Wskaźniki do funkcji

```
/* Sortowanie numerycznie i leksykograficznie */
#include <stdio.h>
#include <string.h>

#define MAXLINES 5000 /* maksymalna liczba wierszy
                         do sortowania */
char *lineptr[MAXLINES];  /* wskazniki do wierszy
                             tekstu */

int readlines(char *lineptr[], int nlines);
void writelines(char *lineptr[], int nlines);

void qqsort(void *lineptr[], int left, int right,
            int (*comp)(void *, void *));
int numcmp(const char *, const char *);

/* uporzadkuj wiersze z wejscia */
main(int argc, char *argv[])
{
  int nlines; /* liczba wszystkich wierszy */
  int numeric=0; /* 1 - jesli sortowanie numeryczne */

  if (argc > 1 && strcmp(argv[1], "-n") == 0)
    numeric=1;
  if ((nlines = readlines(lineptr, MAXLINES)) >= 0) {
    qqsort((void **) lineptr, 0, nlines-1,
          (int (*) (void *, void *))
          (numeric ? numcmp : strcmp));
    writelines(lineptr, nlines);
    return 0;
  } else {
    printf("za duzo wierszy do sortowania\n");
    return 1;
  }
}

/* qsort: uporzadkuj v[left] ... v[right] rosnaco */
void qqsort(void *v[], int left, int right,
           int (*comp)(void *, void *))
{
  int i,last;
  void swap(void *v[], int, int);
  if (left >= right) /* nic nie rob, jesli tablica*/
    return;     /* zawiera mniej niz dwa elementy */
  swap(v, left, (left+right)/2);
  last=left;
  for (i=left+1; i<=right; i++)
    if ((*comp)(v[i],v[left]) < 0)
      swap(v, ++last, i);
  swap(v, left, last);
  qqsort(v, left, last-1, comp);
  qqsort(v, last+1, right, comp);
}

#include <stdlib.h>

/* numcmp: porownaj numerycznie s1 i s2 */
int numcmp(const char *s1, const char *s2)
{
  double v1,v2;

  v1=atof(s1);
  v2=atof(s2);
  if (v1 < v2)
    return -1;
  else if (v1 > v2)
    return 1;
  else
    return 0;
}



void swap(void *v[], int i, int j)
{
  void *temp;

  temp=v[i];
  v[i]=v[j];
  v[j]=temp;
}

#define MAXLEN 1000 /* dlugosc wiersza wejsciowego */
int getline(char *, int);

/* readlines: wczytaj wiersze z wejscia */
int readlines(char *lineptr[], int maxlines)
{
  int len, nlines=0;
  char *p, line[MAXLEN];

  while ((len=getline(line, MAXLEN)) > 0)
    if (nlines >= maxlines || (p=(char *)
                  malloc(len))==NULL)
      return -1;
    else {
      line[len-1]='\0'; /* usun znak nowego wiersza */
      strcpy(p,line);
      lineptr[nlines++]=p;
    }
  return nlines;
}

/* writelines: wypisz wiersze na wyjscie */
void writelines(char *lineptr[], int nlines)
{
  int i;
  for (i=0; i < nlines; i++)
    printf("%s\n", lineptr[i]);
}

/* getline: wczytaj wiersz do s, podaj jego dlugosc */
int getline(char s[], int lim)
{
  int c,i;
  for (i=0; i<lim -1 && (c=getchar()) != EOF
                     && c != '\n'; ++i)
    s[i]=c;
  if (c=='\n')
    s[i]=c;
  else
    ++i;
  s[i]='\0';
  return i;
}
```

Argumenty wskaźnikowe typu *void **

Funkcja **qqsort** ma argumenty typu *void ** dla argumentów wskaźnikowych. Za pomocą operacji rzutowania można dowolny wskaźnik przekształcić do typu *void ** i z powrotem bez utraty informacji.

Deklaracja

```
int (*comp) (void *, void *)
```

deklaruje wskaźnik do funkcji zwracającej wartość całkowitą. Natomiast

```
int *comp(void *, void *)
```

mówi, że *comp* jest funkcją zwracającą wskaźnik do obiektów całkowitych.

Skomplikowane deklaracje

Przykłady:

char **argv	argv: wskaźnik do wskaźnik do char
int (*daytab)[13]	daytab: wskaźnik do tablica[13] o elementach int
int *daytab[13]	daytab: tablica[13] o elementach wskaźnik do int
void *comp()	comp: funkcja zwracająca wskaźnik do void
void (*comp)()	comp: wskaźnik do funkcja zwracająca void
char (*(*x())[])()	x: funkcja zwracająca wskaźnik do tablica[] o elementach wskaźnik do funkcja zwracająca char
char (*(*x[3])())[5]	x: tablica[3] o elementach wskaźnik do funkcja zwracająca wskaźnik do tablica[5] o elementach char

Uproszczona postać reguł skaładniowych gramatyki wprowadzającej deklaratory:

```
deklarator:
        opcjonalne * bezposredni-deklarator

bezposredni-deklarator:
        nazwa
        (deklarator)
        bezposredni-deklarator()
        bezposredni-deklarator[opcjonalny rozmiar]
```

Przykład (program rozszyfrowujący proste deklaracje):

