Школа рачунара и математике

www.rajak.rs
skolarajak@gmail.com programerska.radionica@gmail.com
www.facebook.com/Skola.Rajak.

Програмерска радионица

Програмски језик C

Додатни материјал

Аутори:

Ким Новак, Себастиан Новак

Предговор

Идеја овог писаног материјала је да прошири оно што је речено на предавањима, никада неће излазити ван оквира области за које је предвиђен, односно читање и усвајање и овог градива није неопходно за праћење наставе програмерске радионице.

Такође, на предавањима се неће тражити ништа што није на њима већ речено, како нико не би био оштећен.

Насупрот томе, при излагању градива у овим материјалима, подразумеваће се да је градиво са предавања усвојено и понављање градива са предавања ће бити минимално.

За оне који су заинтересовани и желе да науче више, овај материјал ће служити као помоћ при савладавању сложенијих проблема.

Све примедбе, критике, утиске, сугестије као и уочене грешке (лексичке или везане за градиво) можете послати мејлом на programerska.radionica@gmail.com .

Унапред захвални за Ваш труд:

Аутори

Садржај

1.0 Програмски језик C

1.1 Елементи језика *C*

Програмски језик C користи знакове ASCII табеле, тј. Сва слова велика и мала(Енглеског алфабета), бројеве, беле знакове и специјалне знаке. *C* јесте *case sensitive* односно разликује велика и мала слова (ово је битно јер ако смо нешто назвали A, уколико негде напишемо мало a неће препознати да је то исто). Језик C садржи идентификаторе, константе, операторе, сепараторе и службене речи. Све укупно називамо их Лексичким симболима језика C.

1.2 Наредбе

Наредба је саставни део кода и увек се завршава терминатором **;**
Низ наредби чини програм. Једна наредба може да заузима више редова кода, а и више наредби могу да се нађу у једној линији кода.

1.3 Коментари

Коментари не утичу на програм и не сматрају се делом програма. Употребом коментара описујемо како наш програм ради, наглашавамо важне делове, секције. Такође дајемо информације почетне о нама, ауторима програма, сврси програма (како би онај ко буде можда користио наш код могао лакше и брже да га разуме), итд.

У C-у постоје 2 врсте коментара:
Једнолинијски `int x; // најважнија променљива`
Вишелинијски

```
/*
    Назив програма
    Аутор: Себастиан Новак
    Контакт:            www.rajak.rs programerska.radionica@gmail.com
    Верзија: 1.0.0
*/
```

1.4 Променљива

Представљају именовани простор у меморији рачунара, пре употребе се морају декларисати. Представљају начин за чување података које наш програм користи. Имају типове, подаци одређене врсте иду у променљиве одређеног типа.

Декларишемо променљиве наводећи им тип и идентификатор (име).
Идентификатор представља назив који ћемо доделити некој променљивој, потпрограму и сл. Назив мора започети **словом** а никад бројем или специјалним знаком. Постоје резервисане речи(службене речи) које се не смеју користити као идентификатори. Те речи су:
auto, break, case, char, const, continue, default, do, double, else, enum, extern, float, for, goto, if, int, long, register, return, short, signed, sizeof, static, struct, switch, typedef, union, unsigned, void, volatile, while

1.5 Типови података

За разумевање типова података послушати предавање!

Неки од типова: int, short int, float, double, long double.
int – *integer* представља целобројне вредности, тј. Позитивне и негативне целе бројеве. (На пример: 1,2,3,4,-1,-2..).
float- представља реалне бројеве, односно све целобројне и децималне бројеве (2.0, 3.14, 2,72, 3.8, -5.3...).
double и *long double* представљају исто што и *float* само могу да буду много веће вредности, или много прецизније децимале. Заузимају више меморије и самим тим могу да садрже више цифри него *float.*
char- представља све знакове ASCII табеле. Углавном се користи за текст, али се може користити и за мале целобројне вредности (бројеве од 0 до 255).

