

Wykład 9

- **Funkcje o nieustalonej liczbie parametrów**
- **Przykłady różnych funkcji**

```

#include <stdio.h>
#include <stdarg.h>
/* oblicza sume elemetow ciagu zakonczonego watroscia 0 */
void sum(char *msg, ...)
{
    int total = 0;
    va_list ap;
    int arg;
    va_start(ap, msg);
    while ((arg = va_arg(ap,int)) != 0) {
        total += arg;
    }
    printf(msg, total);
    va_end(ap);
}
int main(void) {
    sum(„Suma elementow listy = %d\n”, 5,2,17,4,35,11,0);
    system(„PAUSE”);
    return 0;
}

```

```
// Funkcje o nieustalonej liczbie parametrów
```

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

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

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

```
double wielomian(double x, int st, ...);
```

```
int main(void)
```

```
{
```

```
    printf("\n Wartosc wielomianu = %lf\n",
```

```
           wielomian(1.0, 3, 1.0, 2.0, 3.0, 4.0));
```

```
    return 0;
```

```
}
```

```
double wielomian(double x, int st, ...)
```

```
{
```

```
    double wa=0, wsp;
```

```
    va_list a;
```

```
    va_start(a, st);
```

```
    for (;st;st--) wa+=va_arg(a, double) * pow(x, st);
```

```
    wa+=va_arg(a, double);
```

```
    va_end(a);
```

```
    return wa;
```

```
}
```

```
Wartosc wielomianu = 10.000000
```

Deklaracja funkcji o nieustalonej liczbie parametrów ma postać następującą:

<typ-\wartości> nazwa-funkcji (parametry-ustalone, ...);

Informację o zmiennej liczbie parametrów stanowi znak przestankowy trzy kropki (symbol niedokończenia). Może on wystąpić jedynie na końcu listy parametrów funkcji.

Przykładowa funkcja *wielomian* pobiera parametr *x* określający punkt, w którym należy obliczyć wartość wielomianu oraz parametr *st* - stopień wielomianu.

Są to *parametry ustalone*. Liczba współczynników zależy od stopnia wielomianu, a zatem liczba parametrów jakie należy przekazać do funkcji jest też różna. Z tego powodu, zamiast deklaracji kolejnych parametrów umieszczono informację, że liczba pozostałych parametrów jest nieokreślona. Parametry te nazywa się *nieustalonymi*.

Dzięki parametrom nieustalonym do funkcji można przekazać jeden współczynnik w przypadku wielomianu stałego, dwa w przypadku stopnia I itd.

Dostęp do parametrów nieustalonych umożliwiają makrodefinicje ***va_start***, ***va_arg*** oraz ***va_end*** oraz typ ***va_list***, zdefiniowane w zbiorze nagłówkowym *stdarg.h*.

Makrodefinicje: ***va_list*** – tablica tego typu zawiera informacje wykorzystywane przez makra *va_arg* i *va_end*. W funkcji, która przyjmuje listę argumentów o nieustalonej liczbie należy zadeklarować zmienną tego typu;

va_start – procedura zaimplementowana jako makro ustawia *ap* tak aby wskazywała pierwszy argument przekazywany do funkcji z listy nieustalonych parametrów;

va_arg – procedura (makro) zwraca wartość wyrażenia podanego typu z następnego przekazywanego do funkcji argumentu

The header file `stdarg.h` declares one type (`va_list`) and three macros (`va_start`, `va_arg`, and `va_end`).

`va_list`: This array holds information needed by `va_arg` and `va_end`. When a called function takes a variable argument list, it declares a variable `ap` of type `va_list`.

`va_start`: This routine sets `ap` to point to the first of the variable arguments being passed to the function. `va_start` must be used before the first call to `va_arg` or `va_end`.

`va_start` takes two parameters: `ap` and `lastfix`. (the name of the last fixed parameter being passed to the called function.)

`va_arg`: This routine expands to an expression that has the same type and value as the next argument being passed (one of the variable arguments). The variable `ap` to `va_arg` should be the same `ap` that `va_start` initialized.

You cannot use `char`, `unsigned char`, or `float` types with `va_arg`.

The first time `va_arg` is used, it returns the first argument in the list. Each successive time `va_arg` is used, it returns the next argument in the list. It does this by first dereferencing `ap`, and then incrementing `ap` to point to the following item. `va_arg` uses the type to both perform the dereference and to locate the following item. Each successive time `va_arg` is invoked, it modifies `ap` to point to the next argument in the list.

`va_end`: This macro helps the called function perform a normal return.

