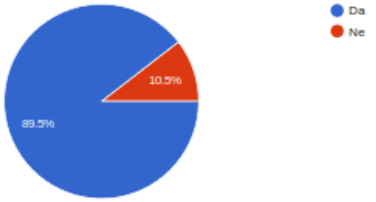
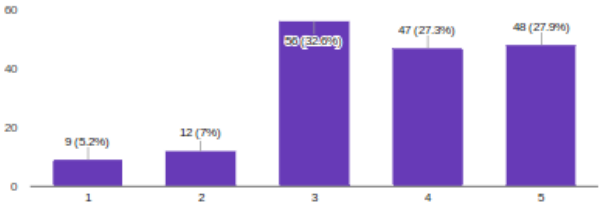
[PPI] 1. Masovne Instrukcije
2016/2017

Masovne

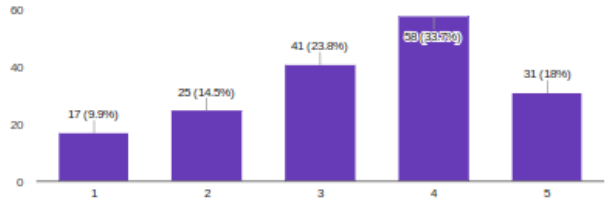
Dosao/la bi na masovne iz PPI-a? (172 responses)



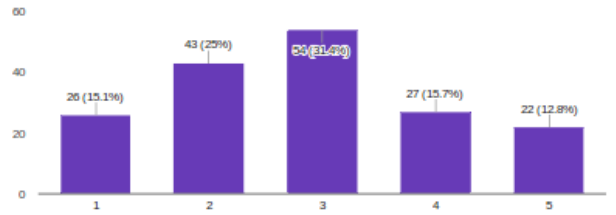
Kakve masovne želite? (172 responses)



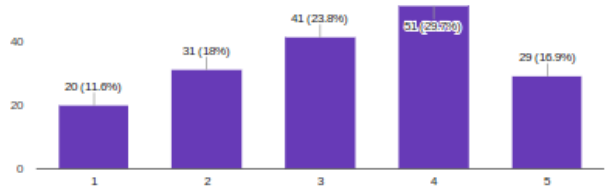
01 - Tipovi Podataka (172 responses)



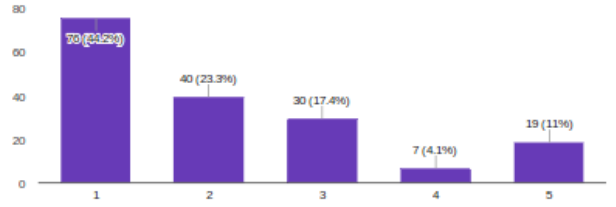
03 - Kontrolne Naredbe (172 responses)



02 - Ostali Operatori (172 responses)



04 - Polja (172 responses)



Why C Sucks!

44 Years Old
Compiled
Manual Memory Management
No run-time error checks

Why C ~~Sucks~~!
Rocks!

44 Years Old
Compiled
Manual Memory Management
No run-time error checks

Right Tool For The Job

C vs Python

hello.c Raw

```c
#include <stdio.h>
#define NAME_LENGHT 30

int main(void) {
  char name[NAME_LENGHT + 1];

  printf("What's your name? ");
  gets(name);

  printf("Hello %s!", name);
  return 0;
}
```

hello.py Raw

```python
#!/usr/bin/python

print "What is your name? "
name = input()
print "Hello " + name + "!"
```

C code is more than 2x longer than Python code

[1] http://blog.wolfram.com/2012/11/14/code-length-measured-in-14-languages/
[IMG] https://gist.github.com/anonymous/79e9e4d92653827c5a79c78dcd47ea28

Every C Program Ever

basic_structure.c — Raw

```c
/* Include statements and preprocessor definitions */
#include ...

/* Function prototypes and variable declarations */
int func(...);

int main(int argc, char *argv[]) {
  /* The body of the main function */

  return 0;
}

/* Definitions of all other functions*/
int func(...) {
  /* Body of some other function */
}
```

[IMG] https://gist.github.com/anonymous/20d9b222c0d79a61a1516e350244c0a2

Data Types

char unsigned char	1 byte
short unsigned short	2 bytes
int unsigned int	4 bytes
long unsigned long	4 bytes
float	4 bytes
double	8 bytes
long double	12 bytes

Data Types: Char, Short, Int, Long

Signed – Two's complement *(complement + 1)*

Char: [-128, 127]
Short: [-32768, 32767]
Int & Long: [-2147483648, 2147483647]
N-bits: $[-2^{n-1}, 2^{n-1}-1]$

Unsigned – ordinary binary representation

Char: [0, 255]
Short: [0, 65535]
Int & Long: [0, 4294967295]
N-bits: $[0, 2^{n}]$

Data Types

Data Types/Declarations

character (1 byte)	`char`
integer	`int`
float (single precision)	`float`
float (double precision)	`double`
short (16 bit integer)	`short`
long (32 bit integer)	`long`
positive and negative	`signed`
only positive	`unsigned`

C Reference Card (ANSI)

Program Structure/Functions

type fnc(*type*$_1$,...)	function declarations
type name	external variable declarations
`main() {`	main routine
declarations	local variable declarations
statements	
`}`	
type fnc(*arg*$_1$,...) {	function definition
declarations	local variable declarations
statements	
`return` *value*;	
`}`	
`/* */`	comments
`main(int argc, char *argv[])`	main with args
`exit(`*arg*`)`	terminate execution

C Preprocessor

include library file	`#include <`*filename*`>`
include user file	`#include "`*filename*`"`
replacement text	`#define` *name text*
replacement macro	`#define` *name*(*var*) *text*
Example. `#define max(A,B) ((A)>(B) ? (A) : (B))`	
undefine	`#undef` *name*
quoted string in replace	`#`
concatenate args and rescan	`##`
conditional execution	`#if, #else, #elif, #endif`
is *name* defined, not defined?	`#ifdef, #ifndef`
name defined?	`defined(`*name*`)`
line continuation char	\

Data Types/Declarations

character (1 byte)	`char`
integer	`int`
float (single precision)	`float`
float (double precision)	`double`
short (16 bit integer)	`short`
long (32 bit integer)	`long`
positive and negative	`signed`
only positive	`unsigned`
pointer to int, float,...	`*int, *float,...`
enumeration constant	`enum`
constant (unchanging) value	`const`
declare external variable	`extern`
register variable	`register`
local to source file	`static`
no value	`void`
structure	`struct`
create name by data type	`typedef` *typename*
size of an object (type is `size_t`)	`sizeof` *object*
size of a data type (type is `size_t`)	`sizeof(`*type name*`)`

Initialization

initialize variable	*type name=value*
initialize array	*type name*`[]={`*value*$_1$,...`}`
initialize char string	`char` *name*`[]="`*string*`"`

Constants

long (suffix)	`L` or `l`
float (suffix)	`F` or `f`
exponential form	`e`
octal (prefix zero)	`0`
hexadecimal (prefix zero-ex)	`0x` or `0X`
character constant (char, octal, hex)	`'a', '\ooo', '\xhh'`
newline, cr, tab, backspace	`\n, \r, \t, \b`
special characters	`\\, \?, \', \"`
string constant (ends with `'\0'`)	`"abc...de"`

Pointers, Arrays & Structures

declare pointer to *type*	*type* `*`*name*
declare function returning pointer to *type*	*type* `*f()`
declare pointer to function returning *type*	*type* `(*pf)()`
generic pointer type	`void *`
null pointer	`NULL`
object pointed to by *pointer*	`*`*pointer*
address of object *name*	`&`*name*
array	*name*`[`*dim*`]`
multi-dim array	*name*`[`*dim*$_1$`][`*dim*$_2$`]...`
Structures	
`struct` *tag* `{`	structure template
declarations	declaration of members
`};`	
create structure	`struct` *tag name*
member of structure from template	*name*`.`*member*
member of pointed to structure	*pointer* `->` *member*
Example. `(*p).x` and `p->x` are the same	
single value, multiple type structure	`union`
bit field with *b* bits	*member* `:` *b*

Operators (grouped by precedence)

structure member operator	*name*`.`*member*
structure pointer	*pointer*`->`*member*
increment, decrement	`++, --`
plus, minus, logical not, bitwise not	`+, -, !, ~`
indirection via pointer, address of object	`*`*pointer*`, &`*name*
cast expression to type	`(`*type*`)` *expr*
size of an object	`sizeof`
multiply, divide, modulus (remainder)	`*, /, %`
add, subtract	`+, -`
left, right shift [bit ops]	`<<, >>`
comparisons	`>, >=, <, <=`
comparisons	`==, !=`
bitwise and	`&`
bitwise exclusive or	`^`
bitwise or (incl)	`|`
logical and	`&&`
logical or	`||`
conditional expression	*expr*$_1$ `?` *expr*$_2$ `:` *expr*$_3$
assignment operators	`+=, -=, *=, ...`
expression evaluation separator	`,`

Unary operators, conditional expression and assignment operators group right to left; all others group left to right.