1.5.1 Дефинисање података

Подаци могу бити променљиви и непроменљиви.
Променљиви подаци односно **променљиве** дефинишемо тако што наводимо тип променљиве и назив променљиве. (пример: int promenljiva1;)

1.6.0 Први програм

Програм мора имати заглавље, почетак, где ћемо убацити библиотеке које су нам потребне и декларисати променљиве. Библиотека у Ц-у садржи функције које користимо у програму. За сваку функцију потребно је додати библиотеку у супротном програм неће функционисати. Библиотеку укључујемо одмах на почетку програма, па би прва ставка кода била:

#include <stdio.h>

Тиме смо укључили функције(наредбе) као што су printf(исписивање текста на стандардни излаз),scanf(учитавање података са стандардног улаза) itd. Стандардни улаз је тастатура, а стандардни излаз је екран.

Даље, главни део кода(програма) почиње са:

Int main(){

Знак **{** користимо за почетак неког блока, сваки блок који је отворен знаком { потребно је на крају затворити знаком }

Int main - Ову функцију зовемо мејн, и то је главни део програма. У мејну ћемо позивати остале функције.

Наш програм ће да исписује кориснику на стандардни излаз „Здраво свете!“

То ћемо урадити користећи наредбу за испис printf:

printf("Zdravo svete! ");

Подсетник: Иза сваке наредбе мора се наћи терминатор **;**
Сваки текст који желимо да испишемо мора се наћи између наводника. Стринг ће бити објашњен у неком од следећих предавања.

За крај потребно је да програм враћа вредност 0 што означава да је успешно извршен програм.

На крају сваког програма ћемо писати return 0;

Пошто смо отворили { потребно је на крају и затворити }. Може се тумачити као *begin* и *end*.

Све заједно код изгледа овако:

```
#include <stdio.h>

 int main(){

     printf("Zdravo svete");

 return 0;
}
```

Ради прегледности програма све наредбе у мејну ћемо увлачити користећи дугме Tab на тастатури (као што је то урађено за printf).

1.6.1 Покретање, чување и компајлирање програма (За *Codeblocks* и *Windows*)

Када смо написали код, програм покрећемо притиском тастера F9 или одете на *Build->Build and run*. Уколико је програм тачно написан појавиће се црни прозор и то је покренут програм. За затварање тог прозора довољно је да притиснете било који тастер. Ако код садржи грешку, компајлер ће је написати и биће видљива у прозору испод кода који се зове Logs & others картица Build log. Ако Вам пише `Process terminated with status 0 (0 minutes, 6 seconds)`
То не значи да нешто не ваља већ само исписује колико је трајало извршавање програма. Сваку промену потребно је сачувати и то радимо кликом на дискету.

1.6.2 Чување програма (За Linux и Gedit)

Када сте завршили са писањем кода потребно га је сачувати. То се ради путем File->Save ili ctrl+s, затим написати име програма и **обавезно ставити .c иза назива!** На пример: program.c

Немојте изаћи из терминала јер је врло вероватно да ће компајлер пријавити грешку коју треба исправити па да не би стално излазили и улазили поново у Гедит.

1.6.3 Компајлирање

Компајлер пролази кроз наш код и уколико је нешто погрешно написано пријављује грешку.

Отворићемо Терминал. У њему прво ћемо написати наредбу ls која ће нам приказати садржај директоријума user's home. Уколико сте програм сачували баш у home фолдеру назив програма ће се наћи у том садржају. Ако је програм сачуван унутар неког под директоријума њему приступамо уз помоћ наредбе cd nazivdirektorijuma.

Када смо пронашли директоријум у којем се налази наш програм, компајлирамо га помоћу наредбе:

Gcc –o program program.c

Program = назив програма који ћемо покренути (компајлиран)

Program.c = назив датотеке у којој се налази наш код

Уколико компајлер није пријавио никакву грешку, можемо покренути програм.

