C Programska kartica

Predprocesor

zamenjava	`#define` *niz zamenjava*
vključevanje datotek	`#include` *ime_dat*

Spremenljivke

tipi	`char`, `int`, `short`, `long`, `float`, `double`
modifikatorji	`signed`, `unsigned`, `const`, `static`
definicija novega tipa	`typedef` *tip novo_ime_tipa*`;`
deklaracija spremenljivke	*tip ime_spr*`;`
deklaracija polja	*tip ime_polja*`[`*velikost*`];`

deklaracija strukture

```
struct struktura{
   tip element1;
   tip element2;
} ime_spr;
```

inicializacija spremenljivke *tip ime = vrednost*`;`

```
int stevilo = 5;
tip polje[] = {v1, v2, ...};
char niz[] = "niz znakov";
```

Argumenti programa

```
main(int argc, char *argv[])
```

`argc` je število argumentov; `argv` je polje kazalcev na nize znakov, pri čemer je vsak niz en argument programa (prvi argument je ime programa).

Matematične funkcije `<math.h>`

Argumenti in vrnjene vrednosti vseh funkcij so tipa `double`.

trigonometrične	`sin(x)`, `cos(x)`, `tan(x)`	
inverzne trig.	`asin(x)`, `acos(x)`, `atan(x)`	
logaritmi	`log(x)`	naravni logaritem
potence	`exp(x)`	e^x
	`pow(x,y)`	x^y,
	`sqrt(x)`	kvadratni koren
zaokroževanje	`ceil(x)`	zaokroži navzgor
	`floor(x)`	zaokroži navzdol
ostalo	`fabs(x)`	absolutna vrednost

Operatorji in njihove prioritete

element polja	`[]`
klic funkcije	`()`
postfiksni inkrement, dekrement	*spr*`++`, *spr*`--`
element strukture	*ime*`.`*element*
element str. na katero kaže kazalec	*kazalec*`->`*element*
prefiksni inkrement, dekrement	`++`*spr*, `--`*spr*
unarni plus in minus	`+`, `-`
logični ne, bitni ne	`!`, `~`
vsebina kazalca	`*`*kazalec*
naslov spremenljivke	`&`*spremenljivka*
pretvorba tipa	`(`*nov_tip*`)` *spremenljivka*
velikost objekta	`sizeof(`*tip*`)`
množenje, deljenje, ostanek	`*`, `/`, `%`
seštevanje, odštevanje	`+`, `-`
bitni premik levo, desno	`<<`, `>>`
primerjave	`<`, `<=`, `>`, `>=`
primerjave	`==`, `!=`
bitni in	`&`
bitni ekskluzivni ali	`^`
bitni ali	`|`
logični in	`&&`
logični ali	`||`
pogojni operator	*izraz* `?` *izr1* `:` *izr2*
prireditveni operatorji	`=`, `+=`, `-=`, `*=`, `/=`, ...
ločilo med izrazi	`,`

unarni operatorji, pogojni operator in prireditveni operatorji se združujejo od desne proti levi;
vsi ostali operatorji se združujejo od leve proti desni

Konverzije `<stdlib.h>`

```
int atoi(char *niz)
double atof(char *niz)
```

Pretvorita začetni del podanega niza (do prvega neštevilskega znaka) v celo število oz. število s plavajočo vejico. Če podani niz ne predstavlja števila, funkciji vrneta `0`.

Operacije z znaki `<ctype.h>`

`int isalpha(int znak)`	je črka
`int isalnum(int znak)`	je črka ali številka
`int isdigit(int znak)`	je številka
`int islower(int znak)`	je mala črka
`int isupper(int znak)`	je velika črka
`int isspace(int znak)`	je prazen prostor

Funkcije vrnejo vrednost različno od 0, če znak spada v opisano kategorijo znakov, in 0, če ne spada v to kategorijo.

`int tolower(int znak)`	pretvori v malo črko
`int toupper(int znak)`	pretvori v veliko črko

Operacije z nizi `<string.h>`

`int strlen(char *niz)`
Vrne dolžino podanega niza.

