Argumenty linii poleceń

```
echo ahoj, przygodo

copy <plik zrodlowy> <plik docelowy>
```

Przykładowy program *echo*

I - traktujemy **argv** jako tablicę wskaźników do znaków:

```
#include <stdio.h>

/*  echo argumentow wywolania; wersja 1  */
main(int argc, char *argv[])
{
  int i;
  for (i=1; i<argc; i++)
    printf("%s%s",argv[i], (i<argc-1) ? " ":"");
  printf("\n");
  return 0;
}
```

II - traktujemy **argv** jako wskaźnik do wskaźnika do znaków:

```
#include <stdio.h>

/*  echo argumentow wywolania; wersja 2  */
main(int argc, char *argv[])
{
  while (--argc > 0)
    printf("%s%s",*++argv, (argc>1) ? " ":"");
  printf("\n");
  return 0;
}
```

Ogólna składnia funkcji main()

```
/*  instrukcje preprocesora  */
/*  instrukcje globalne      */

int main([[[ <licznik argumentow>], <lista
    argumentow>], <lista argumentow
    srodowiska>])
int <licznik argumentow>;
char * <lista argumentow>[];
char * <lista argumentow srodowiska>[];
{
   /*  instrukcje funkcji main  */
}
```

Uwaga: Zgodnie z konwencją:

- argv[0] wskazuje nazwę, z jaką program został wywołany.
- zatem argc musi być większe lub równe 1.
- ponadto standard wymaga, by *argv[argc]* był wskźnikiem pustym

Standard ANSI dopuszcza trzy opcjonalne argumenty. Zgodnie z konwencją argumenty mają następujące nazwy:

Nazwa	Znaczenie
argc	licznik argumentów typu integer
argv	lista argumentów w postaci łańcuchów
env	lista argumentów środowiska w postaci łańcuchów

Wykorzystując konwencjonalne nazwy, składnię funkcji *main()* możemy zapisać:

```
int main([[[argc],argv],env])
int argc;
char *argv[];
char *env[];
{
  /* instrukcje funkcji main()  */
}
```

Zgodnie z aktualnym standardem funkcja *main()* może wystąpić w czterech formatach:

1. Bez argumentów

```
main(void)
{
  /*  instrukcje funkcji main() */
}
```

2. z jednym argumentem (rzadko wykorzystywana)

```
 main( int argc )
 {
   /*  instrukcje funkcji main()  */
 }
```

3. z dwoma argumentami (najczęściej używana)

```
main(int argc, char *argv[])
{
  /* instrukcje funkcji main()  */
}
```

4. z trzema argumentami:

```
main(int argc, char *argv[], char *env[])
{
  /*  instrukcje funkcji main()  */
}
```

Przykład I:

```
/* argc.c */
#include <stdio.h>
main(argc)
int argc;
{
 printf("Liczba argumentow linii "
        "polecen wynosi:   %d\n",argc);
}
```

Przykład II:

```
/* argv.c */
#include <stdio.h>
main(argc,argv)
int argc;
char *argv[];
{
 int count = 0;
 void newline (void);
 printf("Lista argumentow linii polecen : \n\n");
 for (count = 0; count < argc ; count++ )
     printf("%s \n",argv[count]);
 newline();
}

void newline(void)
{
 printf("\n");
}
```

Przykład III:

```
/* env.c */
#include <stdio.h>
main(argc,argv,env)
int argc;
char *argv[];
char *env[];
{
 int count = 0;
 void newline (void);
 printf("Lista argumentow srodowiska "
        "linii polecen :\n\n");
 for (count = 0; env[count] != 0; count++ )
    printf("%s\n",env[count]);
 newline();
}

void newline(void)
{
 printf("\n");
}
```

Przykład IV (wyszukiwanie wzorca):

```
#include <stdio.h>
#include <string.h>
#define MAXLINE 1000

int getline(char s[], int lim);
main(int argc, char *argv[])
{
   char line[MAXLINE];
   int found=0;
   if (argc != 2)
     printf("Format wywolania:  find wzorzec\n");
   else
     while ( getline(line, MAXLINE) > 0)
       if (strstr(line, argv[1]) != NULL){
         printf("%s",line);
         found++;
       }
   return found;
}

/* getline: wczytaj wiersz do tablicy s;
   podaj jego dlugosc  */
int getline(char s[], int lim)
{
   int c,i=0;

   while (--lim>0 && (c=getchar())!=EOF && c!='\n')
     s[i++]=c;
   if (c=='\n')
     s[i++]=c;
   s[i]='\0';
   return i;
}
```

