Header stdarg.h

Header	Opis zawartych funkcji	Dostępne makra
stdarg.h	Makra obsługi funkcji o zmiennej liczbie parametrów różnych typów	va_list(), va_start(), va_arg(), va_end()

Przekazywanie argumentów do funkcji wiąże się z liczbą tych argumentów (oczywiście także z typem każdego z nich). Jak do tej pory - bez takich narzędzi jak **stdarg.h** - chcąc przekazać np. dwa argumenty, musimy mieć prototyp funkcji z deklaracją przyjęcia dokładnie dwóch argumentów.

Jest powszechnie potrzebne przyjmowanie do jednej i tylko jednej zdefiniowanej funkcji zmienną ilość argumentów za każdym razem, gdy się ją wywołuje. W ***Python***'ie jest to koncepcja **args*, gdzie cała lista jest przekazywana do funkcji, która gwiazdką obdziera listę, z już niepotrzebnych nawiasów. Gwiazdka w ***Python***'ie to nie pointer - ***Python*** nie operuje pointerami i dlatego używam słowa *pointer* aby nie mylić go z innymi 'wskaźnikami'.

W jeszcze 'poważniejszym' programowaniu przekazywanie do wbudowanej funkcji elementów macierzy, wiąże się z ustaleniem ilości wierszy i kolumn tej macierzy. Można to zrobić tak (język FORTRAN IV):

```
transp_matrix( [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 ], [ 3, 4 ])
```

gdzie 3 to liczba wierszy a 4 kolumn, dla macierzy

```
1 4 7 10
2 5 8 11
3 6 9 12
```

Chociaż prościej byłoby tak:

```
transp_matrix(3, 4, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12)
```

I z tej koncepcji korzysta **stdarg.h** języka C.

#include<stdarg.h>

Z definicji plika nagłówkowego **stdarg.h**:

```
typedef void *va_list;                <-- lista argumentów
void va_start(va_list ap, lastfix):   <-- va_start ( lista argumentów, Pole_kontrolne )
type va_arg(va_list ap, type);        <-- va_arg ( lista argumentów, typ np. double )
void va_end(va_list ap):              <-- va_end ( lista argumentów )
```

Te makra "listy argumentów" służą do konstruowania funkcji, która akceptuje zmienną liczbę argumentów zadeklarowanych za pomocą wielokropka (`...`). Na przykład można zadeklarować funkcję podobną do tej:

```
void DowolnaFunkcja(int Pole_kontrolne, ...);
```

Funkcja **DowolnaFunkcja()** zwraca wartość void i wymaga co najmniej jednego argumentu typu **int** (**Pole_kontrolne**, poniżej w pierwszym przykładzie to **unsigned int** mówiący ile parametrów jest do odczytu). Wielokropek wskazuje, że oprócz **Pole_kontrolne** instrukcje mają przekazywać co najmniej jedną dodatkową wartość argumentu dowolnego typu (z wyjątkiem wartości typu *char*, *unsigned char* i *float*, które są podnoszone do innych, 'wyższych' typów i dlatego nie są dozwolone na listach argumentów zmiennych; poniżej w przykładzie to lista argumentów typu *double* a nie *float*).

Wewnątrz tej funkcji potrzebny jest specjalny kod, aby uzyskać dostęp do tych dodatkowych parametrów (to znaczy tych, oprócz '**Pole_kontrolne**'). Najpierw deklaruje się zmienną typu **va_list** (wskaźnik do listy argumentów) i inicjuje ją za pomocą **va_start()**:

```
va_list vap;
va_start(vap, Pole_kontrolne);
```

Następnie używa się **va_arg()** aby wyodrębnić jeden lub więcej argumentów dowolnego typu (z wyjątkiem wykluczonych typów wymienionych powyżej). Załóżmy na przykład, że instrukcja przekazuje do funkcji dwie wartości typu **int**. Można załadować te wartości do zmiennych lokalnych w następujący sposób:

```
int v1 = va_arg(vap, int);
int v2 = va_arg(vap, int);
```

Makro **va_arg()** wymaga dwóch argumentów: wskaźnika, tutaj **vap** (sugeruje nazwę '**va p**ointer') do **va_list** i typu argumentu do pobrania (tutaj **int**). Jeśli liczba argumentów nie jest znana, używa się wskaźnika, aby zaznaczyć koniec listy. Można na przykład przekazać unikatową wartość, taką jak -1, do funkcji wieloparametrowych, aby zakończyć listę poprzednio odczytanych wartości. Można wywołać funkcję w ten sposób:

```
DowolnaFunkcja(10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0, -1);
```

