tym wykładzie. Aby wczytać pojedynczy znak z klawiatury można użyć funkcji `getchar()`, która dostępna jest po włączeniu do kodu programu pliku nagłówkowego `stdio.h` Zwraca ona kod ASCII znaku, ale jako wartość typu `int`, a nie `char`. Nie przyjmuje ona żadnych argumentów wywołania.

Operacje na pojedynczy znakach

Funkcje z pliku `ctype.h`

Dużo funkcji operujących na pojedynczych znakach jest dostępnych po włączeniu pliku nagłówkowego `ctype.h`. Wszystkie one przyjmują jako argument znak (kod ASCII), w postaci wartości typu `int` i zwracają wartość typu `int`. Tabela poniżej zawiera opis niektórych z nich.

Prototyp funkcji	**Opis działania**
`int tolower(int c)`	Jeśli znak przekazany jej przez parametr jest wielką literą, to zwraca odpowiadającą jej małą literę.
`int toupper(int c)`	Jeśli znak przekazany jej przez parametr jest małą literą, to zwraca odpowiadającą jej wielką literę.
`int isalnum(int c)`	Jeśli znak jest literą lub cyfrą to zwraca wartość różną od zera, w przeciwnym przypadku zero.

Operacje na pojedynczy znakach

Funkcje z pliku `ctype.h` - kontynuacja

Prototyp funkcji	**Opis działania**
`int isalpha(int c)`	Jeśli znak jest literą to zwraca wartość różną od zera, w przeciwnym przypadku zero.
`int isdigit(int c)`	Jeśli znak jest cyfrą to zwraca wartość różną od zera, w przeciwnym przypadku zero.
`int isspace(int c)`	Jeśli znak jest białym znakiem (spacją, tabulacją, tabulacją pionową itd.) to zwraca wartość rożną od zera w przeciwnym przypadku zero.
`int islower(int c)`	Jeśli znak jest małą literą to zwraca wartość różną od zera, w przeciwnym przypadku zero.
`int isupper(int c)`	Jeśli znak jest wielką literą, to zwraca wartość różną od zera, w przeciwnym przypadku zero.

Operacje na pojedynczych znakach

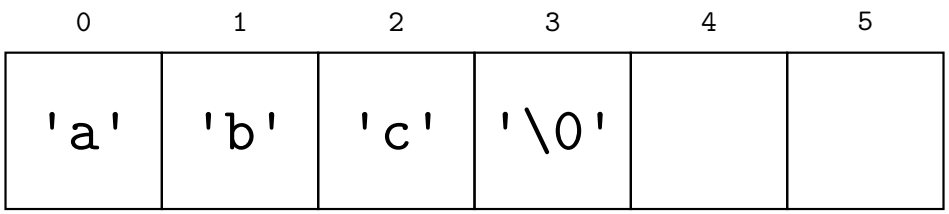
Przykład implementacji

Większość funkcji, które były zaprezentowane na poprzednich slajdach jest łatwa do implementacji. Poniżej zaprezentowana jest funkcja, która przyjmuje przez parametr znak i jeśli jest on literą, to zmienia jego wielkość z wielkiej na małą lub odwrotnie. Aby zrozumieć zasadę jej działania wystarczy wiedzieć, że kody ASCII odpowiadających sobie wielkich i małych liter różnią się ustawieniem szóstego bitu ($2^5 = 32$). W kodach ASCII wielkich liter jest on ustawiony na 0, a dla małych jest ustawiony na 1.

```
char set_upper_or_lower(char input)
{
    if((input>='a'&&input<='z')||(input>='A'&&input<='Z'))
       input ^= 32;
    return input;
}
```

Typ danych dla ciągów znaków

Oprócz pojedynczych znaków komputery mogą efektywnie przetwarzać *łańcuchy znaków* (ang. *string*), nazywanych też *ciągami znaków*. W przeciwieństwie do innych języków programowania, język C **nie posiada** osobnego typu danych do reprezentowania takich wartości. Ciąg znaków jest przechowywany i przetwarzany w tablicy elementów typu `char`. Ponieważ nie wszystkie elementy tej tablicy muszą być zajęte przez znaki należące do ciągu, to każdy ciąg jest zakończony specjalnym znakiem o kodzie ASCII równym 0 nazywanym znakiem końca łańcucha (ciągu). Ten znak jest zapisany w notacji języka C jako `'\0'`. Rysunek ilustruje przykładową sześcioelementową tablicę przechowującą ciąg znaków *abc*.



Inicjacja łańcuchów

Tablicę znaków można zainicjować w miejscu jej deklaracji w ten sam sposób jak zwykłą tablicę, podając wartości jej elementów w nawiasach klamrowych i pamiętając o tym, aby ostatni element miał wartość `'\0'`. Jednakże taka notacja jest bardzo niewygodna. Prościej jest takiej tablicy przypisać ciąg znaków ujęty w cudzysłów. Jeśli taką wartość przypisujemy do tablicy i jednocześnie określamy długość tej tablicy, to musimy pamiętać, że musi ona mieć o jeden element więcej niż jest znaków w tym łańcuchu, aby pomieścić znak końca ciągu. Lepiej w takim przypadku nie określać liczby elementów tablicy - kompilator ustali ją samodzielnie. Można również łańcuch znaków przypisać do wskaźnika typu `char *`. Jednak w takim przypadku próba modyfikacji tego łańcucha w trakcie wykonania programu skończy się błędem wykonania i działanie programu zostanie przerwane.

