Programski jezik C

Pisanje kompleksnih programov

Pisanje obsežnih programov

- Zaglavne datoteke (header files)
- Zunanje spremenljivke in funkcije
- Prednosti uporabe več datotek
- Kako razbijemo program na več datotek
- Organizacija podatkov v vsaki datoteki
- Orodje make
- Napravimo datoteko makefile

Pisanje bolj obsežnih programov terja delitev programov na module oziroma ločene datoteke. Zato bo funkcija main( ) v eni datoteki, druge funkcije pa verjetno v kateri drugi.

Tvorimo lahko knjižnico funkcij, ki so lahko skupinsko pomnjene v takih modulih. To omogoča uporabo pripravljenih modulov v različnih programih, v katere jih dodamo v času samega prevajanja.

Kaj pa grafične aplikacije

Sam jezik nima grafičnega vmesnika (API)

C so razvili precej, preden so se pojavili različni grafični vmesniki oziroma grafično podprti programi. Tako nimamo enotnega grafičnega programskega vmesnika (API, Application Programming Interface), kot je to sicer značilno za Javo).

Obstajajo grafični programski vmesniki za operacijske sisteme UNIX/LINUX in za MS Windows, ki pa se med seboj zelo razlikujejo.

Primer prosto dostopnega grafičnega orodja je **GTK+**. Ta omogoča pisanje grafičnih uporabniških vmesnikov za različne platforme (Linux, MS Windows) in različne programske jezike, kot so C, C++, Python in C#.

Primer: GTK in Hello World (1)


```
#include <gtk/gtk.h>

/* This is a callback function. T*/
static void hello( GtkWidget *widget, gpointer   data ) {
    g_print ("Hello World\n");
}

static gboolean delete_event( GtkWidget *widget, GdkEvent  *event, gpointer   data ){
    g_print ("delete event occurred\n");
    return TRUE;
}
/* Another callback */
static void destroy( GtkWidget *widget,  gpointer   data ) {
    gtk_main_quit ();
}

int main( int   argc,  char *argv[] ) {
    /* GtkWidget is the storage type for widgets */
    GtkWidget *window;
    GtkWidget *button;
```

Primer: GTK in Hello World (2)

```
 /* This is called in all GTK applications. Arguments are parsed  from the command line and are returned
to the application. */
    gtk_init (&argc, &argv);
    /* create a new window */
    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);

    g_signal_connect (G_OBJECT (window), "delete_event", G_CALLBACK (delete_event), NULL);

    g_signal_connect (G_OBJECT (window), "destroy", G_CALLBACK (destroy), NULL);
    /* Sets the border width of the window. */
    gtk_container_set_border_width (GTK_CONTAINER (window), 10);
    /* Creates a new button with the label "Hello World". */
    button = gtk_button_new_with_label ("Hello World");
    g_signal_connect (G_OBJECT (button), "clicked",  G_CALLBACK (hello), NULL);
    g_signal_connect_swapped (G_OBJECT (button), "clicked",
                                                G_CALLBACK (gtk_widget_destroy), G_OBJECT (window));
    /* This packs the button into the window (a gtk container). */
    gtk_container_add (GTK_CONTAINER (window), button);
    /* The final step is to display this newly created widget. */
    gtk_widget_show (button);
    /* and the window */
    gtk_widget_show (window);
    /* All GTK applications must have a gtk_main(). Control ends here and waits for an event to occur  */
    gtk_main ();
    return 0;
}
```

Prednosti uporabe več datotek

Program lahko razvija več programerjev. Vsak dela svojo datoteko. Uporabimo lahko objektno usmerjen pristop. Posamezne datoteke lahko predstavljajo posamezne objekte in vsebujejo ustrezne podatke in operacije nad njimi. Vzdrževanje tako strukturiranih programov je lažje.

Datoteke lahko vsebujejo funkcije določene skupine, na primer vse matrične operacije. Imamo tako lahko knjižnice funkcij.