Flow of Control

statement terminator	`;`
block delimeters	`{ }`
exit from `switch`, `while`, `do`, `for`	`break`
next iteration of `while`, `do`, `for`	`continue`
go to	`goto` *label*
label	*label*`:`
return value from function	`return` *expr*
Flow Constructions	
`if` statement	`if (`*expr*`)` *statement* `else if (`*expr*`)` *statement* `else` *statement*
`while` statement	`while (`*expr*`)` *statement*
`for` statement	`for (`*expr*$_1$`;` *expr*$_2$`;` *expr*$_3$`)` *statement*
`do` statement	`do` *statement* `while(`*expr*`);`

`switch` statement	`switch (`*expr*`) {`
	`case` *const*$_1$`:` *statement*$_1$ `break;`
	`case` *const*$_2$`:` *statement*$_2$ `break;`
	`default:` *statement*
	`}`

ANSI Standard Libraries

`<assert.h>`	`<ctype.h>`	`<errno.h>`	`<float.h>`	`<limits.h>`
`<locale.h>`	`<math.h>`	`<setjmp.h>`	`<signal.h>`	`<stdarg.h>`
`<stddef.h>`	`<stdio.h>`	`<stdlib.h>`	`<string.h>`	`<time.h>`

Character Class Tests `<ctype.h>`

alphanumeric?	`isalnum(c)`
alphabetic?	`isalpha(c)`
control character?	`iscntrl(c)`
decimal digit?	`isdigit(c)`
printing character (not incl space)?	`isgraph(c)`
lower case letter?	`islower(c)`
printing character (incl space)?	`isprint(c)`
printing char except space, letter, digit?	`ispunct(c)`
space, formfeed, newline, cr, tab, vtab?	`isspace(c)`
upper case letter?	`isupper(c)`
hexadecimal digit?	`isxdigit(c)`
convert to lower case?	`tolower(c)`
convert to upper case?	`toupper(c)`

String Operations `<string.h>`

`s,t` are strings, `cs,ct` are constant strings

length of `s`	`strlen(s)`
copy `ct` to `s`	`strcpy(s,ct)`
up to `n` chars	`strncpy(s,ct,n)`
concatenate `ct` after `s`	`strcat(s,ct)`
up to `n` chars	`strncat(s,ct,n)`
compare `cs` to `ct`	`strcmp(cs,ct)`
only first `n` chars	`strncmp(cs,ct,n)`
pointer to first `c` in `cs`	`strchr(cs,c)`
pointer to last `c` in `cs`	`strrchr(cs,c)`
copy `n` chars from `ct` to `s`	`memcpy(s,ct,n)`
copy `n` chars from `ct` to `s` (may overlap)	`memmove(s,ct,n)`
compare `n` chars of `cs` with `ct`	`memcmp(cs,ct,n)`
pointer to first `c` in first `n` chars of `cs`	`memchr(cs,c,n)`
put `c` into first `n` chars of `cs`	`memset(s,c,n)`

Data Types: Float, Double

IEEE 754

$$(-1)^{sign} \times (1.fraction_{(2)}) \times 2^{(exponent)_{(2)}-127}$$

[sign] [exponent] [fraction]

[p] [karakteristika] [mantisa]

Float – [1][8][23]

$K = BE_{(2)} + 127$

$Max \approx 3.4 * 10^{38}$

$Min \approx 1.4 * 10^{-45}$

$|\rho| \leq 2^{-24} \approx 6 \times 10^{-8}$

Double – [1][11][52]

$K = BE_{(2)} + 1023$

$Max \approx 1.8 * 10^{308}$

$Min \approx 4.9 * 10^{-324}$

$|\rho| \leq 2^{-53} \approx 1.1 \times 10^{-16}$

Nula: $0 = 0...0_{(2)}$ $-0 = 10...0_{(2)}$

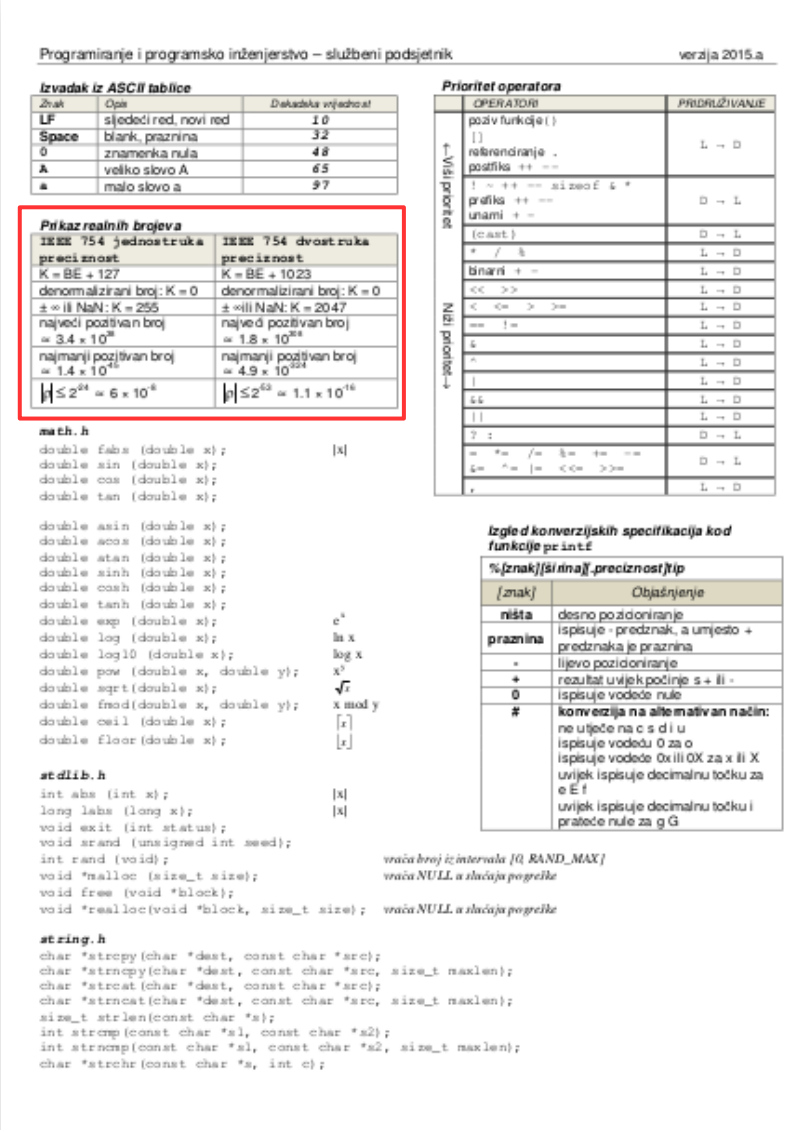
Dernormaliziran: $K = 0$

$+\infty$ / $-\infty$: $K = 1...1_{(2)}$ $F = 0...0_{(2)}$

Data Types: Float, Double

Prikaz realnih brojeva

IEEE 754 jednostruka preciznost	**IEEE 754 dvostruka preciznost**
K = BE + 127	K = BE + 1023
denormalizirani broj: K = 0	denormalizirani broj: K = 0
± ∞ ili NaN: K = 255	± ∞ili NaN: K = 2047
najveći pozitivan broj $\approx 3.4 \times 10^{38}$	najveći pozitivan broj $\approx 1.8 \times 10^{308}$
najmanji pozitivan broj $\approx 1.4 \times 10^{-45}$	najmanji pozitivan broj $\approx 4.9 \times 10^{-324}$
$\lvert\rho\rvert \leq 2^{-24} \approx 6 \times 10^{-8}$	$\lvert\rho\rvert \leq 2^{-53} \approx 1.1 \times 10^{-16}$



Programiranje i programsko inženjerstvo – službeni podsjetnik verzija 2015.a

Izvadak iz ASCII tablice

Znak	*Opis*	*Dekadska vrijednost*
LF	sljedeći red, novi red	*10*
Space	blank, praznina	*32*
0	znamenka nula	*48*
A	veliko slovo A	*65*
a	malo slovo a	*97*

Prikaz realnih brojeva

IEEE 754 jednostruka preciznost	**IEEE 754 dvostruka preciznost**
K = BE + 127	K = BE + 1023
denormalizirani broj: K = 0	denormalizirani broj: K = 0
± ∞ ili NaN: K = 255	± ∞ili NaN: K = 2047
najveći pozitivan broj $\approx 3.4 \times 10^{38}$	najveći pozitivan broj $\approx 1.8 \times 10^{308}$
najmanji pozitivan broj $\approx 1.4 \times 10^{-45}$	najmanji pozitivan broj $\approx 4.9 \times 10^{-324}$
$\lvert\rho\rvert \leq 2^{-24} \approx 6 \times 10^{-8}$	$\lvert\rho\rvert \leq 2^{-53} \approx 1.1 \times 10^{-16}$

Prioritet operatora

	OPERATOR	*PRIDRUŽIVANJE*
← Viši prioritet	poziv funkcije ( ) [ ] referenciranje . postfiks ++ --	L → D
	! ~ ++ -- sizeof & * prefiks ++ -- unarni + -	D → L
	(cast)	D → L
	* / %	L → D
	binarni + -	L → D
	<< >>	L → D
	< <= > >=	L → D
	== !=	L → D
	&	L → D
	^	L → D
	|	L → D
	&&	L → D
	||	L → D
	? :	D → L
	= *= /= %= += -= &= ^= |= <<= >>=	D → L
Niži prioritet →	,	L → D

math.h

```
double fabs (double x);                 |x|
double sin (double x);
double cos (double x);
double tan (double x);

double asin (double x);
double acos (double x);
double atan (double x);
double sinh (double x);
double cosh (double x);
double tanh (double x);
double exp (double x);                  e^x
double log (double x);                  ln x
double log10 (double x);                log x
double pow (double x, double y);        x^y
double sqrt(double x);                  √x
double fmod(double x, double y);        x mod y
double ceil (double x);                 ⌈x⌉
double floor(double x);                 ⌊x⌋
```

stdlib.h

```
int abs (int x);                        |x|
long labs (long x);                     |x|
void exit (int status);
void srand (unsigned int seed);
int rand (void);                        vraća broj iz intervala [0, RAND_MAX]
void *malloc (size_t size);             vraća NULL u slučaju pogreške
void free (void *block);
void *realloc(void *block, size_t size); vraća NULL u slučaju pogreške
```

string.h

```
char *strcpy(char *dest, const char *src);
char *strncpy(char *dest, const char *src, size_t maxlen);
char *strcat(char *dest, const char *src);
char *strncat(char *dest, const char *src, size_t maxlen);
size_t strlen(const char *s);
int strcmp(const char *s1, const char *s2);
int strncmp(const char *s1, const char *s2, size_t maxlen);
char *strchr(const char *s, int c);
```