Inicjacja łańcuchów

Pzykłady

```
int main(void)
{
        char first_string[] = {'P','r','z','y','k','ł','a','d','\0'};
        char second_string[] = "Przykład";
        char *third_string = "Przykład";
        char fourth_string[10];

        second_string[1] = 'z';
        // third_string[1] = 'z'; // To jest zabronione.

        return 0;
}
```

W przykładzie wszystkie tablice są zmiennymi lokalnymi, ale mogą być również globalne. Można je również przekazywać przez parametry do funkcji, jak inne tablice. Zmienna `fourth_string` nie jest zainicjowana, ale można w niej przechowywać ciągi znaków o maksymalnej liczbie 9 znaków. Można również zmienić kod przykładu i przypisać jej ciąg znaków w miejscu deklaracji.

Wprowadzanie i wypisywanie ciągu

Łańcuchy znaków ujęte w cudzysłów można wypisywać na ekran bezpośrednio za pomocą funkcji `printf()` i `puts()`. Jeśli łańcuch jest zapisany w tablicy znaków, to możemy go wypisać używając pierwszej z wymienionych funkcji i stosując specyfikację konwersji `%s`. Ten sam ciąg można zastosować do wczytania ciągu z klawiatury do tablicy przy pomocy funkcji `scanf()`. Należy jednak pamiętać, że ta funkcja odczytuje znaki tylko do napotkania pierwszego znaku białego. Jeśli zatem chcielibyśmy odczytać całe zdanie, to musimy posłużyć się ciągiem formatującym `"%[^\n]s"`. W nawiasie kwadratowym podaje się zbiór znaków, które ma akceptować funkcja `scanf()`. Zapis `^\n` oznacza „wszystkie znaki oprócz znaku nowego wiersza (klawisza Enter)”. Podobny efekt można uzyskać stosując funkcję `fgets()`. W przypadku odczytu z klawiatury pierwszym argumentem jej wywołania jest tablica znaków, do której ma być wczytany ciąg, drugim jest liczba elementów tablicy, a trzecim zmienna `stdin`. Funkcja zwraca w przypadku niepowodzenia odczytu wartość `NULL`, a w przypadku powodzenia wskaźnik na tablicę.

Wprowadzanie i wypisywanie ciągów

Przykład

```c
#include<stdio.h>

int main(void)
{
        char str[40];

         scanf("%s",str); // Wczytanie ciągu do pierwszego napotkanego znaku białego.
         while(getchar()!='\n'); // Usunięcie znaków, łącznie
                                 //z \n ze strumienia wejściowego.
         scanf("%[^\n]s",str); // Wczytanie całego ciągu znaków.
         while(getchar()!='\n'); // Usunięcie znaku \n ze strumienia wejściowego.
         fgets(str,40,stdin); // Wczytanie całego ciągu znaków.

         return 0;
}
```

Zmienna `stdin` to strumień wejściowy, czyli wskaźnik na plik. Proszę zauważyć, że do funkcji `scanf()` i `fgets()` przekazywana jest nazwa tablicy, która jest jednocześnie wskaźnikiem, zatem nie trzeba jej poprzedzać operatorem wyłuskania adresu.

Wprowadzanie i wypisywanie ciągów

Przykład

```c
#include <stdio.h>


int main(void)
{
    char string[11];
    scanf("%10[^\n]s",string);
    printf("%s\n",string);
    return 0;
}
```

Ten przykład pokazuje, jak ograniczyć liczbę znaków odczytywanych z klawiatury przy pomocy funkcji `scanf()`. Bez tego ograniczenia funkcja ta pozwala zapisywać poza obszarem przekazanej jej jako argument wywołania tablicy, co jest bardzo niebezpiecznym działaniem.

Operacje na łańcuchach

Tablice znaków nie mogą być porównywane za pomocą zwykłych operatorów relacyjnych, takich jak np. ==. Ich zastosowanie w stosunku do tych tablic pozwoli jedynie na porównanie adresów tych tablic, a nie ich zawartości. Nie można również przypisać takim tablicom, poza przypadkiem inicjacji, wartości za pomocą operatora =. Te operacje mogą być zrealizowane jedynie poprzez dostęp do poszczególnych elementów tablic znakowych. Aby ułatwić ich przeprowadzenie twórcy standardów języka C przewidzieli do tego celu odpowiednie funkcje biblioteki standardowej języka, które stają się dostępne w programie po włączeniu do jego kodu źródłowego plików nagłówkowych `strings.h` i `string.h`. W dalszej części wykładu zostanie przedstawiona część funkcji zadeklarowana w tym drugim pliku.