Dobro zasnovane funkcije lahko uporabimo v drugih programih in tako skrajšamo čas razvoja.

Pri zelo obsežnih programih morda celotno datoteko zaseda ena pomembna funkcija, ob njej pa je še več nizkonivojskih funkcij.

Če spremenimo neko datoteko, je potrebno za obnovitev izvedljivega programa ponovno prevajanje le te datoteke. Na sistemih Linux si pri tem lahko pomagamo z orodjem make.

Programi z več izvornimi datotekami

C Programi – 2 tipa datotek

- .c datoteke :
  - Vsebujejo izvorno kodo in definicije globalnih spremenljivk
  - datoteke niso nikoli vključene v druge c datoteke
- .h datoteke:
  - Vsebujejo deklaracije funkcij, definicije struktur, definicije konstant

Zaglavne datoteke (header files)

Pri uvedbi takega, modularnega programiranja želimo obdržati definicije spremenljivk, prototipe funkcij ipd. znotraj posameznega modula. Kaj pa,če take definicije souporablja več modulov? Bolje je, če centraliziramo take definicije v eni datoteki in souporabljamo to datoteko v drugih modulih. Taki datoteki pravimo zaglavna datoteka (header file) in ima podaljšek .h

Standardne zaglavne datoteke upoštevamo v našem programu na naslednji znan način:

```
#include <stdio.h>
```

Lahko pa napišemo tudi lastne zaglavne datoteke, ki jih v našem programu upoštevamo tako:

```
#include "mojaDatoteka.h"
```

Primer: napravimo svojo “knjižnico”

```
/* datoteka: util.c */
#include "util.h"
int rand_seed=10;

/***** tvorba nakljucnega stevila med 0 in 32767.  ****/
int rand() {
    rand_seed = rand_seed * 1103515245 +12345;
    return (unsigned int)(rand_seed / 65536) % 32768;
}
/*****    Sortiranje polja celih stevil                    ****/
void bubbleSort(int m,int a[ ]) {
    int x,y,temp;
     for (x=0; x < m-1; x++)
        for (y=0; y < m-x-1; y++)
            if (a[y] > a[y+1])  {
                temp=a[y];
                a[y]=a[y+1];
                a[y+1]=temp;
            }
}
```

Uporaba knjižnice util

```
#include <stdio.h>
#include "util.h"

#define MAX 10
int a[MAX];

void main() {
    int i,t,x,y;
    /* napolnimo polje */
    for (i=0; i < MAX; i++) {
        a[i]=rand();
        printf("%d\n",a[i]);
    }
    bubbleSort(MAX,a);

    /* Izpis urejenega polja */
    printf("--------------------\n");
    for (i=0; i < MAX; i++)  printf("%d\n",a[i]);
}
```

WEB

Demo

Potrebno je narediti nov projekt z vsemi uporabljenimi datotekami

Zaglavna datoteka util.h

```
/* datoteka: util.h */
extern int rand();
extern void bubbleSort(int, int [ ]);
```

Extern pove programu C, da bosta ti dve rutini definirani v drugi datoteki

Kako vse skupaj prevedemo?

Privedemo knjižnico:

gcc -c -g util.c

-c povzoči, da bo prevajalnik tvoril objektno datoteko za knjižnico z imenom **util.o**.

Prevod glavnega programa:

gcc -c -g main.c

Dobimo datoteko z imenom **main.o .** To še ni izvedljiva datoteka, nanjo moramo navezati knjižnico:

gcc -o main main.o util.o

Kako razdelimo program na več datotek

Programerji običajno začnejo snovanje programa tako, da razbijejo problem na več sekcij. Vsako od teh sekcij lahko realiziramo z eno ali več funkcijami. Vse funkcije iste sekcije pomnimo v isti datoteki.

Podobno velja, če implementiramo objekte kot podatkovne strukture. Vse funkcije, ki delajo s tako strukturo, pomnimo v isti datoteki. Kasnejša sprememba "objekta" terja spremembo le v njemu ustrezni datoteki.