То чинимо у терминалу наредбом ./program

Резултат овога требало би да буде испис на екрану Здраво свете!

1.7 Улаз и излаз података (читање и писање података)

Под појмом улазни подаци подразумевамо податке унете путем тастатуре. То може бити неко слово, број, знак, реченица итд. Пошто рачунар ради само са бинарним обликом, потребно је извршити конверзију тих података које смо унели у њему разумљив језик. То радимо користећи улазну конверзију. На пример, питали смо корисника да унесе неки знак са тастатуре и пошто ми унапред не знамо да ли ће то бити број или слово (тако ни рачунар не зна шта ми желимо) потребно је написати знак за конверзију. Узећемо пример да смо хтели да корисник унесе број. Наредба би гласила овако:
scanf("%d", &znak);

Општи облик функције scanf гласи:
scanf("tip/format konverzije", &podatak);

Podatak-представља назив променљиве који је произвољан

Типови конверзија:

%d – int
%f – float

%c –char

%i – decimalni hexadecimalni ili okatalni broj

%x – hexadecimalni (који год знак да се унесе биће претворен у хексадецимални запис)

%o –oktalna konverzija

%li – long int

%s – string (низ знакова – нека реч)

Број конверзија и улазних података мора бити исти.

На пример ако тражимо од корисника да унесе три броја, њих ћемо очитати помоћу:
scanf ("%d%d%d", &broj1, &broj2, &broj3);

Излазни подаци представљају крајњи производ програма који ћемо пружити кориснику. То може бити испис података на стандардни излаз (екран) или смештање податка у неку датотеку. За испис на стандардни излаз користимо printf. Као и код улазних података и код излазних је потребна конверзија. Знак A је могуће представити баш као знак (A), као број (његово место у ASCII табели 65) итд. Да би исписана била баш она вредност коју желимо потребно је навести конверзију за тај податак.

```
printf("%d", podatak);
```

Општи облик наредбе printf је:

```
printf ("tip/format promenljive", promenljiva);
```

или само за испис неког текста:

```
printf ("tekst");
```

или комбиновано:

printf ("tekst tip/format promenljive teskt", promenljiva);

Примери за улазне и излазне податке:

1.

printf("unesite neki karakter sa tastature\n");
scanf("%c", &znak);
printf("%c", znak);//ispisivanje znaka kao znak, ako je uneto A ispisace A

2.

printf("unesite neki karakter sa tastature\n");
scanf("%c", &znak);
printf("%d", znak); //ispisivanje znaka kao malog celog broja, ako je uneto A ispisace 65

3.

printf("unesite koordinate tacke A I B\n");
scanf("%d%d%d%d", &x1, &y1, &x2, &y2);
printf("tacka A ima koordinate (%d, %d), a tacka B (%d, %d)\n", x1, y1, x2, y2);

1.7.1 Пример C програма који чита две променљиве и касније исписује њихове вредности:

```c
#include <stdio.h>

int main()
{
    int x;
    float y;
    printf("Unesite neki celi broj i pritisnite enter.\n");
    scanf("%d", &x);
    printf("Uneli ste %d\n", x);
    printf("Unesite neki realni broj i pritisnite enter.\n");
    scanf("%f", &y);
    printf("Uneli ste %f\n", y);
    return 0;
}
```

2.0 ОПЕРАТОРИ

Зову се операторима, јер врше неке операције. Они над којима се врше операције називају се операнди. У зависности над колико се операнада неки оператор примењује, оператори се деле на бинарне (два операнда) и унарне (само један операнд).
У зависности од намене оператори се такође деле на аритметичке, релационе и логичке.
Осим ових, имамо оператор доделе вредности.