Operacje na łańcuchach

Przykład użycia	Opis
`unsigned long int a = strlen(string);`	Funkcja zwraca liczbę znaków łańcucha w przekazanej jej przez argument wywołania tablicy znaków. Pomijany jest znak końca łańcucha.
`strcpy(string_2, string_1);`	Funkcja kopiuje łańcuch znaków przekazany jej jako drugi argument do tablicy przekazanej jej jako pierwszy argument. Zwracany przez nią wskaźnik na łańcuch docelowy jest najczęściej ignorowany.
`strncpy(string_2, string_1, number);`	Jak wyżej, ale funkcja kopiuje maksymalnie tylko `number` pierwszych znaków. Zapobiega to niebezpiecznej sytuacji, gdy kopiowany łańcuch ma więcej znaków, niż może pomieścić `string_2`.

`if(strcmp(string_1,string_2)==0) {`
`…`
`} else {`
`…`
`}`
	Funkcja porównuje dwa ciągi znaków i zwraca wartość mniejszą od zera, jeśli `string_1` jest mniejszy od `string_2`, większą od zera, jeśli zachodzi relacja odwrotna lub równą zero, jeśli ciągi są sobie równe. Łańcuchy porównywane są znak po znaku, aż znaleziona zostanie różniąca się para i porównane zostaną kody ASCII należących do niej znaków, lub jeden z ciągów będzie krótszy.

Operacje na łańcuchach

Przykład użycia	Opis

`if(strncmp(string_1,string_2,length)==0) {`
`…`
`} else {`
`…`
`}`
	Funkcja działa tak jak `strcmp()`, ale porównuje maksymalnie tylko tyle znaków, ile określi argument `length`. Jest to zabezpieczenie przed przekroczeniem zakresu, któregoś z porównywanych łańcuchów.

`strcat(string_1,string_2);`	Funkcja dołącza łańcuch zawarty w drugim argumencie do łańcucha zawartego w pierwszym argumencie i zwraca wskaźnik na ten pierwszy łańcuch. Najczęściej wartość przez nią zwracana jest ignorowana.
`strncat(string_1,string_2,length);`	Funkcja działa podobnie jak `strcat()`, ale dołącza maksymalnie tylko tyle znaków drugiego łańcucha, ile określa trzeci argument wywołania.
`char *result = strstr(string,pattern);`	Funkcja wyszukuje pierwsze wystąpienie wzorca `pattern` w łańcuchu `string`. Zwraca wskaźnik na element łańcucha, który zawiera pierwszą literę wzorca.

Operacje na łańcuchach

Przykład użycia	Opis
`char *result = strtok(string,delimiters);`	Funkcja służy do podziału łańcucha `string` na mniejsze części według znaków umieszczonych w łańcuchu `delimiters`. Jej sposób użycia jest dosyć skomplikowany, dlatego w dalszej części wykładu zaprezentowany zostanie przykład jej wykorzystania.
`char *result = strchr(string,character);`	Funkcja zawraca wskaźnik na element łańcucha `string`, który zawiera pierwsze wystąpienie znaku `character`. Drugi parametr funkcji jest typu `int`.
`char *result = strrchr(string,character);`	Funkcja działa podobnie jak `strchr()`, ale wyszukuje ostatnie wystąpienie znaku w ciągu.

Operacje na łańcuchach

Podsumowanie

W tabeli przedstawiono jedne z najczęściej używanych w języku C funkcji do wykonywania operacji na ciągach znaków. Jeśli funkcja dysponuje zamiennikiem, który za pomocą argumentów wywołania pozwala ograniczyć liczbę znaków, na jakich ona operuje (np. `strcpy()` i `strncpy()`) to zaleca się stosowanie tego zamiennika. Jego działanie jest zazwyczaj bardziej bezpieczne.
Wszystkie funkcje opisane w tabeli można zaimplementować samodzielnie, wiedząc, że do łańcuchów znaków można odwoływać się jak do zwykłych tablic. Wiele takich implementacji znajduje się w książce B. W. Kernighana i D. M. Ritchiego pt. „Język ANSI C. Programowanie". Na wykładzie też zostaną przedstawione implementacje wybranych funkcji działających na ciągach znaków. Wcześniej zostanie zaprezentowany dokładniejszy opis działania funkcji `strcmp()`.

Operacje na łańcuchach

Funkcja `strcmp()`

Tak jak napisano w opisie tej funkcji porównuje ona ze sobą dwa łańcuchy znaków przekazane jej jako argumenty wywołania. To porównanie zaczyna się od pierwszej pary znaków w obu ciągach. Jeśli kod ASCII pierwszego jest większy od kodu ASCII drugiego, to funkcja zwraca wartość większą od zera i tym samym pierwszy ciąg jest uznawany za „większy". W odwrotnej sytuacji funkcja zwraca wartość ujemną i pierwszy ciąg uznawany jest za „mniejszy". Jeśli oba znaki są równe, to funkcja sprawdza kolejne pary. To sprawdzanie może się zakończyć na jeden z kilku sposobów:

1. Oba ciągi zawierają tyle samo znaków i wszystkie te znaki są takie same - funkcja zwraca zero, co oznacza, że ciągi są identyczne.
2. Jeśli pierwszy ciąg jest przedrostkiem drugiego, to uznawany jest on za mniejszy.
3. Funkcja znajduje różniącą się parę i zwraca wartość w opisany wcześniej sposób.