`va_end` might modify `ap` in such a way that it cannot be used unless `va_start` is recalled. `va_end` should be called after `va_arg` has read all the arguments; failure to do so might cause strange, undefined behavior in your program.

`va_start` and `va_end` return no values; `va_arg` returns the current argument in the list (the one that `ap` is pointing to).

Schemat wykorzystania parametrów nieustalonych jest następujący:

- zadeklarować zmienną, np. *a* typu *va_list*
- przed rozpoczęciem dostępu do parametrów nieustalonych wywołać makro *va_start* z parametrami: *ap* oraz identyfikatorem ostatniego parametru ustalonego funkcji. W przykładzie funkcji *wielomian* wywołanie takie ma postać:
va_start(a, st);
- aby uzyskać dostęp do parametrów nieustalonych należy wywołać makrodefinicję *va_arg* z parametrami, przykładowo *a* i nazwą typu pobieranego parametru, np. *va_arg(a, double)*. Spowoduje to pobranie nieustalonego argumentu i potraktowanie go jako liczby rzeczywistej. Kolejne wywołania makrodefinicji *va_arg* będą powodować pobieranie kolejnych parametrów funkcji.

Przed zakończeniem operacji na parametrach nieustalonych należy wywołać makrodyrektywę `va_end(a)`. Pozwoli to normalnie zakończyć wykonywanie funkcji o zmiennej liczbie parametrów. Makrodyrektywę `va_end` należy wywoływać dopiero po przeczytaniu wszystkich argumentów nieustalonych za pomocą `va_arg`. Niespełnienie tego warunku może spowodować nieokreślone zachowanie programu

Dalej przedstawiono kilka przykładów programów, w których zastosowano podział zadań na mniejsze problemy i ich opis w postaci funkcji.

Pierwszy przykład to program, który oblicza wartość funkcji

$$f(x) = e^x$$

zadaną dokładnością ϵ z szeregu:

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots + \frac{x^k}{k!} + \dots$$

Szereg powyższy jest identyczny z:

$$e^x = \frac{x^0}{0!} + \frac{x^1}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots + \frac{x^k}{k!} + \dots$$

```
// Obliczanie exp(x) z szeregu
#include <stdio.h>
// Obliczanie wartości silnia w mianownikach
double silnia(double q)
{
    double s = 1.0;
    if (q > 0.0)
        for (double i = 1.0; i <= q; i++)
            s *= i;
    return s;
}
// Obliczanie pojedynczego wyrazu szeregu
double wyraz(double x, double c)
{
    if (c == 0.0)
        return 1.0;
    return pow(x, c)/silnia(c);
}
```

```

int main(void)
{
    double x, y = 0.0, epsilon, c = 0.0, w;
    int n = 0;
    printf("\nPodaj wartosc x: ");
    scanf("%lf", &x);
    printf("Podaj dokladnosc: ");
    scanf("%lf", &epsilon);
    do
    {
        w = wyraz(x, c);
        y += w;
        c++;
        n++;
    } while (fabs(w)/fabs(y) > epsilon);
    // Warunek zakonczenia obliczen - patrz uwaga dalej
    printf("Wynik: %20.8lf, po %d krokach\n", y, n);
    return 0;
}

```

Obliczenie przybliżonej wartości e^x wymaga uwzględnienia tylko n pierwszych wyrazów szeregu. Kiedy zatem zakończyć obliczenia aby błąd względny popełniony po odrzuceniu "ogona" szeregu – wyrazów od $n + 1$ – ego był nie większy od zadanej dokładności ?

Założono w tym przykładzie warunek zakończenia obliczeń taki, aby błąd względny spowodowany odrzuceniem sumy wyrazów $n+1$, $n+2$, itd. był mniejszy niż suma pierwszych n wyrazów szeregu co po pominięciu wyrazów $n+2$, $n+3$, itd. daje nierówność:

$$\left| \frac{x^{n+1}}{(n+1)!} \right| / \left| \sum_{i=0}^n \frac{x^i}{i!} \right| < \varepsilon$$

```

// Operacje na lancuchach (dlugosc i kopiowanie) a wskazniki
#include <stdio.h>
#include <stdlib.h>
#include <iostream.h>
void StrCpy(const char *, char *);
int StrLen(const char *);
// -----
int main(void)
{
    char strA[81], strB[81]="0123456789";
    cout << endl << "Napisz lancuch znakow: ";
    cin >> strA;
    cout << "Lancuch B przed kopiowaniem:" << endl;
    cout << strB << endl;
    cout << "Dlugosc lancucha A = " << StrLen(strA) << endl;
    StrCpy(strA, strB);    // kopiuj lancuch
    cout << "Lancuch B po kopiowaniu:" << endl;
    cout << strB << endl;
    system("PAUSE");    // Zamiast getch() w systemie Borland
    return 0;
}