2.1 Аритметички оператори

Бинарни аритметички оператори (односно оператори који узимају два операнда) јесу:

1. Оператор + , који сабира левог операнда са десним. Примери употребе оператора:
   x+x , y+x, x+3, 3+y, 3+3, 1+1
2. Оператор - , који одузима вредност левог операнда од вредности десног операнда. Примери употребе оператора:
   x-x , y-x, x-3, 3-y, 3-3, 1-1
3. Оператор * , који множи левог операнда са десним. Примери употребе оператора:
   x*x , y*x, x*3, 3*y, 3*3, 1*1
4. Оператор / , који множи левог операнда са десним. Примери употребе оператора:
   x/x , y/x, x/3, 3/y, 3/3, 1/1
   Овде је важно навести да се оператор / понаша на 2 различита начина, у зависности од типа операнада. Први начин је целобројно дељење, дешава се када су оба операнда цели бројеви или променљиве целобројног типа. Пример:

```
int x;
int y;
int s;
float z;
float u;
float r;
x=5;
y=3;
s=x/y;                          //      5/3=1  зареза нема, целобројно дељење

z=3.3;
u=1.0;
r=z/u;                          //   3.3 / 1.0 = 3.3 реална вредност, реално дељење
```

5. Оператор %, даје остатак при целобројном дељењу десног операнда са левим операндом. Пример:
   4%3 даје резултат 1 (3 у 4 једном, остатак до 4 је 1) , 4%2 даје резултат 0 (2 у 4 , два пута,

остатак до 4 је 0)
11%3 даје резултат 2 (3 у 11 три пута, остатак до 11 је 2)

Постоје и скраћене форме оператора +, -, *, / које се користе када се променљива са леве стране на неки једноставан начин модификује.
x+=5; је исто што и x = x + 5;
x-=5; је исто што и x = x - 5;
x*=5; је исто што и x = x * 5;
x/=5; је исто што и x = x / 5;

Унарни аритметички оператори (односно оператори који врше операцију само над једним операндом) јесу:

1. Оператор ++ (оператор инкрементације), увећава вредност операнда за 1. Односно је скраћена форма овога: x++; је исто што и x=x+1;
2. Оператор -- (оператор инкрементације), умањује вредност операнда за 1. Односно је скраћена форма овога: x--; је исто што и x=x-1;

2.2 Релациони оператори

Врше поређење односа, односно релације, два операнда. Њихов резултат је тачно или нетачно. **У програмском језику С, се тачним сматра све различито од нуле, док се нула сматра као нетачно.**
Од релационих оператора, имамо:

1. Оператор == , пореди вредност левог операнда са вредношћу десног операнда. Резултат операције је тачно или нетачно.
   Примери: 3==3 резултује тачно, 3==4 резултује нетачним.
   int x=3;
   int y=4;
   x==y нетачно
   y=3;
   x==y тачно
2. Оператор !=, проверава да ли су вредности левог и десног операнда различите односно да ли нису једнаке. Примери:
   int x=3;
   int y=4;
   x!=y тачно

y=3;
x!=y нетачно

3. Оператор >, проверава да ли је вредност десни операнд већа од вредности левог. Примери:
   int x=3;
   int y=4;
   x>y нетачно
   y=5;
   x>y тачно

4. Оператор >=, проверава да ли је десни операнд већи или једнак вредности левог операнда. Примери:
   int x=3;
   int y=4;
   x>=y нетачно
   y=3;
   x>=y тачно

5. Оператор <, провера да ли је вредност левог операнда мања од вредности десног операнда. Примери:
   int x=3;
   int y=4;
   x<y тачно
   y=-1;
   x<y нетачно

6. Оператор <, провера да ли је вредност левог операнда мања од вредности десног операнда. Примери:
   int x=3;
   int y=4;
   x<y тачно
   y=-1;
   x<y нетачно
7. Оператор <=, провера да ли је вредност левог операнда мања или једнака вредности десног операнда. Примери:
   int x=3;
   int y=3;
   x<=y тачно
   y=-1;

x<=y нетачно

2.3 Логички оператори

Користимо их за логичке операције, при раду са условима, односно при конструкцији исказа. Имамо:

1. Оператор за логичко ИЛИ, који се означава као ||. Врши операцију логичког ИЛИ.
   У суштини, користимо када желимо да исказ буде тачан када је бар један од услова тачан.
   Пример: 1 || 1==2 || 0 , ово ће бити тачно јер је на почетку написано 1, што је различито од нуле, односно представља тачно.
2. Оператор за логико И, који се означава као &&. Врши операцију логичког И.
   Користимо када нам је неопходно да сви услови неког исказа буду тачни, јер је код логичког И тачно само ако су сви операнди, односно услови тачни.
   Пример: 1 && 1==1 && 2 ће бити тачно, док на пример ово неће:
   1 && 1<0 && 20
3. Оператор негације , који се означава са !. Врши операцију негације.
   Ако је нешто било тачно, након примене оператора ! ће бити нетачно и обрнуто.
   Примери: !0==3 && 3 ће бити тачно, !3==3 & 3 ће бити нетачно.

2.4 Оператор доделе вредности

Оператор доделе вредности, означава се са =. Са леве стране је неопходно да стоји операнд, који мора бити променљива у коју се може уписати вредност која ће се израчунати са десне стране. У програмском језику С оно што стоји са леве стране знака једнакости зове лврединост (***lvalue***). Са десне стране знака једнакости може писати свашта, аритметички изрази, друге променљиве, иста променљива којој се додељује вредност. Све са десне стране се претвара у конкретне вредности, ако су написане неке операције, онда се те операције извршавају над операндима и коначан резултат који се добије након свега тога се уписује у променљиву са леве стране.

Примери:

```
int x=1;
int y=2;
x = y*2 +3 +3 -4 – x*x + x + x*2 – ((x*y)-1)*2
```

3.0 Контрола тока програма

3.1 Селекције

Можемо утицати на ток извршавање нашег програма, задавањем неких логичких услова, на основу којих ће се бирати, односно одлучивати куда ће даље програм наставити са извршавањем.
На располагању имамо:
if-else, if-else if- else if- ...- else if-else, switch-case-case-case-case-case-case...case-default

3.1.1 if-else

Поставља се логички услов, односно исказ, у зависности од тачности истог, се одлучује даљи ток програма.

```
if(3==3){
  printf(“Tacno”);
}
else {
   printf(“Netacno”);

}
```

```
if(!3==3){
  printf(“Tacno”);
}
else {
   printf(“Netacno”);

}
```

Блок else није неопходан, могуће је написати и овако нешто:

```
if(3==3){
  printf(“Tacno”);
}
```

3.1.2 if-else if-...-else

Омогућава постављање више узастопних услова, од којих се редом проверава сваки и програм наставља даље на првом чији услов буде тачан. Ако ниједан од услова не буде тачан, одлази се на else.

Пример:

```
if(x>0){
        printf("x je vece od nule!");
}
else if(x==0){
        printf("x je jednako  nula!");
}
else{
// x је онда сигурно мање од нула, ако није веће ни једнако нули
        printf("x je manje od nula!");
}
```

3.1.3 switch-case-case-...-default

За разлику од прошлих селекција, *switch* се не грана на основу логичких услова, већ на основу целобројне вредности, које се зада унутар *switch* . Потребно је дефинисати посебан случај односно *case* за све потенцијалне вредности на основу које се *switch* грана. На крају сваког *case* пише се ***break***.

Блок *default* није неопходан. Али ако се деси да за вредност на основу које се switch грана НЕ ПОСТОЈИ ОДГОВАРАЈУЋ *case* програм ће се понашати непредвидиво, бесконачно тражећи *case* којег нема.