Operacje na łańcuchach

Funkcja `strlen()`

Działanie tej funkcji, jest stosunkowo proste - szuka ona znaku końca ciągu (`'\0'`) i zlicza ile znaków od początku tablicy „minęła” zanim go znalazła. To pozwala ustalić jej liczbę znaków w tym ciągu. Następny slajd zawiera przykładową implementację tej funkcji, ale pod inną nazwą.

Operacje na łańcuchach

Realizacja `strlen()`

```
unsigned int string_length(char *string)
{
    unsigned int i;
    for(i=0;string[i];i++)
        ;
    return i;
}
```

Operacje na łańcuchach

Realizacja `strlen()` - komentarz

Zaprezentowany na poprzednim slajdzie kod jest jedną z możliwości implementacji tej funkcji. Realizacja z użyciem arytmetyki wskaźników została opisana we wspomnianej wcześniej książce. Proszę zwrócić uwagę, że w zaprezentowanej funkcji większość działania wykonywana jest w pętli `for`.

Operacje na łańcuchach

Funkcja `strncpy()`

Ta funkcja kopiuje zawartość drugiego przekazanego jej jako argument wywołania łańcucha do pierwszego, ale maksymalnie tylko tyle znaków, ile zostało określonych przez trzeci argument. Jako ten argument należy przekazać liczbę znaków, które można umieścić w tablicy docelowej. Dzięki temu unikniemy sytuacji, w której znaki będą zapisywane poza tablicą, co może doprowadzić do nieprawidłowego działania programu. Dalej zaprezentowano dwie wersje tej funkcji o zmienionej nazwie i kolejności dwóch pierwszych parametrów.

Operacje na łańcuchach

Realizacja strncpy() - pierwsza wersja

```
void string_copy(char source[], char destination[], int length)
{
    int i = 0;
    while(length!=0 && source[i]!='\0') {
        destination[i]=source[i];
        i++;
        length--;
    }
    destination[i]='\0';
}
```

Operacje na łańcuchach

Komentarz do pierwszej wersji `strncpy()`

Kopiowanie odbywa się w pętli `while`, gdzie znak po znaku przepisywany jest łańcuch z tablicy `source` do tablicy `destination`. Pętla ta może zakończyć się na dwa sposoby. Zmienna `length` może osiągnąć wartość zero lub w odwiedzanym elemencie tablicy `source` może znajdować się znak końca łańcucha. W każdym przypadku, po zakończeniu pętli, w i-tym elemencie tablicy `destination` trzeba umieścić znak końca łańcucha.

Operacje na łańcuchach

Realizacja `strncpy()` - wersja z użyciem arytmetyki wskaźników

```
void string_copy(char *source, char *destination, int length)
{
    destination[length]='\0';
    while((length--)&&(*destination++=*source++))
        ;
}
```

Operacje na łańcuchach

Komentarz do drugiej wersji `strncpy()`

Zaprezentowana na poprzednim slajdzie druga wersja `strncpy()` jest krótsza od swojej poprzedniczki dzięki zastosowaniu arytmetyki wskaźników i kilku własności składni języka C. Po pierwsze wykorzystano w niej fakt, że każda wartość różna od zera jest traktowana jako prawda, a równa zero jako fałsz. Dzięki temu nie jest konieczne przyrównywanie wyrażeń `length--` i `*destination++=*source++` do zera. Po drugie, skorzystano ze skróconych obliczeń stosowanych dla operatora `&&`, dzięki czemu jeśli pierwsze wyrażenie jest fałszywe, drugie nie jest już wykonywane. Po trzecie zastosowano dostęp do tablicy za pomocą arytmetyki wskaźników. Operator postinkrementacji nie zwiększa wartości elementu wskazywanego przez wskaźnik, ale adres zawarty w tym wskaźniku. Dzięki temu wskaźnik jest „przestawiany" na kolejny element tablicy. Przed wykonaniem pętli w tablicy docelowej wstawiany jest znak końca ciągu w elemencie określonym przez wartość zmiennej `length`. Dzięki temu przekopiowany ciąg zostanie zakończony znakiem końca łańcucha, nawet jeśli oryginalny ciąg ma więcej znaków niż określa zmienna `length`.