***Izgled konverzijskih specifikacija kod funkcije* `printf`**

%[znak][širina][.preciznost]tip

[znak]	*Objašnjenje*
ništa	desno pozicioniranje
praznina	ispisuje - predznak, a umjesto + predznaka je praznina
-	lijevo pozicioniranje
+	rezultat uvijek počinje s + ili -
0	ispisuje vodeće nule
#	**konverzija na alternativan način:** ne utječe na c s d i u ispisuje vodeću 0 za o ispisuje vodeće 0x ili 0X za x ili X uvijek ispisuje decimalnu točku za e E f uvijek ispisuje decimalnu točku i prateće nule za g G

Data Types: Char & ASCII

Decimal	Hexadecimal	Binary	Octal	Char
0	0	0	0	[NULL]
1	1	1	1	[START OF HEADING]
2	2	10	2	[START OF TEXT]
3	3	11	3	[END OF TEXT]
4	4	100	4	[END OF TRANSMISSION]
5	5	101	5	[ENQUIRY]
6	6	110	6	[ACKNOWLEDGE]
7	7	111	7	[BELL]
8	8	1000	10	[BACKSPACE]
9	9	1001	11	[HORIZONTAL TAB]
10	A	1010	12	[LINE FEED]
11	B	1011	13	[VERTICAL TAB]
12	C	1100	14	[FORM FEED]
13	D	1101	15	[CARRIAGE RETURN]
14	E	1110	16	[SHIFT OUT]
15	F	1111	17	[SHIFT IN]
16	10	10000	20	[DATA LINK ESCAPE]
17	11	10001	21	[DEVICE CONTROL 1]
18	12	10010	22	[DEVICE CONTROL 2]
19	13	10011	23	[DEVICE CONTROL 3]
20	14	10100	24	[DEVICE CONTROL 4]
21	15	10101	25	[NEGATIVE ACKNOWLEDGE]
22	16	10110	26	[SYNCHRONOUS IDLE]
23	17	10111	27	[ENG OF TRANS. BLOCK]
24	18	11000	30	[CANCEL]
25	19	11001	31	[END OF MEDIUM]
26	1A	11010	32	[SUBSTITUTE]
27	1B	11011	33	[ESCAPE]
28	1C	11100	34	[FILE SEPARATOR]
29	1D	11101	35	[GROUP SEPARATOR]
30	1E	11110	36	[RECORD SEPARATOR]
31	1F	11111	37	[UNIT SEPARATOR]
32	20	100000	40	[SPACE]
33	21	100001	41	!
34	22	100010	42	"
35	23	100011	43	#
36	24	100100	44	$
37	25	100101	45	%
38	26	100110	46	&
39	27	100111	47	'
40	28	101000	50	(
41	29	101001	51	)
42	2A	101010	52	*
43	2B	101011	53	+
44	2C	101100	54	,
45	2D	101101	55	-
46	2E	101110	56	.
47	2F	101111	57	/
48	30	110000	60	0
49	31	110001	61	1
50	32	110010	62	2
51	33	110011	63	3
52	34	110100	64	4
53	35	110101	65	5
54	36	110110	66	6
55	37	110111	67	7
56	38	111000	70	8
57	39	111001	71	9
58	3A	111010	72	:
59	3B	111011	73	;
60	3C	111100	74	<
61	3D	111101	75	=
62	3E	111110	76	>
63	3F	111111	77	?
64	40	1000000	100	@
65	41	1000001	101	A
66	42	1000010	102	B
67	43	1000011	103	C
68	44	1000100	104	D
69	45	1000101	105	E
70	46	1000110	106	F
71	47	1000111	107	G
72	48	1001000	110	H
73	49	1001001	111	I
74	4A	1001010	112	J
75	4B	1001011	113	K
76	4C	1001100	114	L
77	4D	1001101	115	M
78	4E	1001110	116	N
79	4F	1001111	117	O
80	50	1010000	120	P
81	51	1010001	121	Q
82	52	1010010	122	R
83	53	1010011	123	S
84	54	1010100	124	T
85	55	1010101	125	U
86	56	1010110	126	V
87	57	1010111	127	W
88	58	1011000	130	X
89	59	1011001	131	Y
90	5A	1011010	132	Z
91	5B	1011011	133	[
92	5C	1011100	134	\
93	5D	1011101	135	]
94	5E	1011110	136	^
95	5F	1011111	137	_
96	60	1100000	140	`
97	61	1100001	141	a
98	62	1100010	142	b
99	63	1100011	143	c
100	64	1100100	144	d
101	65	1100101	145	e
102	66	1100110	146	f
103	67	1100111	147	g
104	68	1101000	150	h
105	69	1101001	151	i
106	6A	1101010	152	j
107	6B	1101011	153	k
108	6C	1101100	154	l
109	6D	1101101	155	m
110	6E	1101110	156	n
111	6F	1101111	157	o
112	70	1110000	160	p
113	71	1110001	161	q
114	72	1110010	162	r
115	73	1110011	163	s
116	74	1110100	164	t
117	75	1110101	165	u
118	76	1110110	166	v
119	77	1110111	167	w
120	78	1111000	170	x
121	79	1111001	171	y
122	7A	1111010	172	z
123	7B	1111011	173	{
124	7C	1111100	174	|
125	7D	1111101	175	}
126	7E	1111110	176	~
127	7F	1111111	177	[DEL]

Data Types: Constant Storage

5	00000000000000000000000000000101	int
‘5’	00110101	char (from ASCII)
5.	0100000000010100000.....0	double (8 byte)
5.f	010000001010000....0	float (4 byte)
05	00000000000000000000000000000101	octal (4 byte)
0x5	00000000000000000000000000000101	hex (4 bytes)
“5”	00110101 00000000	char ary (2x1 byte)

- Poseban problem studenti imaju u predočavanju raznih "nula"

```
  0    00000000000000000000000000000000  (cjelobrojna konstanta u 32 bita)
'0'    00110000                          (znakovna konstanta u 8 bita)
'\0'   00000000          (znakovna konstanta "nul-karakter", u 8 bita)
"0"    0011000000000000  (konstantni znakovni niz, 2 puta po 8 bita)
```

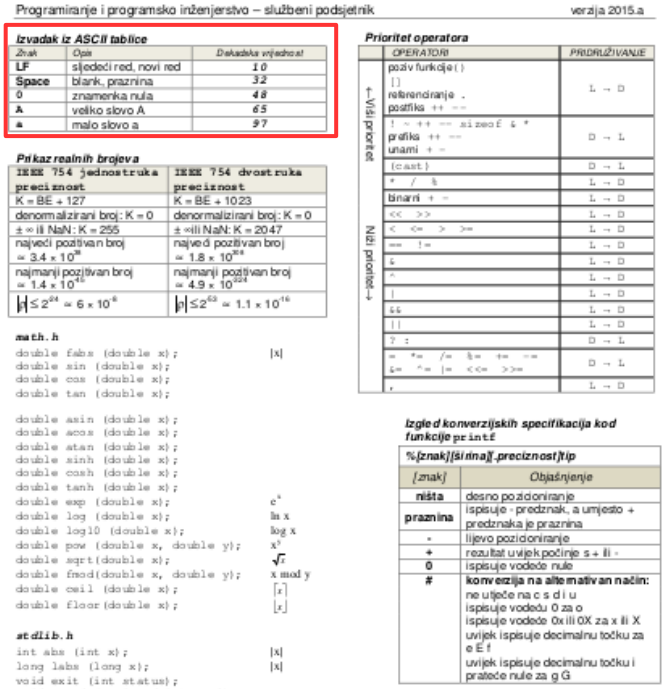
Data Types: Constants & ASCII

Izvadak iz ASCII tablice

Znak	*Opis*	*Dekadska vrijednost*
LF	sljedeći red, novi red	***10***
Space	blank, praznina	***32***
0	znamenka nula	***48***
A	veliko slovo A	***65***
a	malo slovo a	***97***

Constants

long (suffix)	`L` or `l`
float (suffix)	`F` or `f`
exponential form	`e`
octal (prefix zero)	`0`
hexadecimal (prefix zero-ex)	`0x` or `0X`
character constant (char, octal, hex)	`'a'`, `'\ooo'`, `'\xhh'`
newline, cr, tab, backspace	`\n`, `\r`, `\t`, `\b`
special characters	`\\`, `\?`, `\'`, `\"`
string constant (ends with `'\0'`)	`"abc...de"`



Programiranje i programsko inženjerstvo – službeni podsjetnik verzija 2015.a

Izvadak iz ASCII tablice

Znak	*Opis*	*Dekadska vrijednost*
LF	sljedeći red, novi red	*10*
Space	blank, praznina	*32*
0	znamenka nula	*48*
A	veliko slovo A	*65*
a	malo slovo a	*97*

Prikaz realnih brojeva

IEEE 754 jednostruka preciznost	IEEE 754 dvostruka preciznost
K = BE + 127	K = BE + 1023
denormalizirani broj: K = 0	denormalizirani broj: K = 0
± ∞ ili NaN: K = 255	± ∞ ili NaN: K = 2047
najveći pozitivan broj ≈ 3.4×10^{38}	najveći pozitivan broj ≈ 1.8×10^{308}
najmanji pozitivan broj ≈ 1.4×10^{-45}	najmanji pozitivan broj ≈ 4.9×10^{-324}
$\lvert x \rvert \leq 2^{24} \approx 6 \times 10^{8}$	$\lvert x \rvert \leq 2^{53} \approx 1.1 \times 10^{16}$