Савет: Увек написати и default блок.

```
int x=3;
switch(x){
        case 1:
                printf("x je 1!");
                break;
        case 2:
                printf("x je 2!");
                break;
        case 3:
```

```
            printf("x je 3!");
            break;
    default:
            printf("nije ni 1 ni 2 ni 3!!");
            break;

}
```

3.2. Петље

Петље нам омогућавају да више пута извршимо исту наредбу или читав блок наредби.
Тако да, ако на пример желимо да напишемо, употребом *printf* функције, 100 пута „Знање је моћ!!!“ морали би да 100 пута напишемо исту наредбу. Док употребом одговарајуће петље би то могло да буде само једна наредба. Самим тим код је компактнији.
Друга предност јесте у томе што се петља извршава на основу исказа, који се може у сваком тренутку преправити, док су наредбе које су налепљене једна за другом „забетониране“ у коду и морале би се све редом исправљати када би требало избвршити измену.
Од петљи имамо:
for, while и do-while

3.2.1 for петља

Често се назива бројачком петљом, јер њено извршавање зависи од вредности бројача. Користимо је када ТАЧНО знамо колико ће нам итерација (пролазака кроз петљу) требати. Бројач је променљива која се декларише за потребе *for* петље. Бројач је потребно поставити на неку почетну вредност. **Мора се дефинисати услов или граница до које ће бројач ићи**. Такође је неопходно дефинисати и корак модификације бројача, обично је то увећавање за 1, али може да буде било какав.

Примери неких for петљи:

```
int i;
for (i=0;i<30;i++){   // 30 итерација
    printf(„Zdravo, Svete!“);
}
```

```
int i;
int x=0;
for (i=4;i>x;i--){ // 4 итерације
    printf(„Zdravo, Svete!“);
}
```

```
int i;
int x=50;
for (i=40;i<x;i=i+10){  // Само 1 итерација
    printf(„Zdravo, Svete!“);
}
```

3.2.2 while петља

While петљу користимо када нисмо сигурни, односно не знамо тачан број итерација. Извршавање петље зависи од исказа, односно логичких услова које дефинишемо. Петља ће се понављати докле год исказ буде тачан. Да не би долазили до ситуација да имамо бесконачну петљу, морамо у току извршавања петље да некако утичемо на услове од којих зависи да ли ће се петља понављати. Ако на неки начин не утичемо на тачност услова, односно на променљиве од којих исказ зависи, постоји и други начин за излажење из тела петље, употребом наредбе *break;*
Видели смо ову наредбу код *switch*-а али је нисмо објаснили. Ова наредба тера програм да искочи из тела тренутне петље и да је заврши. Односно ако имамо нешто овако:

```
int  x=30;
while(x >0){
     if(x==5){
       break; //  искаче се из while
     }
     x  = x – 5;
}
// програм након break овде наставља извршавање.
```

Или ако имамо петљу унутар петље, само унутрашњу прекинути, јер ће break; прекинути и изаћи само из оне петље у којој је.

```
int i=90000;
while(i>0){
    while(1){  // бесконачна петља
          i--;
          if( i==0){
```

```
            break; // излази се из ове унутрашње петље
        }

    } // наставља се извршавање спољне петље, чији услов извршавања неће више бити
//тачан и самим тим ће се и спољна петља завршити.
}
```

Пример неке *while* петље где ми утичемо на услов извршавања и затим га проверавамо:

```
int x= 32,000; // Гокуов power-level током борбе са Вегетом, након што употреби Kaio-ken x4
while(x>9000){
    printf("It's OVER 9000!!!");
    x--;
}
```

3.2.3 do-while петља

Слично обичној while петљи, са једном разликом. Наредбе, односно тело петље се прво изврши, па се так онда провери тачност исказа на основу код се одлучује да ли треба поновити. Код while је било могуће да се не изврши ни један једини пут, док ће се do-while увек извршити бар једном. Користимо је када нам променљиве које модификујемо и на основу којих понављамо петљу још нису спремне, унете односно њихова вредност још није дата. Да не би стављали променљиву сами на неку почетну вредност, можемо ставити да се прво она унесе или израчуна, и да се одраде наредбе које треба извршити и да се касније процени да ли радње треба поновити.
Пример do-while петље:

```
    int x=0; // број тренутно сакупљених Змајевих кугли
    do{
        skupljanje_zmajevih_kugli(&x); // функција која симулира скупљање змајевих кугли
                                       // мора & испред x јер ће га изменити

        printf("Imam %d Zmajevih kugli",x);
}while(x <7);
```

Функција skupljanje_zmajevih_kugli(&x) може на различите начине да измени променљиву x, тако да је чак могуће из прве сакупити све Змајеве кугле и тада би се петља извршила само једном.