Operacje na łańcuchach

Funkcja `strstr()`

Funkcja `strstr()` sprawdza, czy łańcuch przekazany jej jako drugi argument wywołania zawarty jest w łańcuchu przekazanym jej jako pierwszy argument wywołania. Jeśli go znajdzie, to zwraca wskaźnik na element, który zawiera pierwszy znak drugiego ciągu, jeśli nie, to zwraca wartość `NULL`. Opisana operacja nazywa się „wyszukiwaniem wzorca” (ang. *pattern matching*) w łańcuchu znaków. Jest wiele algorytmów, które ją przeprowadzają. Dla długich wzorców i przeszukiwanych ciągów efektywne będą skomplikowane algorytmy Boyera-Moore'a i KMP. My zapoznamy się z naiwnym algorytmem wyszukiwania wzorca, który według prof. S. Skieny jest efektywny dla wzorców, których długość nie przekracza pięciu znaków. Algorytm ten szuka w badanym ciągu najpierw pierwszego znaku wzorca. Jeśli go znajdzie, to sprawdza, czy za nim znajdują się pozostałe. Jeśli tak, to wzorzec został znaleziony.

Operacje na łańcuchach

Wyszukiwanie wzorca - implementacja

```
int find_match(char string[], char pattern[])
{
    int i,j,
    pattern_length=strlen(pattern),
    string_length=strlen(string);

    for(i=0;i<=(string_length-pattern_length);i++) {
        j=0;
        while((j<pattern_length)&&(string[i+j]==pattern[j]))
            j++;
        if(j==pattern_length)
            return i;
    }
    return -1;
}
```

Operacje na łańcuchach

Wyszukiwanie wzorca - komentarz

Funkcja `find_match()` działa inaczej niż `strstr()`. Zamiast wskaźników (adresów) zwraca wartość indeksu elementu, który zawiera pierwszy znak wzorca w badanym łańcuchu, lub wartość `-1`, gdy wzorzec nie występuje w przeszukiwanym ciągu. Funkcja najpierw wyznacza liczbę znaków (długości) obu ciągów. Następnie w pętli `for` odwiedza każdy z elementów przeszukiwanego ciągu (`string`) i porównuje jego wartość z wartością pierwszego elementu wzorca (`pattern`). Jeśli są one zgodne to zostanie wykonana pętla `while`. Wykonanie tej pętli będzie tak długo powtarzane, jak długo będzie zgodność między kolejnymi znakami ciągu i wzorca lub gdy skończy się wzorzec. Ten ostatni przypadek sprawdzany jest w instrukcji `if`. Jeśli jest on prawdziwy, to oznacza to, że wzorzec został znaleziony, a i-ty element ciągu zawiera jego pierwszy znak, dlatego zwracana jest wtedy wartość zmiennej `i`. W przeciwnym przypadku wzorzec nie został znaleziony i należy sprawdzić kolejne znaki ciągu.

Operacje na łańcuchach

Wyszukiwanie wzorca - komentarz

Proszę zwrócić uwagę, kiedy kończone jest przeszukiwanie łańcucha - wtedy, gdy zostanie do przeszukania mniej znaków niż zawiera wzorzec. Jeśli dotąd nie udało się go znaleźć, to znaczy, że nie występuje w ciągu i funkcja zwróci wartość -1.

Operacje na łańcuchach

Przykład użycia strtok()