Najbolje je,če za vsako od teh izvornih datotek (s podaljškom .c) napišemo še ustrezno zaglavno datoteko (z enakim imenom, toda s podaljškom .h). Zaglavna datoteka vsebuje definicije vseh funkcij, ki jih "njena" izvorna datoteka uporablja.

Druge datoteke,ki uporabljajo te funkcije morajo na začetku imeti ustrezen stavek #include z navedbo datoteke .h

Organizacija podatkov v datotekah

Tipičen vrstni red je naslednji:

- Definicija vseh konstant s stavki #define,
- Definicija vseh potrebnih zaglavnih datotek s stavki #include
- Definicija vseh važnejših podatkovnih tipov s stavki typedef
- Deklaracija globalnih in eksternih spremenljivk. Globalne spremenljivke lahko sočasno tudi inicializiramo.
- Ena ali več funkcij.

Orodje make

Kaj je make?

- Program za vodenje ponovnega prevajanja.
- Avtomatsko ugotovi, kateri del večjega programa mora biti spet preveden.
- Sproži ukaze za ponoven prevod takih datotek.
- Uporabimo ga lahko z različnimi programskimi jeziki.

Kako uporabimo make ?

- Napišemo tekstovno datoteko z imenom "**Makefile**“. Ta opisuje relacije med različnimi datotekami programa in nudi ukaze za njihovo posodobitev.

- V terminalskem oknu (konzoli) vpišemo ukaz

make

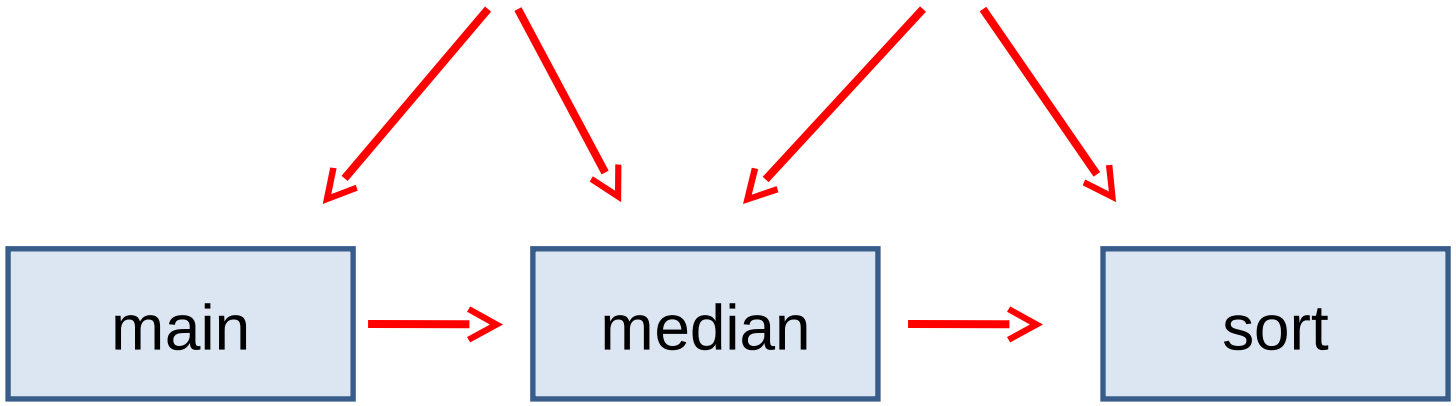
In s tem sprožimo vse potrebne ponovne prevode in tvorbo posodobljenega programa.