`int strcmp(char *niz1, char *niz2)`
Primerja niza `niz1` in `niz2`. Če sta enaka, vrne `0`, če je `niz1` po leksikografski urejenosti pred `niz2`, vrne vrednost, ki je manjša od `0`, sicer vrne vrednost, ki je večja od `0`.

`char *strcpy(char *niz1, char *niz2)`
Kopira `niz2` v `niz1`. Vrne `niz1`.

`char *strcat(char *niz1, char *niz2)`
Prilepi kopijo niza `niz2` na konec niza `niz1`. Vrne `niz1`.

`char *strchr(char *niz1, int znak)`
Poišče prvo pojavitev znaka v nizu. Vrne kazalec na prvo pojavitev znaka ali `NULL`, če znaka ni v nizu.

`char *strstr(char *niz1, char *niz2)`
Poišče prvo pojavitev niza `niz2` v nizu `niz1`. Vrne kazalec na prvo pojavitev `niz2` v `niz1` ali `NULL`, če take pojavitve ni.

Standardni vhod in izhod `<stdio.h>`

`int getchar()`
Prebere in vrne znak iz standardnega vhoda. Ob napaki vrne konstanto `EOF`.

C Programska kartica

`char *gets(char *niz1)`
Prebere znake iz standardnega vhoda do konca vrstice in jih zapiše v niz `niz1`. Vrne kazalec na `niz1`.

`int putchar(int znak)`
Zapiše `znak` na standardni izhod. Ob napaki vrne konstanto `EOF`, sicer vrne zapisani znak.

`int puts(char *niz1)`
Zapiše niz `niz1` na standardni izhod. Ob napaki vrne konstanto `EOF`, sicer vrne nenegativno vrednost.

Datoteke `<stdio.h>`

`FILE *dat;` deklaracija dat. kazalca

`stdin`, `stdout`, `stderr`
Predefinirani datotečni kazalci na standardni vhod, standardni izhod in standardni izhod za napake.

`FILE *fopen(char *ime_dat, char *nacin)`
Odpre datoteko, katere ime je podano z nizom `ime_dat`, in vrne datotečni kazalec odprte datoteke ali `NULL`, če pride do napake. Niz `nacin` določa način odprtja datoteke in sicer:

- `r` branje obstoječe datoteke
- `w` pisanje od začetka (zbriše obstoječe podatke)
- `a` dodajanje na konec datoteke
- `r+` branje/pisanje v obstoječo datoteko
- `w+` ustvari novo datoteko za branje/pisanje
- `a+` pisanje na konec datoteke, branje povsod
- `b` označuje binarne datoteke (npr. `rb` za branje binarne datoteke)

`int feof(FILE *dat)`
Vrne neničelno vrednost, če smo pri branju datoteke prišli do konca (end of file) ali `0`, če še nismo prišli do konca.

`int fclose(FILE *dat)`
Zapre datoteko, pred tem pa vanjo zapiše vse še nezapisane podatke. Vrne `0`, če je zapiranje uspešno, ob napaki vrne `EOF`.

`int fgetc(FILE *dat)` ali
`int getc(FILE *dat)`
Iz datoteke, določene z datotečnim kazalcem `dat`, prebere en znak in ga vrne. Ob napaki vrne konstanto `EOF`.

`char *fgets(char *niz1, int N, FILE *dat)`
Iz datoteke, ki jo določa datotečni kazalec `dat`, prebere eno vrstico. Če je vrstica daljša od `N-1` znakov, jih prebere le `N-1`. Znake zapiše v niz `niz1`. Funkcija vrne `niz1` ali `NULL`, če je pri branju prišlo do napake.

`int fputc(int znak, FILE *dat)` ali
`int putc(int znak, FILE *dat)`
Zapiše `znak` v datoteko. Ob uspešnem pisanju vrne zapisani znak, ob napaki vrne `EOF`.

`int fputs(char *niz, FILE *dat)`
Zapiše `niz` v datoteko. Ob uspešnem pisanju vrne `0`, ob napaki vrne `EOF`.