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

int main(void)
{
    char string[51] = {'\0'};
    scanf("%50[^\n]s",string);
    char *result = strtok(string," ");
    while(result!=NULL) {
        printf("%s\n",result);
        result=strtok(NULL," ");
    }
    return 0;
}
```

Operacje na łańcuchach

Przykład użycia `strtok()` - komentarz

Zaprezentowany na poprzednim slajdzie program pokazuje w jaki sposób można użyć funkcji `strtok()`, aby podzielić ciąg znaków na mniejsze części, rozdzielone spacjami. W przypadku programu każdy z tych mniejszych ciągów wypisywany jest na ekranie. Najpierw funkcja jest wywoływana poza pętlą. Przekazywany jest do niej jako argument wywołania ciąg, który należy podzielić i ciąg zawierający spację, stanowiącą separator. Jeśli ta funkcja zwróci wartość różną od `NULL`[1], to zwrócony przez funkcję wskaźnik jest używany przez funkcję `printf()`, aby wypisać na ekranie wyznaczony fragment ciągu i ponownie wywoływana jest funkcja `strtok()`, aby znaleźć kolejny fragment. Tym razem jednak, jako pierwszy argument wywołania przekazywana jest jej stała `NULL`. Te dwie operacje wykonywane są w pętli `while`, tak długo, aż `strtok()` zwróci wartość `NULL`, co będzie oznaczało, że skończyły się już znaki w dzielonym łańcuchu.

[1] Zamiast tej stałej można użyć po prostu zera.

Operacje na łańcuchach

Dwie kolejne zaprezentowane funkcje, nie mają swoich odpowiedników w standardowej bibliotece języka C. Istnieją podobne podprogramy w języku Pascal. Wykonują one na tyle użyteczne działania, że warto stworzyć ich odpowiedniki w języku C.

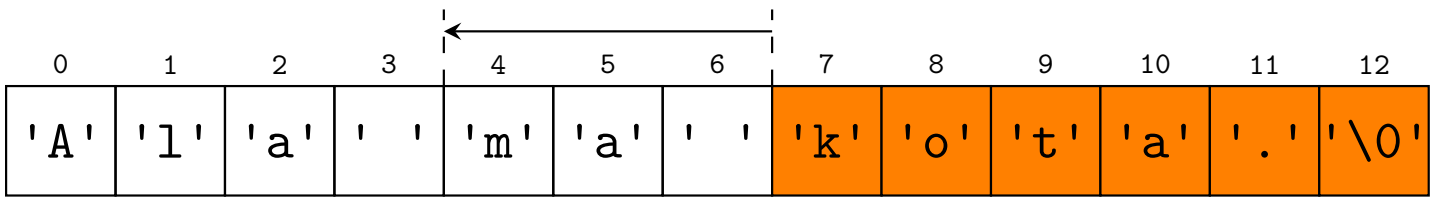
Operacje na łańcuchach

Usunięcia fragmentu łańcucha

Pierwsza funkcja usuwa część łańcucha zawierającą określoną liczbę znaków, począwszy od wskazanego miejsca. Okazuje się, że aby usunąć fragment ciągu należy wszystkie znaki znajdujące się za nim, łącznie ze znakiem końca ciągu, skopiować w lewo o tyle elementów w tablicy, ile ten fragment zawiera znaków. Aby lepiej zrozumieć ten opis zapoznamy się z krótką symulacją, w której ze zdania „Ala ma kota.” będzie usuwany wyraz „ma” wraz z występującą po nim spacją.

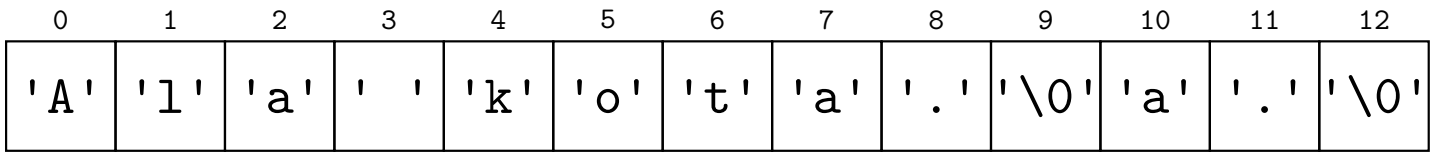
Operacje na łańcuchach

Symulacja - usunięcie fragmentu łańcucha



Operacje na łańcuchach

Symulacja - usunięcie fragmentu łańcucha



Operacje na łańcuchach

Przykład - usunięcie fragmentu łańcucha

```
int delete_from_string(char *string, unsigned int where, unsigned int how_many)
{
    if(where>=strlen(string))
        return -1;
    if(how_many>strlen(string)-where)
        return -2;

    int i;
    for(i=where;i<strlen(string);i++)
        string[i] = string[how_many+i];