Primer strukture programskega projekta

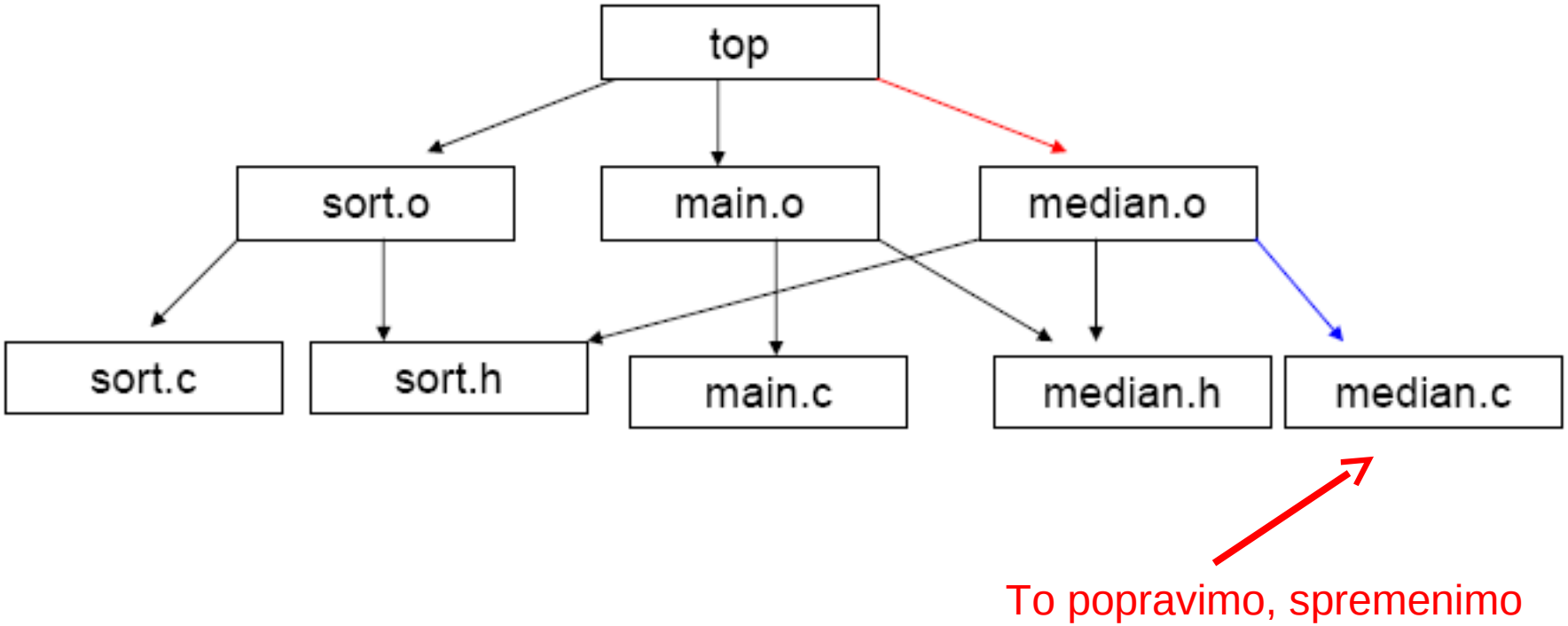
Strukturo projekta in odvisnosti datotek lahko predstavimo z grafom

Primer programskega projekta s 5 datotekami:

main.c, median.h, median.c, sort.h., sort.c



Graf odvisnosti



Če spremenimo median.c, moramo še narediti:

- gcc –c median.c
- gcc median.o main.o sort.o –o top

Sintaksa makefile

Odvisnosti

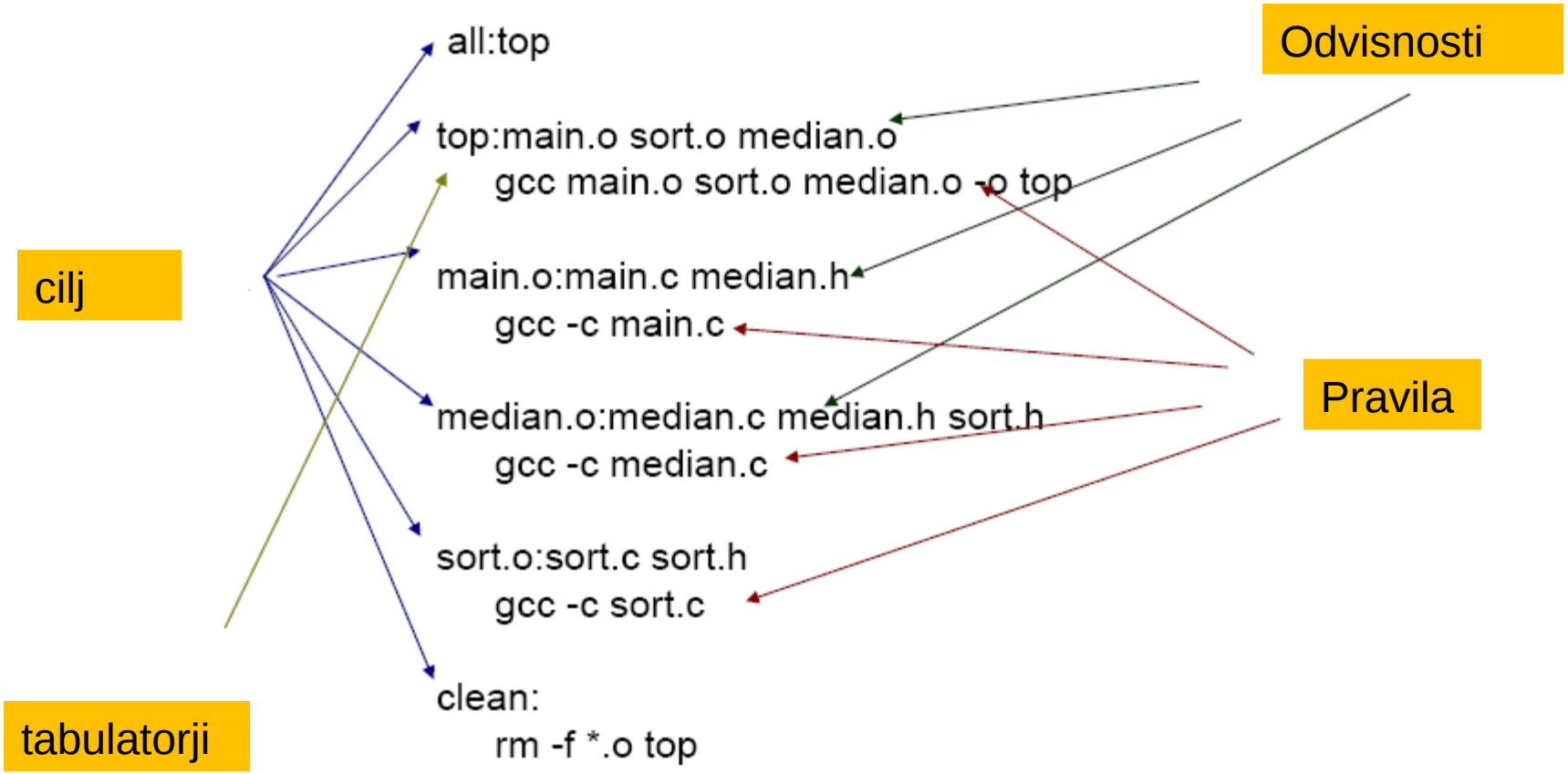
main.o: main.c median.h

gcc –c main.c

Pravilo

tabulator

Vsebina makefile



Izboljšani makefile

```
CC=gcc
CFLAGS=-c -Wall
all:top
top:main.o sort.o median.o
	$(CC) main.o sort.o median.o -o top
main.o:main.c
	$(CC) $(CFLAGS) main.c
sort.o:sort.h sort.c
	$(CC) $(CFLAGS) sort.c
median.o:median.h median.c
	$(CC) $(CFLAGS) median.c
clean:
	rm -f *.o top
```

Več o tem

Še nekaj napotkov….