Zakładając, że -1 jest wskaźnikiem końca listy, w **DowolnaFunkcja** używa się pętli, aby uzyskać dostęp do jej argumentów:

```
void DowolnaFunkcja(int Pole_kontrolne, ...)
{
   va_list vap;                    // Wskaźnik do listy argumentów
   int v;                          // Przechowuje wartość kolejnego argumentu
   printf("%d\n", Pole_kontrolne); // Wyświetla wartość 'pola kontrolnego'
   va_start(vap, Pole_kontrolne);  // Rozpoczyna uzyskiwanie dostępu do listy 'vap'
   while (( v = va_arg(vap, int)) != -1) // Pobiera jeden argument w kroku pętli
      print("%d\n", v);            // Wyświetla wartość argumentu
   va_end(vap);                    // Sygnalizuje koniec listy
}
```

Jednakże w poniższych przykładach pozycja końca listy będzie kontrolowana tak, że podobnego do wyżej opisanego 'wskaźnika końca listy' nie będziemy potrzebować.

Ostatni krok w procesie przekazuje zainicjowany wskaźnik **vap** do **va_end()**, aby zakończyć poprzednie wywołanie metody **va_start()**.

Parametry:

va_list vap - Wskaźnik do listy argumentów zmiennych. Przekazuje ją do **va_start()** w celu zainicjowania, do **va_arg()** w celu pobrania następnej wartości argumentu i do **va_end()** w celu zasygnalizowania zakończenia procesu pobierania argumentów.

lastfix - **Pole_kontrolne**. W pierwszym z poniższych przykładów jest to numer ostatniego (czyli skrajnie po prawej stronie) wpisanego argumentu na liście argumentów. W przypadku argumentów różnych typów, **Pole_kontrolne** jako ciąg charakterystycznych znaków, kontroluje ich odczyt. To argument przekazywany tylko do **va_start()**.

type - Typ danych, który ma zostać zwrócony przez **va_arg()**. Kolejne argumenty mogą być różnych typów przy ich odczycie przez **va_arg()**. Nie mogą być nimi *char*, *unsigned char* ani *float*.

Wiemy już, że w wywołanej funkcji **va_list** wprowadza listę jej argumentów a **va_start** ją uruchamia. Są to pojedyncze instrukcje w wywołanej funkcji. **va_arg** natomiast jest uruchamiane za każdym razem, gdy

pobierany jest kolejny element listy. **va_end** to znowu pojedyncza instrukcja w wywołanej funkcji - mówi, że zakończył się proces pobierana elementów listy.

W poniższym przykładzie:

```
wynik = Srednia(5, 64.5, 79.0, 26.5, 89.0, 43.0);
```

`5` to **Pole_kontrolne/ilosc**, podające liczbę przekazywanych wartości, które w wywołanej funkcji stają się jej argumentami zajmując symbol wielokropka.

`64.5, 79.0, 26.5, 89.0, 43.0` to lista argumentów. Te liczby są tu typu **double**.

```
// Przekazywanie argumentów do funkcji zdolnej przyjmować zmienną liczbię argumentów
#include <stdio.h>
#include <stdarg.h>                    // ten temat
#include<locale.h>                     // dla 'setlocale()'

double Srednia(unsigned ilosc, ...);  // prototyp funkcji

int main()
{  setlocale(LC_CTYPE, "Polish");      // polskie znaki
   double wynik;

   wynik = Srednia(5, 64.5, 79.0, 26.5, 89.0, 43.0);
   printf("Średnia = %.2lf\n", wynik);

   wynik = Srednia(3, 12.3, 45.6, 78.9);
   printf("Średnia = %.2lf\n", wynik);

   return 0;
}

// Oblicz średnią z zestawu liczb typu 'double'
// Ustaw 'ilosc' na ilość liczb, które będą zaraz po niej podane
double Srednia(unsigned ilosc, ...)
{
   va_list vap;
   va_start(vap, ilosc);

   double suma = 0.0;
   for (int i = 0; i < ilosc; i++)
      suma += va_arg(vap, double);

   va_end(vap);
   return suma / ilosc;
}
```

Źródło powyższych informacji: Tom Swan, *Mastering Borland C++ 4.5*, Second edition, SAMS, 1995, ISBN 0-672-30546-1, strony 1339-1341.