Prioritet operatora

	OPERATOR	PRIDRUŽIVANJE
↑ Viši prioritet	poziv funkcije `()`	
	`[]`	
	referenciranje `.`	L → D
	postfiks `++ --`	
	`! ~ ++ -- sizeof & *`	
	prefiks `++ --`	D → L
	unarni `+ -`	
	`(cast)`	D → L
	`* / %`	L → D
	binarni `+ -`	L → D
	`<< >>`	L → D
	`< <= > >=`	L → D
	`== !=`	L → D
	`&`	L → D
	`^`	L → D
	`|`	L → D
	`&&`	L → D
	`||`	L → D
	`? :`	D → L
	`= *= /= %= += -= &= ^= |= <<= >>=`	D → L
↓ Niži prioritet	`,`	L → D

Izgled konverzijskih specifikacija kod funkcije `printf`

%[znak][širina][.preciznost]tip	
[znak]	*Objašnjenje*
ništa	desno pozicioniranje
praznina	ispisuje - predznak, a umjesto + predznaka je praznina
-	lijevo pozicioniranje
+	rezultat uvijek počinje s + ili -
0	ispisuje vodeće nule
#	konverzija na alternativan način: ne utječe na c s d i u ispisuje vodeću 0 za o ispisuje vodeće 0x ili 0X za x ili X uvijek ispisuje decimalnu točku za e E f uvijek ispisuje decimalnu točku i pratеće nule za g G

`math.h`

```
double fabs (double x);                 |x|
double sin (double x);
double cos (double x);
double tan (double x);

double asin (double x);
double acos (double x);
double atan (double x);
double sinh (double x);
double cosh (double x);
double tanh (double x);
double exp (double x);                  e^x
double log (double x);                  ln x
double log10 (double x);                log x
double pow (double x, double y);        x^y
double sqrt(double x);                  √x
double fmod(double x, double y);        x mod y
double ceil (double x);                 ⌈x⌉
double floor(double x);                 ⌊x⌋
```

`stdlib.h`

```
int abs (int x);                        |x|
long labs (long x);                     |x|
void exit (int status);
void srand (unsigned int seed);
int rand (void);                        vraća broj iz intervala [0, RAND_MAX]
void *malloc (size_t size);             vraća NULL a služi za pogreške
void free (void *block);
void *realloc(void *block, size_t size); vraća NULL a služi za pogreške
```

`string.h`

```
char *strcpy(char *dest, const char *src);
char *strncpy(char *dest, const char *src, size_t maxlen);
char *strcat (char *dest, const char *src);
char *strncat(char *dest, const char *src, size_t maxlen);
size_t strlen(const char *s);
int strcmp(const char *s1, const char *s2);
int strncmp(const char *s1, const char *s2, size_t maxlen);
char *strchr(const char *s, int c);
```

C Reference Card (ANSI)

Program Structure/Functions

type fnc`(`*type*$_1$`,...)`	function declarations
type name	external variable declarations
`main() {`	main routine
declarations	local variable declarations
statements	
`}`	
type fnc`(`*arg*$_1$`,...) {`	function definition
declarations	local variable declarations
statements	
`return` *value*`;`	
`}`	
`/* */`	comments
`main(int argc, char *argv[])`	main with args
`exit(`*arg*`)`	terminate execution

C Preprocessor

include library file	`#include <`*filename*`>`
include user file	`#include "`*filename*`"`
replacement text	`#define` *name text*
replacement macro	`#define` *name*`(`*var*`)` *text*
Example. `#define max(A,B) ((A)>(B) ? (A) : (B))`	
undefine	`#undef` *name*
quoted string in replace	`#`
concatenate args and rescan	`##`
conditional execution	`#if, #else, #elif, #endif`
is *name* defined, not defined?	`#ifdef, #ifndef`
name defined?	`defined(`*name*`)`
line continuation char	`\`

Data Types/Declarations

character (1 byte)	`char`
integer	`int`
float (single precision)	`float`
float (double precision)	`double`
short (16 bit integer)	`short`
long (32 bit integer)	`long`
positive and negative	`signed`
only positive	`unsigned`
pointer to `int, float,...`	`*int, *float,...`
enumeration constant	`enum`
constant (unchanging) value	`const`
declare external variable	`extern`
register variable	`register`
local to source file	`static`
no value	`void`
structure	`struct`
create name by data type	`typedef` *typename*
size of an object (type is `size_t`)	`sizeof` *object*
size of a data type (type is `size_t`)	`sizeof(`*type name*`)`

Initialization

initialize variable	*type name*`=`*value*
initialize array	*type name*`[]={`*value*$_1$`,...}`
initialize char string	`char` *name*`[]="`*string*`"`

Constants

long (suffix)	`L` or `l`
float (suffix)	`F` or `f`
exponential form	`e`
octal (prefix zero)	`0`
hexadecimal (prefix zero-ex)	`0x` or `0X`
character constant (char, octal, hex)	`'a', '\ooo', '\xhh'`
newline, cr, tab, backspace	`\n, \r, \t, \b`
special characters	`\\, \?, \', \"`
string constant (ends with `'\0'`)	`"abc...de"`

Pointers, Arrays & Structures

declare pointer to *type*	*type* `*`*name*
declare function returning pointer to *type*	*type* `*f()`
declare pointer to function returning *type*	*type* `(*pf)()`
generic pointer type	`void *`
null pointer	`NULL`
object pointed to by *pointer*	`*`*pointer*
address of object *name*	`&`*name*
array	*name*`[dim]`
multi-dim array	*name*`[dim`$_1$`][dim`$_2$`]...`
Structures	
`struct` *tag* `{` *declarations* `};`	structure template
create structure	`struct` *tag name*
member of structure from template	*name*`.`*member*
member of pointed to structure	*pointer* `->` *member*
Example. `(*p).x` and `p->x` are the same	
single value, multiple type structure	`union`
bit field with *b* bits	*member* `:` *b*

Operators (grouped by precedence)

structure member operator	*name*`.`*member*
structure pointer	*pointer*`->`*member*
increment, decrement	`++, --`
plus, minus, logical not, bitwise not	`+, -, !, ~`
indirection via pointer, address of object	`*`*pointer*`, &`*name*
cast expression to type	`(`*type*`)` *expr*
size of an object	`sizeof`
multiply, divide, modulus (remainder)	`*, /, %`
add, subtract	`+, -`
left, right shift [bit ops]	`<<, >>`
comparisons	`>, >=, <, <=`
comparisons	`==, !=`
bitwise and	`&`
bitwise exclusive or	`^`
bitwise or (incl)	`|`
logical and	`&&`
logical or	`||`
conditional expression	*expr*$_1$ `?` *expr*$_2$ `:` *expr*$_3$
assignment operators	`+=, -=, *=, ...`
expression evaluation separator	`,`

Unary operators, conditional expression and assignment operators group right to left; all others group left to right.

Flow of Control

statement terminator	`;`
block delimeters	`{ }`
exit from `switch, while, do, for`	`break`
next iteration of `while, do, for`	`continue`
go to	`goto` *label*
label	*label*`:`
return value from function	`return` *expr*
Flow Constructions	
`if` statement	`if (`*expr*`)` *statement* `else if (`*expr*`)` *statement* `else` *statement*
`while` statement	`while (`*expr*`)` *statement*
`for` statement	`for (`*expr*$_1$`;` *expr*$_2$`;` *expr*$_3$`)` *statement*
`do` statement	`do` *statement* `while(`*expr*`);`

`switch` statement	`switch (`*expr*`) {`
	`case` *const*$_1$`:` *statement*$_1$ `break;`
	`case` *const*$_2$`:` *statement*$_2$ `break;`
	`default:` *statement*
	`}`

ANSI Standard Libraries

`<assert.h>`	`<ctype.h>`	`<errno.h>`	`<float.h>`	`<limits.h>`
`<locale.h>`	`<math.h>`	`<setjmp.h>`	`<signal.h>`	`<stdarg.h>`
`<stddef.h>`	`<stdio.h>`	`<stdlib.h>`	`<string.h>`	`<time.h>`

Character Class Tests `<ctype.h>`

alphanumeric?	`isalnum(c)`
alphabetic?	`isalpha(c)`
control character?	`iscntrl(c)`
decimal digit?	`isdigit(c)`
printing character (not incl space)?	`isgraph(c)`
lower case letter?	`islower(c)`
printing character (incl space)?	`isprint(c)`
printing char except space, letter, digit?	`ispunct(c)`
space, formfeed, newline, cr, tab, vtab?	`isspace(c)`
upper case letter?	`isupper(c)`
hexadecimal digit?	`isxdigit(c)`
convert to lower case?	`tolower(c)`
convert to upper case?	`toupper(c)`