    return 0;
}
```

Operacje na łańcuchach

Usunięcie fragmentu łańcucha - komentarz

Przez pierwszy parametr przekazywany jest do funkcji łańcuch, którego fragment należy usunąć. Parametr `where` określa indeks elementu, od którego należy zacząć usuwanie, a parametr `how_many` ile znaków należy usunąć. Funkcja `delete_from_string()` sprawdza, czy operacja możliwa jest do realizacji. Jeśli parametr `where` miałby wartość przekraczającą liczbę znaków w ciągu, to funkcja zwróci wartość `-1` i zakończy swoje działanie. Podobnie, jeśli wartość parametru `how_many` określałaby, że trzeba usunąć więcej znaków niż jest możliwe, to funkcja zakończy swe działanie zwracając wartość -2. Jeśli wartości parametrów są poprawne, to w pętli `for` dokonywane jest kopiowanie znaków. Na zakończenie funkcja zwraca wartość zero sygnalizującą pomyślne zakończenie operacji.

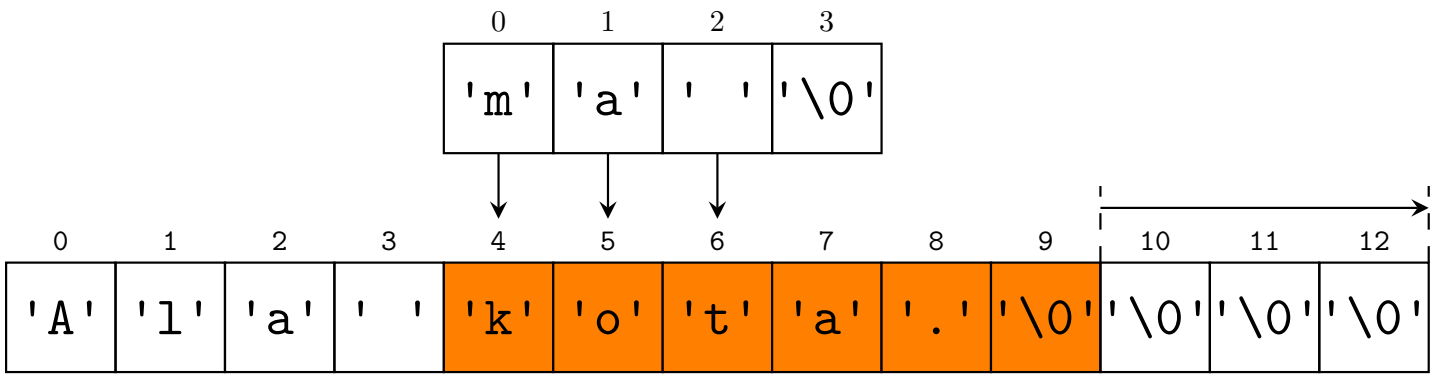
Operacje na łańcuchach

Wstawianie do łańcucha

Kolejna funkcja wstawia łańcuch (wzorzec) w określone miejsce innego łańcucha. Aby wykonać tę operację, należy najpierw znaki z łańcucha docelowego, począwszy od ustalonego elementu, skopiować w prawo o tyle elementów, ile jest znaków we wstawianym wzorcu. Następnie należy skopiować ze wzorca wszystkie znaki w przygotowane w ten sposób miejsce. Operacja ta jest zilustrowana za pomocą symulacji, w której do ciągu „Ala kota.” wstawiany jest ciąg „ma” (ze spacją), począwszy od piątego elementu.

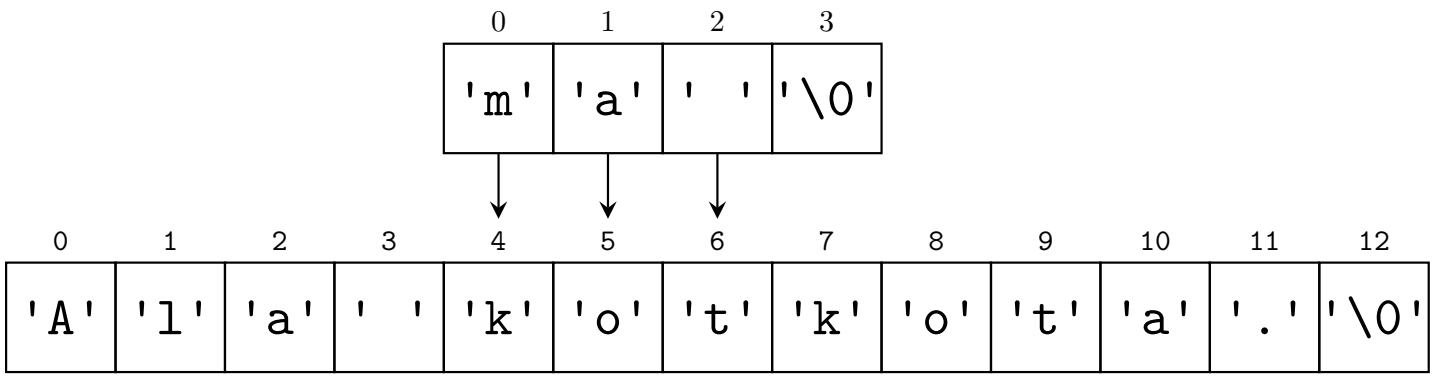
Operacje na łańcuchach

Symulacja - wstawienie do łańcucha



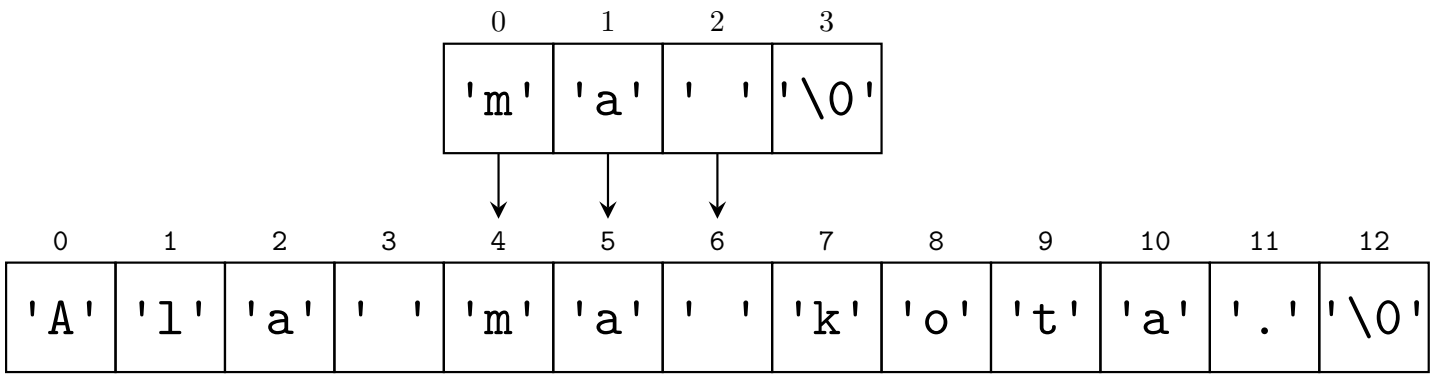
Operacje na łańcuchach

Symulacja - wstawienie do łańcucha



Operacje na łańcuchach

Symulacja - wstawienie do łańcucha



Operacje na łańcuchach

Przykład - wstawienie do łańcucha