- Vedno vse inicializirajte pred uporabo (posebno kazalce)
- Ne uporabljajte kazalcev, ko jih sprostite
- Ne vračajmo lokalnih spremenljivk neke funkcije po referenci
- Ni izjem– torej sami preverjajte možnost napak
- Polje je vedno kazalec, katerega vrednosti pa ne moremo spreminjati (je naslov).

- Pa še kaj bi se našlo.

C v primerjavi z Javo

- Java je objektno usmerjena, C je funkcijsko usmerjen
- Java se strogo drži podatkovnih tipov, C is je pri tem površen
- Java podpira polimorfizem (na primer. + ==), C nima polimorfizma
- Javanski razredi (classes) pomagajo pri nadzoru imenskega prostora (name space), C ima en sam imenski prostor
- Spremenljivke C lahko definiramo le na začetku bloka
- Programe C prevajalnik pred-procesira, za Javo to ne velja
- Java ima kompleksen večplastni vhodno-izhodni model, C ima le preprost vhodni oziroma izhodni tok bajtov
- Java ima avtomatsko upravljanje s pomnilnikom, pri C moramo za to sami skrbeti
- Java nima eksplicitnih kazalcev, C uporablja kazalce pogosto
- Java ima parametre po referenci in po vrednosti , C jih ima le po vrednosti
- Java ima izjeme in rokovanje z izjemami, C pozna signale (Pri LINUX)
- Java ima primitive za konkurenčnost, C izvaja konkurenčnost preko funkcijskih knjižnic
- Polja v C nimajo atributa "length"
- Nizi v Javi (Strings) so dobro definirani objekti, nizi v C so polja char[]

Kako naj ne programiramo v C

```
#include <stdio.h>
main(t,_,a)char *a;{return!0<t?t<3?main(-79,-13,a+main(-87,1-_,
main(-86,0,a+1)+a)):1,t<_?main(t+1,_,a):3,main(-94,-27+t,a)&&t==2?_<13?
main(2,_+1,"%s %d %d\n"):9:16:t<0?t<-72?main(_,t,
"@n'+,#'/*{}w+/w#cdnr/+,{}r/*de}+,/*{*+,/w{%+,/w#q#n+,/#{l,+,/n{n+,/+#n+,/#\
;#q#n+,/+k#;*+,/'r :'d*'3,}{w+K w'K:'+}e#';dq#'l \
q#'+d'K#!/+k#;q#'r}eKK#}w'r}eKK{nl]'/#;#q#n'){)#}w'){){nl]'/+#n';d}rw' i;# \
){nl]!/n{n#'; r{#w'r nc{nl]'/#{l,+'K {rw' iK{;[{nl]'/w#q#n'wk nw' \
iwk{KK{nl]!/w{%'l##w#' i; :{nl]'/*{q#'ld;r'}{nlwb!/*de}'c \
;;{nl'-{}rw]'/+,}##'*}#nc,',#nw]'/+kd'+e}+;#'rdq#w! nr'/ ') }+}{rl#'{n' ')# \
}'+}##(!!/")
:t<-50?_==*a?putchar(31[a]):main(-65,_,a+1):main((*a=='/')+t,_,a+1)
  :0<t?main(2,2,"%s"):*a=='/'||main(0,main(-61,*a,
"!ek;dc i@bK'(q)-[w]*%n+r3#l,{}:\nuwloca-O;m .vpbks,fxntdCeghiry"),a+1);}
```

DEMO

In rezultat tega

On the first day of Christmas my true love gave to me
a partridge in a pear tree.

On the second day of Christmas my true love gave to me
two turtle doves
and a partridge in a pear tree.

On the third day of Christmas my true love gave to me
three french hens, two turtle doves
and a partridge in a pear tree.

On the fourth day of Christmas my true love gave to me
four calling birds, three french hens, two turtle doves
and a partridge in a pear tree.

On the fifth day of Christmas my true love gave to me
five gold rings;
four calling birds, three french hens, two turtle doves
and a partridge in a pear tree.

.

..... Se nadaljuje do dvanajstega dne