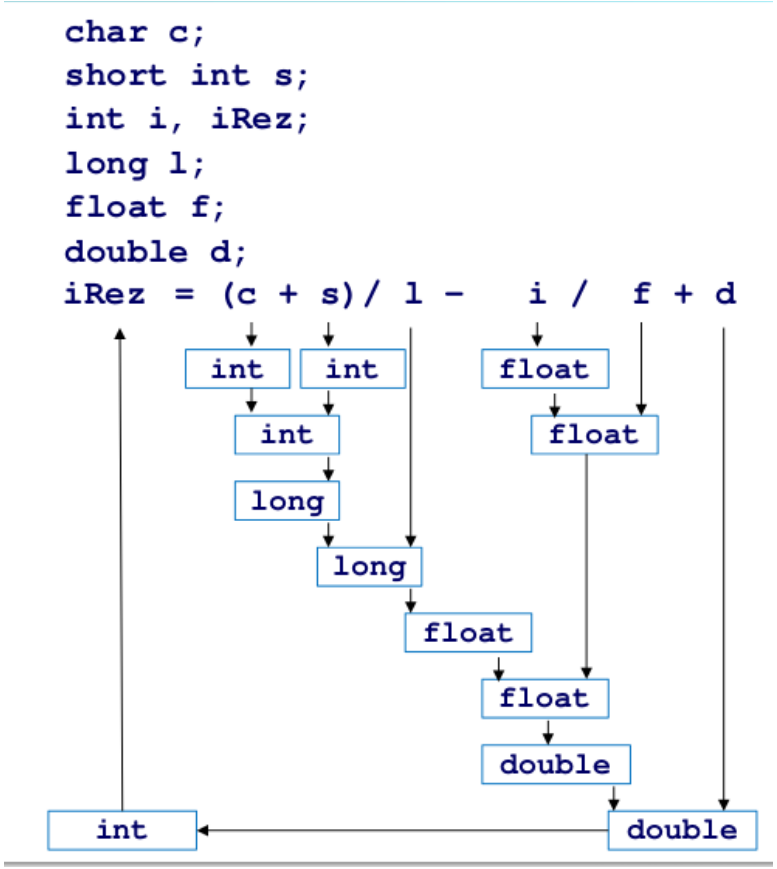
String Operations `<string.h>`

s,t are strings, cs,ct are constant strings

length of s	`strlen(s)`
copy ct to s	`strcpy(s,ct)`
up to n chars	`strncpy(s,ct,n)`
concatenate ct after s	`strcat(s,ct)`
up to n chars	`strncat(s,ct,n)`
compare cs to ct	`strcmp(cs,ct)`
only first n chars	`strncmp(cs,ct,n)`
pointer to first c in cs	`strchr(cs,c)`
pointer to last c in cs	`strrchr(cs,c)`
copy n chars from ct to s	`memcpy(s,ct,n)`
copy n chars from ct to s (may overlap)	`memmove(s,ct,n)`
compare n chars of cs with ct	`memcmp(cs,ct,n)`
pointer to first c in first n chars of cs	`memchr(cs,c,n)`
put c into first n chars of cs	`memset(s,c,n)`

Data Types: Type Casting

char → short → int
int < float < double < long double



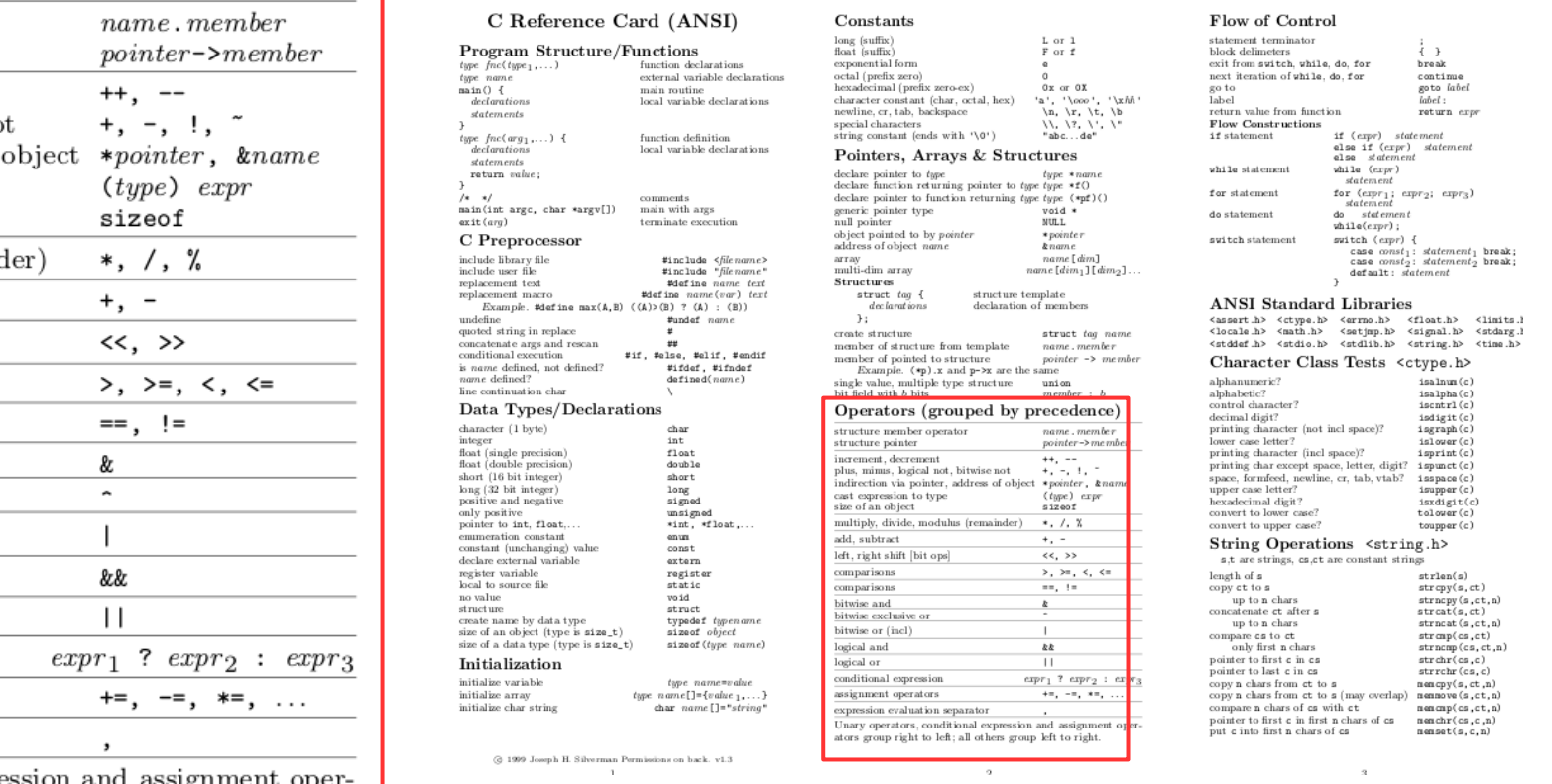

```
int n = (int )5.3

n = 5

Double a = 1 / 2

a = 0
```

Operators

Operators (grouped by precedence)

structure member operator structure pointer	*name.member* *pointer->member*
increment, decrement	`++, --`
plus, minus, logical not, bitwise not	`+, -, !, ~`
indirection via pointer, address of object	`*`*pointer*`, &`*name*
cast expression to type	`(`*type*`)` *expr*
size of an object	`sizeof`
multiply, divide, modulus (remainder)	`*, /, %`
add, subtract	`+, -`
left, right shift [bit ops]	`<<, >>`
comparisons	`>, >=, <, <=`
comparisons	`==, !=`
bitwise and	`&`
bitwise exclusive or	`^`
bitwise or (incl)	`|`
logical and	`&&`
logical or	`||`
conditional expression	$expr_1$? $expr_2$: $expr_3$
assignment operators	`+=, -=, *=, ...`
expression evaluation separator	`,`

Unary operators, conditional expression and assignment operators group right to left; all others group left to right.



C Reference Card (ANSI)

Program Structure/Functions

type fnc(type1,...) function declarations
type name external variable declarations
main() { main routine
declarations local variable declarations
statements
}
type fnc(arg1,...) { function definition
declarations local variable declarations
statements
return value;
}
/* */ comments
main(int argc, char *argv[]) main with args
exit(any) terminate execution

C Preprocessor

include library file #include <filename>
include user file #include "filename"
replacement text #define name text
replacement macro #define name(var) text
Example. #define max(A,B) ((A)>(B) ? (A) : (B))
undefine #undef name
quoted string in replace #
concatenate args and rescan ##
conditional execution #if, #else, #elif, #endif
is name defined, not defined? #ifdef, #ifndef
name defined? defined(name)
line continuation char \

Data Types/Declarations

character (1 byte) char
integer int
float (single precision) float
float (double precision) double
short (16 bit integer) short
long (32 bit integer) long
positive and negative signed
only positive unsigned
pointer to int, float,... *int, *float,...
enumeration constant enum
constant (unchanging) value const
declare external variable extern
register variable register
local to source file static
no value void
structure struct
create name by data type typedef typename
size of an object (type is size_t) sizeof object
size of a data type (type is size_t) sizeof(type name)

Initialization

initialize variable type name=value
initialize array type name[]={value1,...}
initialize char string char name[]="string"

© 1999 Joseph H. Silverman Permissions on back. v1.3

1

Constants

long (suffix) L or l
float (suffix) F or f
exponential form e
octal (prefix zero) 0
hexadecimal (prefix zero-ex) 0x or 0X
character constant (char, octal, hex) 'a', '\ooo', '\xhh'
newline, cr, tab, backspace \n, \r, \t, \b
special characters \\, \?, \', \"
string constant (ends with '\0') "abc...de"

Pointers, Arrays & Structures

declare pointer to type type *name
declare function returning pointer to type type *f()
declare pointer to function returning type type (*pf)()
generic pointer type void *
null pointer NULL
object pointed to by pointer *pointer
address of object name &name
array name[dim]
multi-dim array name[dim1][dim2]...