```
int insert_into_string(char *string, const char *pattern, unsigned int where)
{
    if(strlen(pattern)+strlen(string)+1>NUMBER_OF_ELEMENTS)
        return -1;
    if(where>strlen(string))
        return -2;
    int i;
    for(i=strlen(string);i>=where;i--)
        string[i+strlen(pattern)]=string[i];
    for(i=0;i<strlen(pattern);i++)
        string[where+i]=pattern[i];
    return 0;
}
```

Operacje na łańcuchach

Wstawianie łańcucha - komentarz

Funkcja `insert_into_string()` wstawia przekazany przez parametr `patter`
ciąg do ciągu przekazanego przez parametr `string`, począwszy od elementu
o indeksie `where`. Najpierw sprawdza ona, czy łańcuch powstały w wyniku
takiego działania nie miałby więcej elementów, niż może pomieścić tablica
`string`. Jeśli tak byłoby, to kończy swoje działanie i zwraca wartość `-1`.
Jeśli nie, to sprawdza, czy parametr `where` ma poprawną wartość, tzn. czy
wskazuje miejsce wewnątrz łańcucha, a nie poza nim. Jeśli ten test dałby
niepomyślny rezultat, to funkcja zwróci wartość `-2`. Jeśli jednak oba testy
dadzą pomyślny wynik, to w pierwszej pętli `for` następuje kopiowanie zna-
ków w prawo o liczbę elementów określoną długością wzorca. W drugiej pętli
znaki ze wzorca kopiowane są w odpowiednie miejsce w łańcuchu docelo-
wym. W przypadku obu pętli proszę zwrócić uwagę na konstrukcję wyrażeń
indeksujących. Po zakończeniu ostatniej pętli funkcja zwraca wartość `0` sy-
gnalizującą pomyślne wykonanie operacji.

Konwersje

W standardowej bibliotece języka C dostępne są także funkcje, które przekształcają łańcuch przekazany im jako argument wywołania na liczbę określonego typu. Można ich użyć w programie po załączeniu pliku nagłówkowego `stdlib.h`. Tabela na następnym slajdzie zawiera zestawienie niektórych z nich.

Konwersje

Sposób użycia	**Opis**
`int number = atoi("45");`	Funkcja przekształca ciąg na liczbę typu `int`. Przekształcenie następuje nawet wtedy, gdy ciąg *zaczyna się* liczbą, a pozostałe znaki ciągu nie są cyframi. Jeśli ciąg nie daje się przekształcić na liczbę, to funkcja zwraca wartość 0.
`long int number = atol("45");`	Jak wyżej, ale funkcja zwraca liczbę typu `long int`.
`long long int number = atoll("45");`	Jak wyżej, ale funkcja zwraca liczbę typu `long long int`.
`double number = atof("45.5");`	Jak wyżej, ale zwracana liczba jest typu `double`. Część ułamkowa w łańcuchu zapisywana jest po kropce, nie po przecinku. Wartość liczby w łańcuchu może być również zapisana z użyciem notacji wykładniczej.

Konwersje

Konwersje odwrotne, tj. przekształcenie liczby na łańcuch są dokonywane za pomocą funkcji `sprintf()` i `snprintf()`, które są dostępne po włączeniu do programu pliku nagłówkowego `stdio.h`. Funkcje te działają tak, jak `printf()`, ale wynik swojego działania nie zapisują na ekranie, a do tablicy znaków, która jest podawana jako pierwszy argument ich wywołania. Funkcja `snprintf()` dodatkowo pobiera przez drugi parametr maksymalną liczbę znaków, jakie można zapisać do tej tablicy. Jest więc bezpieczniejszą wersją `sprintf()`.

Bezpieczeństwo

Funkcja `snprintf()` może być zastosowana do kopiowania, a także łączenia łańcuchów znaków, oferując przy tym większe bezpieczeństwo niż `strncpy()` lub `strncat()`. Oprócz tego, że pozwala ona określić limit kopiowanych znaków, zwraca także liczbę znaków równą lub większą od tego limitu, jeśli zostanie on przekroczony. To pozwala wykryć przypadki, kiedy wynikowy ciąg znaków został „ucięty". W niektórych przypadkach może to mieć znaczenie. Działanie `snprintf()` jest bezpieczne, jeśli przekazany jej ciąg formatujący jest stały i został określony przez programistę.
Do przetwarzania łańcuchów pochodzących bezpośrednio od użytkownika nie należy stosować `strcpy()`, `strcat()` i innych opisanych wcześniej funkcji.

Podziękowania

Składam podziękowania dla dra inż. Grzegorza Łukawskiego i dra inż. Leszka Ciopińskiego za udostępnienie materiałów, których fragmenty zostały wykorzystane w tym wykładzie.

Pytania

?

KONIEC

Dziękuję Państwu za uwagę.