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#define MAXTOKEN 100

enum ( NAME, PARENS, BRACKETS };

void dcl(void);
void dirdcl(void);
int gettoken(void);
int tokentype;        /* typ ostatniego elementu  */
char token[MAXTOKEN]; /* tekst ostatniego elementu */
char name[MAXTOKEN];  /* nazwa wystepujaca w dekl. */
char datatype[MAXTOKEN]; /*typ danych: char,int,itp.*/
char out[1000];       /* wyjsciowy opis slowny */

main()  /* dcl: zamien deklaracje C na opis slowny */
{
  while (gettoken() != EOF) {/* pierwszy leksem w */
    strcpy(datatype, token)  /* wierszu jest typem*/
    out[0]='\0';             /* danych */
    dcl();    /* analiza skladniowa reszty wiersza */
    if (tokentype != '\n')
      printf("blad skladniowy\n");
    printf("%s: %s %s\n", name, out, datatype);
  }
  return 0;
}

int gettoken(void) /* podaj nastepny leksem */
{
  int c, getch(void);
  void ungetch(int );

  while ((c=getch()) == ' ' || c == '\t')
    ;
  if (c == '(') {
    if ((c=getch()) == ')') {
      strcpy(token, "()");
      return tokentype = PARENS;
    } else {
      ungetch(c);
      return tokentype = '(';
    }
  } else if (c == '[') {
      for (*p++ = c; (*p++ = getch()) != ')';)
        ;
      *p='\0';
      return tokentype = BRACKETS;
  } else if (isalpha(c)) {
      for (*p++ = c; isalnum(c=getch()); )
        *p++=c;
      *p='\0';
      ungetch(c);
      return tokentype = NAME;
  } else
      return tokentype = c;
}

#define BUFSIZE 100  /* maks. rozmiar bufora */

char buf[BUFSIZE];  /* bufor na zwroty z ungetch */
int bufp=0;  /* nastepne wolne miejsce w buforze */

int getch(void) /* wez znak, byc moze oddany na */
{               /*  wejscie */
  return (bufp > 0) ? buf[--bufp]:getchar();
}

void ungetch(int c) /* oddaj znak z powrotem na */
{                   /*  wejscie */
  if (bufp >= BUFSIZE)
    printf("ungetch: za wiele zwrotow\n");
  else
    buf[bufp++]=c;
}


/* dcl: analiza skladniowa deklaratora */
void dcl(void)
{
  int ns;
  for (ns=0; gettoken()=='*';) /* zlicza *-i */
    ns++;
  dirdcl();
  while (ns-- > 0)
    strcat(out, "wskaznik do");
}

/* dirdcl: analiza skladniowa bezposredniego
                                deklaratora */
void dirdcl(void)
{
  int type;
  if (tokentype == '(') {  /* (deklarator) */
    dcl();
    if (tokentype != ')')
      printf("blad: brak nawiasu )\n");
  } else if (tokentype == NAME) /* nazwa zmiennej */
          strcpy(name, token);
  else
    printf("blad: spodziewana nazwa lub"
                                " (deklarator)\n");
  while (type=gettoken())==PARENS || type==BRACKETS)
    if (type == PARENS)   /* para nawiasow () */
      strcat(out,"funkcja zwracajaca");
    else {                /* para nawiasow [] */
      strcat(out, " tablica");
      strcat(out,token);  /* ew. rozmiar */
      strcat(out," o elementach");
    }
}
```

Zmienna długość list argumentów

```
#include <stdio.h>
#include <stdarg.h>
/* minprintf: minimalna printf ze zmienna
liczba argumentow */
void minprintf(char *fmt, ... )
{
  va_list ap; /* wskazuje po kolei kazdy nienazwany*/
  char *p, *sval;           /*  argument */
```

```
  int ival;
  double dval;
  va_start(ap, fmt); /* ap wskazuje pierwszy nienaz-
   wany argument;   fmt - ostatni nazwany argument */
  for (p=fmt; *p; p++) {
    if (*p != '%') {
      putchar(*p);  continue;
    }
    switch(*++p) {
    case 'd':
      ival = va_arg(ap, int);
      printf("%d",ival);
      break;
    case 'f':
      dval=va_arg(ap, double);
      printf("%f",dval);
      break;
    case 's':
      for (sval=va_arg(ap, char *); *sval; sval++)
        putchar(*sval);
      break;
    default:
      putchar(*p);
      break;
    }
  }
  va_end(ap); /* po pracy wyczysc po sobie */
}
```

Trzy makra zdefiniowane w `<stdarg.h>`:

```
va_start(va_list ap, lastfix);
type va_arg(va_list ap, type);
void va_end(va_list ap);

va_list   -  tablica, ktora przechowuje informacje
             potrzebne dla va_arg i va_end
```

`va_arg` i `va_start` zapewniają przenaszalny sposób dostępu do zmiennych list argumentów.

- `va_start` ustawia wskaźnik do pierwszego nienazwanego argumentu przekazanego do funkcji
- `va_arg` rozszerza się do wyrażenia mającego ten sam typ i wartość jak następny przekazany argument
- `va_end` pomaga funkcji wykonać normalny powrót

Przykład (sumowanie liczb):

```
#include <stdio.h>
#include <stdarg.h>
/* calculate sum of a 0 terminated list */
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("The total of 1+2+3+4 is %d\n", 1,2,3,4,0);
   return 0;
}
```