Structures

struct tag { structure template
declarations declaration of members
};
create structure struct tag name
member of structure from template name.member
member of pointed to structure pointer -> member
Example. (*p).x and p->x are the same
single value, multiple type structure union
bit field with b bits member : b

Operators (grouped by precedence)

structure member operator name.member
structure pointer pointer->member
increment, decrement ++, --
plus, minus, logical not, bitwise not +, -, !, ~
indirection via pointer, address of object *pointer, &name
cast expression to type (type) expr
size of an object sizeof
multiply, divide, modulus (remainder) *, /, %
add, subtract +, -
left, right shift [bit ops] <<, >>
comparisons >, >=, <, <=
comparisons ==, !=
bitwise and &
bitwise exclusive or ^
bitwise or (incl) |
logical and &&
logical or ||
conditional expression expr1 ? expr2 : expr3
assignment operators +=, -=, *=, ...
expression evaluation separator ,
Unary operators, conditional expression and assignment operators group right to left; all others group left to right.

2

Flow of Control

statement terminator ;
block delimeters { }
exit from switch, while, do, for break
next iteration of while, do, for continue
go to goto label
label label:
return value from function return expr

Flow Constructions

if statement if (expr) statement
else if (expr) statement
else statement
while statement while (expr)
statement
for statement for (expr1; expr2; expr3)
statement
do statement do statement
while(expr);
switch statement switch (expr) {
case const1: statement1 break;
case const2: statement2 break;
default: statement
}

ANSI Standard Libraries

<assert.h> <ctype.h> <errno.h> <float.h> <limits.h>
<locale.h> <math.h> <setjmp.h> <signal.h> <stdarg.h>
<stddef.h> <stdio.h> <stdlib.h> <string.h> <time.h>

Character Class Tests <ctype.h>

alphanumeric? isalnum(c)
alphabetic? isalpha(c)
control character? iscntrl(c)
decimal digit? isdigit(c)
printing character (not incl space)? isgraph(c)
lower case letter? islower(c)
printing character (incl space)? isprint(c)
printing char except space, letter, digit? ispunct(c)
space, formfeed, newline, cr, tab, vtab? isspace(c)
upper case letter? isupper(c)
hexadecimal digit? isxdigit(c)
convert to lower case? tolower(c)
convert to upper case? toupper(c)

String Operations <string.h>

s,t are strings, cs,ct are constant strings

length of s strlen(s)
copy ct to s strcpy(s,ct)
up to n chars strncpy(s,ct,n)
concatenate ct after s strcat(s,ct)
up to n chars strncat(s,ct,n)
compare cs to ct strcmp(cs,ct)
only first n chars strncmp(cs,ct,n)
pointer to first c in cs strchr(cs,c)
pointer to last c in cs strrchr(cs,c)
copy n chars from ct to s memcpy(s,ct,n)
copy n chars from ct to s (may overlap) memmove(s,ct,n)
compare n chars of cs with ct memcmp(cs,ct,n)
pointer to first c in first n chars of cs memchr(cs,c,n)
put c into first n chars of cs memset(s,c,n)

3

Operators: Increment & Decrement

```
int i = 2, j;

j = --i;
```

```
i = 1
j = 1
```

```
int i = 2, j;

j = i++;
```

```
i = 3
j = 2
```

Operators:
Plus, Minus, Logical Not, Bit Not

```
float x = 5.0f, y = 2.0f;
        + x + y          →  7.0f
        +x + +y          →  7.0f
        +x + + y         →  7.0f
        - x + - y        → -7.0f
        -x - -y          → -3.0f
        - x - - y        → -3.0f
```

+x NIJE apsolutna vrijednost od x

```
x = -5.0f;
        +x               →  -5.0f
```

Logical Not (!)

```
int i = 0;

!i = 1
!i = 0
```

Bit Not (~)

```
int char = 0;

~i = -128
```

Operators: Math

```
*, /, %,
  +, -
```

Operators: Bit Ops

```
   2 << 2 = 8
0010 << 2 = 1000

   4 >> 1 = 2
0100 >> 1 = 0010
```

Operators: Comparators

```
<, >, <=, >=
==, !=
```

Operators: Bitwise

```
And - &
0111 & 1010 = 0010

XOR - ^
0110 ^ 0101 = 0011

Or - |
0101 | 1100 = 1101
```

Operators: Logical

And - &&

Or - ||

DIFFERENT FROM BITWISE!!!!

Operators: Conditional (Ternary)

```
expr ? true : false
```

Operators: Asignment

=

+=, -=, *=, /=, %=,
<<=, >>=, &=, ^=, |=

```
n #= 2    n = n # 2
```

Operators: Asignment

=

+=, -=, *=, /=, %=,
<<=, >>=, &=, ^=, |=

```
n #= 2    n = n # 2
```

Flow of Control

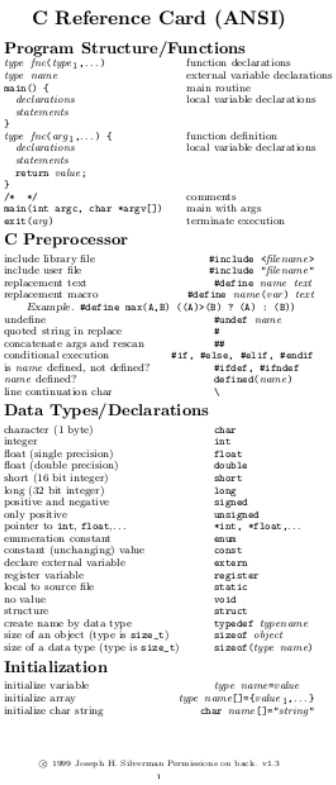
Flow of Control

statement terminator	`;`
block delimeters	`{ }`
exit from `switch`, `while`, `do`, `for`	`break`
next iteration of `while`, `do`, `for`	`continue`
go to	`goto` *label*
label	*label*`:`
return value from function	`return` *expr*

Flow Constructions

`if` statement	`if (`*expr*`)` *statement* `else if (`*expr*`)` *statement* `else` *statement*
`while` statement	`while (`*expr*`)` *statement*
`for` statement	`for (`$expr_1$`;` $expr_2$`;` $expr_3$`)` *statement*
`do` statement	`do` *statement* `while(`*expr*`);`

`switch` statement	`switch (`*expr*`) {`
	`case` $const_1$`:` $statement_1$ `break;`
	`case` $const_2$`:` $statement_2$ `break;`
	`default:` *statement*
	`}`



C Reference Card (ANSI)

Program Structure/Functions

type *fnc*(*type*$_1$,...)	function declarations
type *name*	external variable declarations
`main() {`	main routine
declarations	local variable declarations
statements	
`}`	
type *fnc*(*arg*$_1$,...) `{`	function definition
declarations	local variable declarations
statements	
`return` *value*`;`	
`}`	
`/* */`	comments
`main(int argc, char *argv[])`	main with args
`exit(`*arg*`)`	terminate execution

C Preprocessor

include library file	`#include <`*filename*`>`
include user file	`#include "`*filename*`"`
replacement text	`#define` *name text*
replacement macro	`#define` *name*`(`*var*`)` *text*
Example. `#define max(A,B) ((A)>(B) ? (A) : (B))`	
undefine	`#undef` *name*
quoted string in replace	`#`
concatenate args and rescan	`##`
conditional execution	`#if, #else, #elif, #endif`
is *name* defined, not defined?	`#ifdef, #ifndef`
name defined?	`defined(`*name*`)`
line continuation char	`\`

Data Types/Declarations

character (1 byte)	`char`
integer	`int`
float (single precision)	`float`
float (double precision)	`double`
short (16 bit integer)	`short`
long (32 bit integer)	`long`
positive and negative	`signed`
only positive	`unsigned`
pointer to `int`, `float`,...	`*int, *float,...`
enumeration constant	`enum`
constant (unchanging) value	`const`
declare external variable	`extern`
register variable	`register`
local to source file	`static`
no value	`void`
structure	`struct`
create name by data type	`typedef` *typename*
size of an object (type is `size_t`)	`sizeof` *object*
size of a data type (type is `size_t`)	`sizeof(`*type name*`)`

Initialization

initialize variable	*type name=value*
initialize array	*type name*`[]={`*value*$_1$,...`}`
initialize char string	`char` *name*`[]="`*string*`"`

Constants

long (suffix)	`L` or `l`
float (suffix)	`F` or `f`
exponential form	`e`
octal (prefix zero)	`0`
hexadecimal (prefix zero-ex)	`0x` or `0X`
character constant (char, octal, hex)	`'a', '\ooo', '\xhh'`
newline, cr, tab, backspace	`\n, \r, \t, \b`
special characters	`\\, \?, \', \"`
string constant (ends with `'\0'`)	`"abc...de"`

Pointers, Arrays & Structures

declare pointer to *type*	*type* `*`*name*
declare function returning pointer to *type*	*type* `*f()`
declare pointer to function returning *type*	*type* `(*pf)()`
generic pointer type	`void *`
null pointer	`NULL`
object pointed to by *pointer*	`*`*pointer*
address of object *name*	`&`*name*
array	*name*`[`*dim*`]`
multi-dim array	*name*`[`*dim*$_1$`][`*dim*$_2$`]...`

Structures

`struct` *tag* `{`	structure template
declarations	declaration of members
`};`	
create structure	`struct` *tag name*
member of structure from template	*name*`.`*member*
member of pointed to structure	*pointer* `->` *member*
Example. `(*p).x` and `p->x` are the same	
single value, multiple type structure	`union`
bit field with *b* bits	*member* `:` *b*

Operators (grouped by precedence)

structure member operator structure pointer	*name*`.`*member* *pointer*`->`*member*
increment, decrement	`++, --`
plus, minus, logical not, bitwise not	`+, -, !, ~`
indirection via pointer, address of object	`*`*pointer*`, &`*name*
cast expression to type	`(`*type*`)` *expr*
size of an object	`sizeof`
multiply, divide, modulus (remainder)	`*, /, %`
add, subtract	`+, -`
left, right shift [bit ops]	`<<, >>`
comparisons	`>, >=, <, <=`
comparisons	`==, !=`
bitwise and	`&`
bitwise exclusive or	`^`
bitwise or (incl)	`|`
logical and	`&&`
logical or	`||`
conditional expression	$expr_1$ `?` $expr_2$ `:` $expr_3$
assignment operators	`+=, -=, *=, ...`
expression evaluation separator	`,`

Unary operators, conditional expression and assignment operators group right to left; all others group left to right.