```

```

// Funkcja StrCpy - kopiuje lancuch pierwszy do drugiego
void StrCpy(const char *s1, char *s2)
{
    while (*s2++=*s1++)
        ; // Puste ciało pętli
}
// Funkcja StrLen - wyznacza dlugosc lancucha
int StrLen(const char *s)
{
    int l = 0;
    for (; *s++; l++)
        ; // Puste ciało pętli
    return l; // Zwraca długość łańcucha
}

```

Napisz lancuch znakow: przemarzanie

Lancuch B przed kopiowaniem:

0123456789

Dlugosc lancucha A = 12

Lancuch B po kopiowaniu:

przemarzanie

Naciśnij dowolny klawisz, aby kontynuować . . .

W deklaracjach i w nagłówkach definicji funkcji **StrCpy** oraz **StrLen** oznaczenie typu parametru (pierwszego w funkcji **StrCpy** i jedyne w funkcji **StrLen** poprzedza słowo kluczowe **const**. Zabezpiecza ono w tym wypadku parametry funkcji przed ewentualną zmianą ich wartości w wyniku działania funkcji.

```

// Szukaj-znaku - wyszukiwanie zadanego znaku w lancuchu
#include <stdio.h>
#include <conio.h>
#include <string.h> // tam sa protoptypy funkcji lancuchowych
#include <stdlib.h>
int main(void)
{
    char ch, linia[81], *ptr;
    puts("\nWprowadz lancuch do przeszukania: ");
    gets(linia);
    printf("Wprowadz znak jakiego nalezy poszukiwac: ");
    ch = getch();
    ptr = strchr(linia, ch); // zwraca wskaznik na znak
    printf("\nLancuch zaczyna sie od adresu: %p\n", linia);
    printf("Znak %c na pozycji %p\n", ch, ptr);
    printf("To jest pozycja %d (liczac od 0)\n", ptr-linia);
    system("PAUSE");
    return 0;
}

```

Wprowadz lancuch do przeszukania:

bitwa pod Racławicami

Wprowadz znak jakiego nalezy poszukiwac: m

Lancuch rozpoczyna sie od adresu: 0012FF28

Znak m na pozycji 0012FF3B

To jest pozycja 19 (liczac od 0)

Naciśnij dowolny klawisz, aby kontynuować . . .

```

// Sortowanie.exe - sortuje liste lancuchow umieszczonych w tablicy
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
void CharSwap(char **w1, char **w2);
void CharSort(char **wsk, int n);
int main(void)
{
    const int MAXNS = 30;    // maksymalna liczba lancuchow
    const int MAXDL = 81;   // maksymalna dlugosc lancucha
    char lanc[MAXNS][MAXDL]; // tablica na pomieszczenie lancuchow
    char *wsk[MAXNS];        // tablica wskaznikow do lancuchow
    int ile = 0;             // ile lancuchow
    printf("\n");
    while ( ile < MAXNS )    // pobieranie lancuchow
    {
        printf("\nLancuch nr %d: ", ile+1);
        gets(lanc[ile]);
        if ( strlen(lanc[ile])==0 )
            break;          // skoncz jezeli pusty lancuch
        wsk[ile++] = lanc[ile]; // wsk wskazuje na lancuch
    }
}

```

```
CharSort(wsk, ile);           // sortowanie wskaźników
printf("\nUporzadkowana lista: \n");
for (int out=0; out<ile; out++)
    printf("Lancuch %d: %s\n", out+1, wsk[out]);
system("PAUSE");
return 0;
}
```

```

// Zamienia miejscami wskazniki na lancuchy
void CharSwap(char **w1, char **w2)
{
    char *tmp = *w1;
    *w1 = *w2;
    *w2 = tmp;
}
// Sortuje lancuchy
void CharSort(char **wsk, int n)
{
    int out, in;
    for (out=0; out<n-1; out++) // dla kazdego lancucha
        for (in=out+1; in<n; in++) // sprawdzaj mniejszy
            if ( strcmpi(wsk[out],wsk[in]) > 0 ) // porownaj lancuchy
                CharSwap(&wsk[out], &wsk[in]); // jezeli mniejszy
                // zamien miejscami
}

```