Wynik działania programu:

```
Średnia = 60.40
Średnia = 45.60
```

```
--------------------------------
Process exited after 2.165 seconds with return value 0
Press any key to continue . . . _
```

Poniższy kod nie jest wykorzystaniem idei 'akceptacji zmiennej liczby argumentów' ale:
– prowadzi do niej w następnym kodzie, który jest rozwinięciem tego kodu,
– pozwala śledzić pracę makr, szczególnie **va_arg()**.

```c
#include <stdio.h>
#include <stdarg.h>                 // ten temat
#include <locale.h>                 // dla 'setlocale()'

void rozne_typy_argumentow(const char* typ, ...);   // prototyp funkcji

int main(void)
{   setlocale(LC_CTYPE, "Polish");  //polskie znaki

    rozne_typy_argumentow("", 'A', 1, 2.2222, -3.333);
    return 0;
}

void rozne_typy_argumentow(const char* typ, ...)
{   va_list vap;
    va_start(vap, typ);

    // znak (char, tu 'A') będzie 'podniesiony' do typu całkowitego (int)
    int a = va_arg(vap, int);
    printf("Znak                      : %c\n", a);

    int b = va_arg(vap, int);
    printf("Liczba całkowita          : %d\n", b);

    // typ zmiennoprzecinkowy (float) będzie 'podniesiony' do typu liczby podwójnej
    //   presyzji (double) - dotyczy dwóch poniższych przypadków
    double c = va_arg(vap, double);
    printf("Liczba zmiennoprzecinkowa : %.2lf\n", c);

    double d = va_arg(vap, double);
    printf("Liczba zmiennoprzecinkowa : %.2lf\n", d);

    va_end(vap);
}
```

Wynik działania programu:

```
Znak                      : A
Liczba całkowita          : 1
Liczba zmiennoprzecinkowa : 2.22
Liczba zmiennoprzecinkowa : -3.33

--------------------------------
```

```
Process exited after 2.342 seconds with return value 0
Press any key to continue . . . _
```

Poniższy kod już stosuje zalety 'akceptacji zmiennej liczby argumentów o różnych typach'.
Zapis jak `"cisf"` może sugerować: pierwszy element to *char*, drugi to *int*, trzeci to *string*, czwarty to *float*.
Po raz kolejny zaznaczam, że *char*, *unsigned char* i *float* są tu promowane do 'wyższych' typów.

```c
#include <stdio.h>
#include <stdarg.h>                 // ten temat
#include <locale.h>                 // dla 'setlocale()'

void rozne_typy_argumentow(const char* typ, ...);  // prototyp funkcji

int main(void)
{   setlocale(LC_CTYPE, "Polish");  //polskie znaki
    rozne_typy_argumentow("cicffci", 'A', 1, 'a', 2.2222, -3.333, 'b', 4);
    printf("\n");
    rozne_typy_argumentow("fcisf", -88.88, 'Z', 77, "Artur", 999.999);
    return 0;
}

void rozne_typy_argumentow(const char* typ, ...)
{   va_list vap;
    va_start(vap, typ);
    while(*typ != '\0')
    {  if (*typ == 'c')
       {  // znak (char, tu 'A') będzie 'podniesiony' do typu całkowitego (int)
          int a = va_arg(vap, int);
          printf("Znak                      : %c\n", a);
       } else
         if (*typ == 'i')
         {  int b = va_arg(vap, int);
            printf("Liczba całkowita          : %d\n", b);
         } else
           if (*typ == 'f')
           {  // typ zmiennoprzecinkowy (float) będzie 'podniesiony' do typu liczby
              //   podwójnej precyzji (double)
              double c = va_arg(vap, double);
              printf("Liczba zmiennoprzecinkowa : %.2lf\n", c);
           } else
             if (*typ == 's')
             {  char* d = va_arg(vap, char*);
                printf("Łańcuch znakowy           : %s\n", d);
             }
       ++typ;
    }

    va_end(vap);
}
```

Wynik działania programu:

```
Znak                      : A
```

```
Liczba całkowita          : 1
Znak                      : a
Liczba zmiennoprzecinkowa : 2.22
Liczba zmiennoprzecinkowa : -3.33
Znak                      : b
Liczba całkowita          : 4

Liczba zmiennoprzecinkowa : -88.88
Znak                      : Z
Liczba całkowita          : 77
Łańcuch znakowy           : Artur
Liczba zmiennoprzecinkowa : 1000.00

--------------------------------
Process exited after 3.184 seconds with return value 0
Press any key to continue . . . _
```