Przykład V (wyszukiwanie wzorca z opcjami **-x** (wypisywanie linii nie zawierających wzorca) **-n** (podawanie numeru wypisywanej lini):

```
#include <stdio.h>
#include <string.h>
#define MAXLINE 1000

int getline(char s[], int lim);

/* find: wypisz wiersze pasujace do wzorca z
pierwszego, obowiazkowego argumentu */
main(int argc, char *argv[])
{
   char line[MAXLINE];
   long lineno=0;
   int c, except=0, number=0, found=0;

   while (--argc > 0 && (*++argv)[0] == '-')
     while (c=*++argv[0])
       switch (c) {
       case 'x':
         except=1;
         break;
       case 'n':
         number=1;
         break;
       default:
         printf("find: nieznana opcja %c\n",c);
         argc=0;
         found=-1;
         break;
       }
   if (argc != 1)
     printf("Format wywolania: find -x -n wzorzec\n");
   else
     while (getline(line,MAXLINE) > 0) {
       lineno++;
       if ((strstr(line, *argv) != NULL) != except) {
         if (number)
           printf("%ld:",lineno);
         printf("%s",line);
         found++;
       }
     }
   return found;
}

/* getline: wczytaj wiersz do tablicy s;
   podaj jego dlugosc  */
int getline(char s[], int lim)
{
   int c,i=0;

   while (--lim>0 && (c=getchar())!=EOF && c!='\n')
     s[i++]=c;
   if (c=='\n')
     s[i++]=c;
   s[i]='\0';
   return i;
}
```

Przykład VI (przekazanie nazwy pliku do otwarcia jako argumentu programu):

```
#include <stdio.h>
int main(argc, argv)
int argc;
char *argv[];
{
 FILE *fileptr;
 if (( fileptr = fopen (argv[1], "rb" )) == NULL )
   {
     printf( "Zbior %s nie istnieje\n", argv[1] );
     return(1);
   }
 else
   printf( " Zbior %s otwarty poprawnie\n", argv[1] );
fclose(fileptr);
return(0);
}
```

Parametryzacja programu

Trzy różne mechanizmy parametryzacji:

- Poprzez dyrektywy PREPROCESORA
- argumenty funkcji main()
- deklarację *typedef*

Deklaracja *typedef*

Służy do tworzenia nowych nazw typów danych.
Przykład:

```
typedef int Length;  /*  Dlugosc  */
```

Dalej w deklaracjach i rzutowaniach można korzystać z typu *Length* tak samo, jak z typu *int*

```
Length len, maxlen;
Length *lengths[];
```

Przykład II (deklaracja):

```
typedef char *String; /* Tekst  */
```

użycie:

```
String p, lineptr[MAXLINES], alloc(int);
int strcmp( String, String );
p=(String) malloc(200);
```

Przykład III:

```
typedef struct tnode { /* wezel drzewa */
  char *word;    /* wskaznik do tekstu slowa */
  int count;     /* licznik wystapien  */
  struct tnode *left;  /* lewy potomek  */
  struct tnode *right; /* prawy potomek */
} Treenode, *Treeptr;
/* Treenode - wezel,
   Treeptr - wskaznik do wezla */
```

Obie deklaracje tworzą dwa nowe słowa kluczowe dla określenia typów: *Treenode* (struktura) i *Treeptr* (wskaźnik do takiej struktury).

Funkcję przydziału pamięci dla nowego węzła można napisać np. tak:

```
Treeptr talloc(void)
{
  return (Treeptr) malloc(sizeof(Treenode)}
}
```

Uwaga: *Typedef* nie tworzy nowego typu; daje tylko nową nazwę dla typu już istniejącego:

Jest więc podobna do dyrektywy preprocesora
`#define`
jest jednak interpretowana przez kompilator.

Inne przykłady:

```
1)  typedef int (*PFI)(); /* tworzy typ PFI
     jako wskaznik do funkcji zwracajacej
     wartosc calkowita */

2)  typedef char Arr[3];
    Arr Vec, *Ptr;
    /* zadeklarowano typ Arr utozsamiajac go
       z typem  "char  [3]"
       a nastepnie tablice  Ver  typu "char  [3]"
       i zmienna  Ptr typu "char (*) [3]" */

3)  typedef int *REF;
    REF Ptr, Arr[2];

4)  typedef struct CPLX {
      double Re, Im;
    } COMPLEX;
    COMPLEX Vec[3]; /* tablica typu complex */

5)  typedef float *Vec[3], (*REF) (long,int);
    Vec Arr[2];
    REF *Ptr; /* typu "float (**) (long,int)" */
```

Dwa główne powody przemawiające za używaniem deklaracji *typedef*:

1. parametryzacja programu w związku z przenoszeniem na inne maszyny

2. lepsze komentowanie programu