Flow of Control

statement terminator	`;`
block delimeters	`{ }`
exit from `switch`, `while`, `do`, `for`	`break`
next iteration of `while`, `do`, `for`	`continue`
go to	`goto` *label*
label	*label*`:`
return value from function	`return` *expr*

Flow Constructions

`if` statement	`if (`*expr*`)` *statement* `else if (`*expr*`)` *statement* `else` *statement*
`while` statement	`while (`*expr*`)` *statement*
`for` statement	`for (`$expr_1$`;` $expr_2$`;` $expr_3$`)` *statement*
`do` statement	`do` *statement* `while(`*expr*`);`

`switch` statement	`switch (`*expr*`) {`
	`case` $const_1$`:` $statement_1$ `break;`
	`case` $const_2$`:` $statement_2$ `break;`
	`default:` *statement*
	`}`

ANSI Standard Libraries

`<assert.h>`	`<ctype.h>`	`<errno.h>`	`<float.h>`	`<limits.h>`
`<locale.h>`	`<math.h>`	`<setjmp.h>`	`<signal.h>`	`<stdarg.h>`
`<stddef.h>`	`<stdio.h>`	`<stdlib.h>`	`<string.h>`	`<time.h>`

Character Class Tests `<ctype.h>`

alphanumeric?	`isalnum(c)`
alphabetic?	`isalpha(c)`
control character?	`iscntrl(c)`
decimal digit?	`isdigit(c)`
printing character (not incl space)?	`isgraph(c)`
lower case letter?	`islower(c)`
printing character (incl space)?	`isprint(c)`
printing char except space, letter, digit?	`ispunct(c)`
space, formfeed, newline, cr, tab, vtab?	`isspace(c)`
upper case letter?	`isupper(c)`
hexadecimal digit?	`isxdigit(c)`
convert to lower case?	`tolower(c)`
convert to upper case?	`toupper(c)`

String Operations `<string.h>`

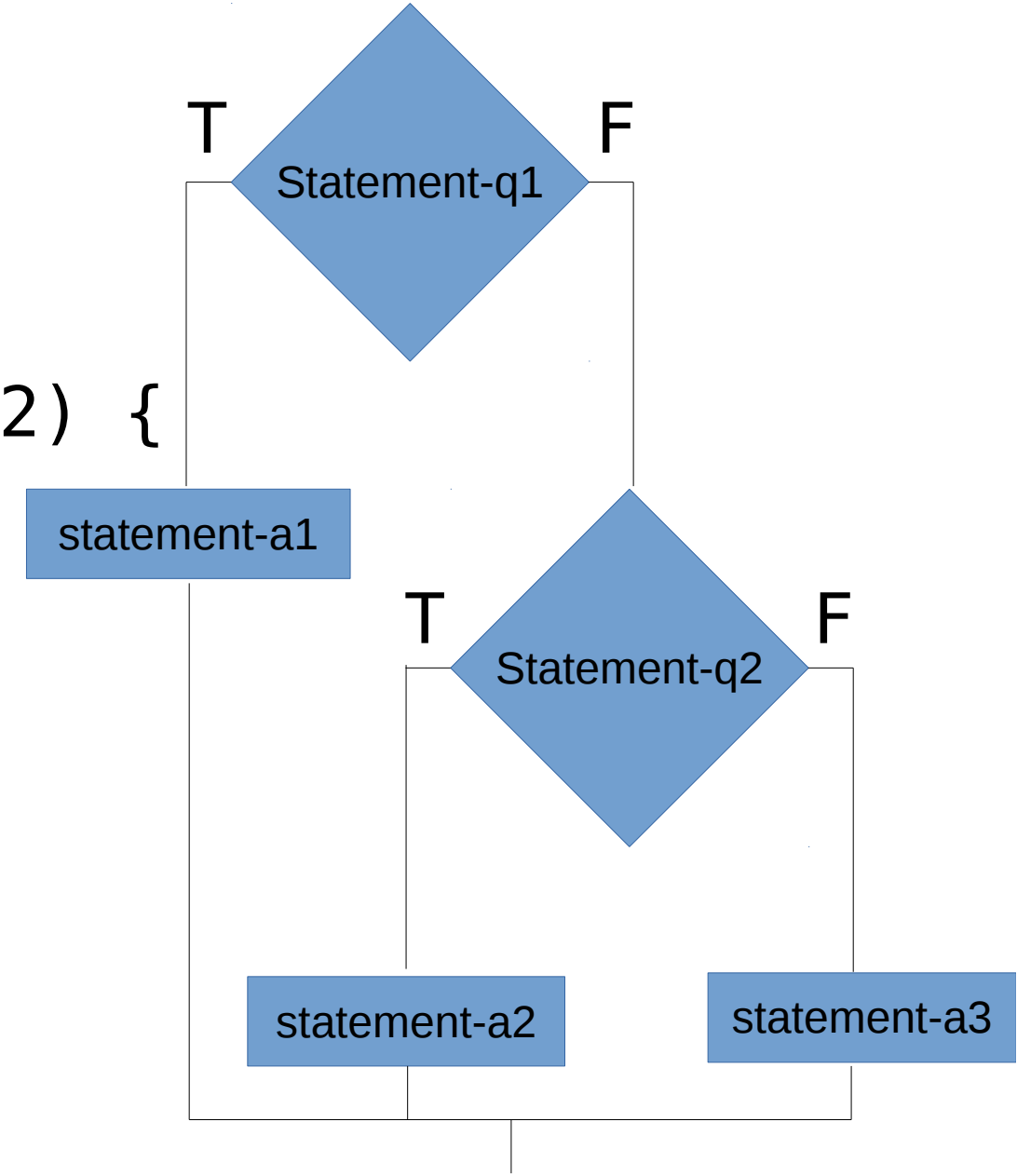
`s,t` are strings, `cs,ct` are constant strings

length of `s`	`strlen(s)`
copy `ct` to `s`	`strcpy(s,ct)`
up to `n` chars	`strncpy(s,ct,n)`
concatenate `ct` after `s`	`strcat(s,ct)`
up to `n` chars	`strncat(s,ct,n)`
compare `cs` to `ct`	`strcmp(cs,ct)`
only first `n` chars	`strncmp(cs,ct,n)`
pointer to first `c` in `cs`	`strchr(cs,c)`
pointer to last `c` in `cs`	`strrchr(cs,c)`
copy `n` chars from `ct` to `s`	`memcpy(s,ct,n)`
copy `n` chars from `ct` to `s` (may overlap)	`memmove(s,ct,n)`
compare `n` chars of `cs` with `ct`	`memcmp(cs,ct,n)`
pointer to first `c` in first `n` chars of `cs`	`memchr(cs,c,n)`
put `c` into first `n` chars of `cs`	`memset(s,c,n)`


I COULD RESTRUCTURE THE PROGRAM'S FLOW
OR USE ONE LITTLE 'GOTO' INSTEAD.
EH, SCREW GOOD PRACTICE. HOW BAD CAN IT BE?
goto main_sub3;
COMPILE

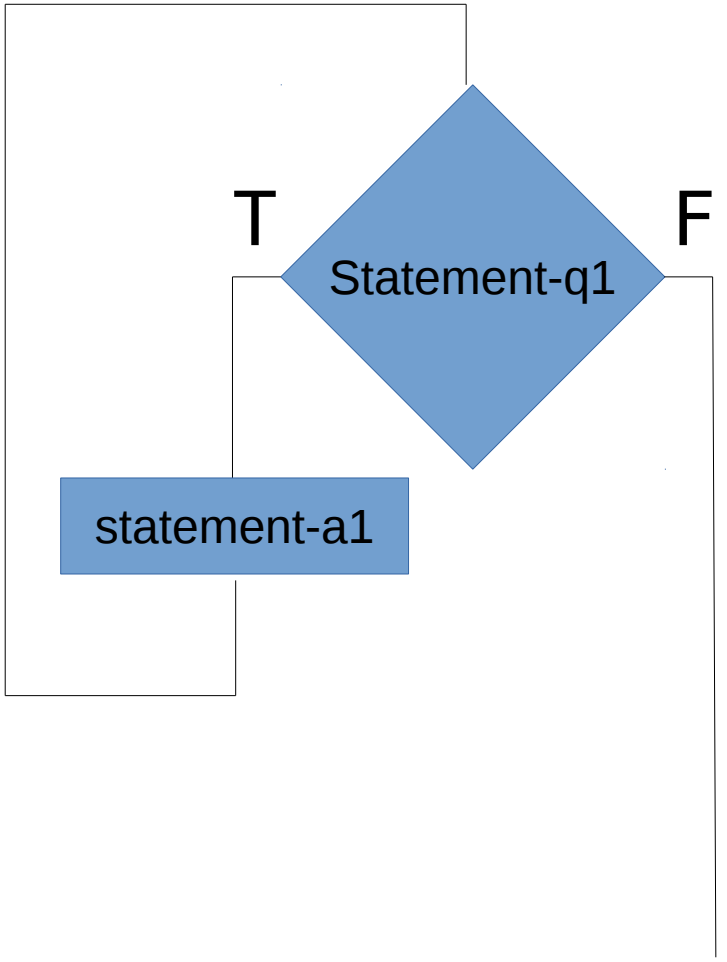
Flow of Control: If

```
if (statement-q1) {
   statement-a1;
} else if (statement-q2) {
   statement-a2;
} else {
   statement-a3;
}
```