Formatiran vhod in izhod `<stdio.h>`

`int printf(char *format, [argumenti])`
Na standardni izhod izpiše niz, določen z nizom `format` in `argumenti`. Posebne oznake v formatu določajo način izpisa argumentov; nekatere oznake so: `%d` za celo število, `%c` za znak, `%s` za niz, `%f` za število s plavajočo vejico ... Pri formatu lahko uporabimo še dodatne oznake: `%`*`n`*`d` izpiše število na *n* mest; s piko določimo natančnost izpisa, kjer `%`*`n`*`.`*`dec`*`f` izpiše decimalno število z *dec* decimalkami, skupaj na *n* mest; znak za odstotke izpišemo z `%%`.
Funkcija vrne število izpisanih znakov.

`int fprintf(FILE *dat, char *format, [arg])`
Deluje enako kot funkcija `printf`, le da zapisuje v datoteko, določeno s kazalcem `dat`.

`int sprintf(char *niz1, char *format, [arg])`
Deluje enako kot funkcija `printf`, le da zapisuje v niz `niz1`. Funkcija vrne število bytov, zapisanih v `niz1`.

`int scanf(char *format, [argumenti])`
S standardnega vhoda prebere vrsto vnosov, kot jih določa niz `format`, ter jih po vrsti zapiše na naslove, ki so podani kot `argumenti`. `scanf` obravnava enega ali več praznih prostorov (presledek, tabulator, nova vrstica) kot en sam presledek, ta pa predstavlja ločilo med vnosi. Posebne oznake v formatu določajo tipe vnosov; nekatere oznake so: `%d` za celo število, `%c` za znak, `%s` za niz, `%f` za število s plavajočo vejico ... Funkcija vrne število prebranih vnosov.

`int fscanf(FILE *dat, char *format, [argum.])`
Deluje kot `scanf`, le da podatke bere iz datoteke, določene s kazalcem `dat`.

`int sscanf(char *niz, char *format, [argum.])`
Deluje kot `scanf`, le da podatke bere iz niza `niz1`.

Delo s pomnilnikom `<stdlib.h>`

`void *malloc(int size)`
Alocira `size` bytov prostora v pomnilniku in vrne kazalec na alocirani prostor. V kolikor alokacija ni uspešna, funkcija vrne `NULL`.

`void *calloc(int numEl, int sizeEl)`
Alocira pomnilnik za `numEl` elementov velikosti `sizeEl` bytov (efektivno `numEl*sizeEl` bytov). Funkcija vrne kazalec na alocirani prostor oz., v kolikor alokacija ni uspešna, `NULL`.

`void *realloc(void *buf, int size)`
Poveča ali zmanjša že alocirani pomnilniški prostor, na katerega kaže kazalec `buf`, na `size` bytov. Funkcija poskrbi, da vsebina prostora, na katerega kaže `buf`, ostane nespremenjena (če se prostor zmanjša, so podatki, ki segajo preko nove meje, izgubljeni). Funkcija vrne kazalec na realocirani prostor oz. `NULL`, če realokacija ni uspešna.

`void free(void *buf)`
Sprosti pomnilnik, na katerega kaže kazalec `buf`.

Delo s pomnilnikom `<string.h>`

`void *memcpy(void *buf1, void *buf2, int N)`
Prekopira `N` bytov iz pomnilniške lokacije, na katero kaže `buf2`, v `buf1`. Če se regiji prekrivata, obnašanje funkcije ni definirano. Funkcija vrne `buf1`.

`void *memmove(void *buf1, void *buf2, int N)`
Premakne `N` bytov iz pomnilniške lokacije, na katero kaže `buf2`, v `buf1`. Tudi če se regiji prekrivata, funkcija deluje pravilno. Funkcija vrne `buf1`.

`void *memset(void *buf, int C, int N)`
Funkcija nastavi prvih `N` bytov, na katere kaže kazalec `buf`, na vrednost `(unsigned char)C`. Funkcija vrne `buf`.