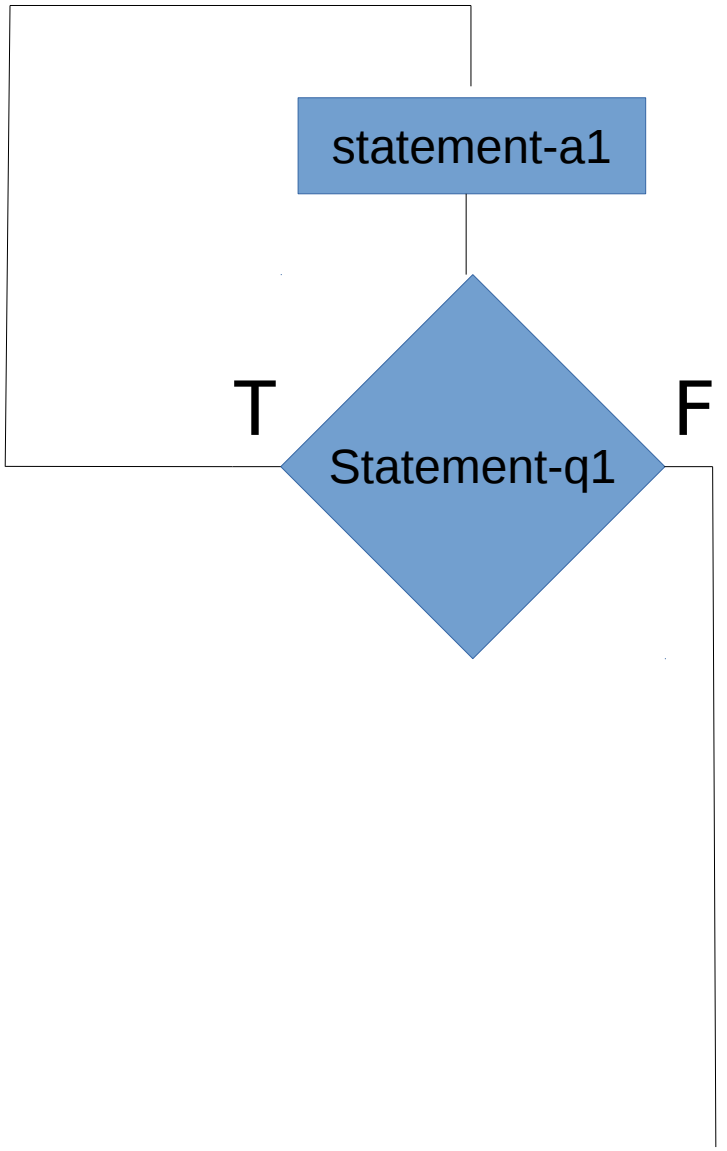
Flow of Control: While

```
while (statement-q1) {
   statement-a1;
}
```



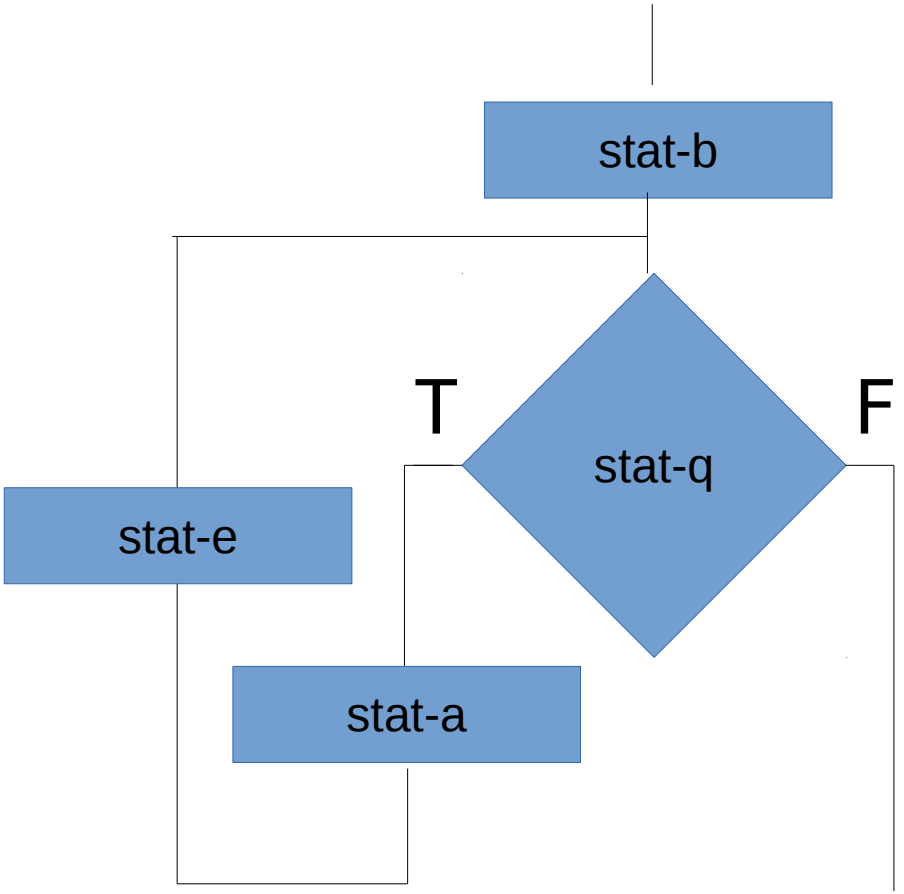
Flow of Control: Do While

```
do  {
   statement-a1;
} while (statement-q1);
```



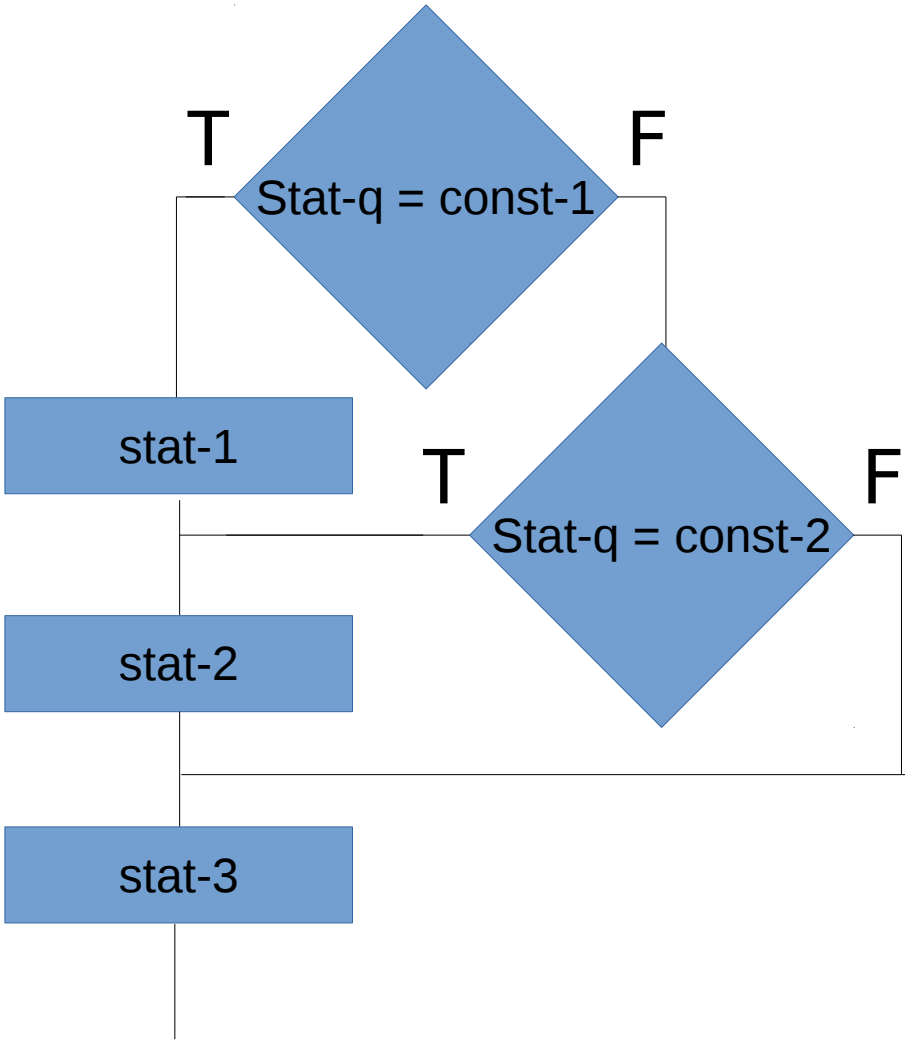
Flow of Control: For

```
for (stat-b; stat-q; stat-e) {
   stat-a;
}
```



Flow of Control: Switch

```
switch (stat-q) {
   case const-1 : {
      stat-1;
   } case const-2 : {
      stat-2;
   } default : {
      Stat-3;
   }
}
```



Flow of Control: Continue & Break

continue;
Finishes this **one** loop iteration,
ignores everything else.

Break;
Ends the loop,
ignores everything else.

Flow of Control: Continue & Break

continue;
Finishes this **one** loop iteration,
ignores everything else.

Break;
Ends the loop,
ignores everything else.

Arrays

```
int ary[size];

int ary[size] = {1, 2, …, size};

int ary[size] = {5};

int ary[xsize][ysize][zsize];

int ary[xsize][ysize][zsize] =
  {{1, 2, 3, 4},
  {5, 6, 7, 8},
  {9, 10, 11, 12}};
```

Your Mistakes

49% Syntax errors – 162 sec

31% Semantic errors – 438 sec

20% Type errors – 86 sec

[1]

[1] https://kar.kent.ac.uk/46742/1/fp1187-altadmri.pdf