4.0 Кориснички мени

Употребом свега до сада наведеног, може се реализовати једноставан мени за наше конзолне апликације. Ово је класичан шаблон и ако буде у задацима потребно, користићете исти код сваки пут, са минималним изменама. Кориснички мени у конзолним апликацијама се своди на испис могућих опција, које су све нумерисане, и затим чекање на корисника да унесе неки од понуђених бројева ставки нашег менија. При избору корисника се извршава део кода који одговара ставци у менију.
Такође, употребом петљи можемо омогућити да се наш програм извршава више пута заредом, ако корисник то жели, да не мора да сваки пут изнова покреће нашу апликацију.

Прво, морамо да декларишемо неку променљиву, која ће служити за унос корисниковог избора. Рецимо да ће то бити нека променљива целобројног типа. Морамо исписати које су све опције односно избори доступни кориснику, уз помоћ *printf*. Након исписа свих опција, чекамо корисника да унесе, где ћемо употребити *scanf*.
Затим, морамо да напишемо одговајући *switch* који ће да покрије све могуће изборе корисника, обавезно са default блоком. Међу case-ове *switch-a* ћемо стављати код који треба да се изврши.

Ако желимо да се наш програм може заредом извршавати, морамо цео код менија, заједно са исписом, прихватањем избора корисника и целим *switch*-ом ставити унутар једне do-while петље, која ће се вртети докле год корисник не изабере опцију за излаз из програма.

Пример овог шаблона је на следећој страни и доступан за скидање на следећем линку:
Сајт школе Рајак

```
/**********************************
PROGRAMERSKA RADIONICA
SKOLA RAJAK
----------------------
Autor: Sebastian Novak
Datum: 3.3.2013.
Verzija: 1.0
Kontakt:
programerska.radionica@gmail.com
www.rajak.rs
Opis:
Sablon za korisnicki meni
konzolnih
aplikacija.
**********************************/

#include <stdio.h>

int main()
{
   int izbor;
   int x=0, y=0, rez;

   do{
      printf("PROGRAM ZA VRSENJE OSNOVNIH ARITMETICKIH OPERACIJA 1.0\n----------\n");
      printf("Autor: Sebastian Novak www.rajak.rs\n\n");
      printf("1. Unos operanada x i y\n");
      printf("2. Ispis trenutne vrednosti x i y\n");
      printf("3. Sabiranje x i y, ispis rezultata\n");
      printf("4. Oduzimanje x i y, ispis rezultata\n");
      printf("5. Mnozenje x i y, ispis rezultata\n");
      printf("6. Celobrojno deljenje x i y, ispis rezultata\n");
      printf("1337. IZLAZ\n==============\n");
      printf("Vas izbor?");
      scanf("%d", &izbor);

      switch(izbor){
      case 1:
         printf("\nUnesite neku celobrojnu vrednost za vrednost operanda x:");
         scanf("%d",&x);
         printf("\nUnesite neku celobrojnu vrednost za vrednost operanda y:");
```

```
        scanf("%d",&y);
        break;
    case 2:
        printf("\nVrednost operanda x je %d \n", x);
        printf("Vrednost operanda y je %d \n", y);
        break;
    case 3:
        rez = x + y;
        printf(" %d + %d = %d \n", x,y,rez);
        break;
    case 4:
         rez = x - y;
        printf(" %d - %d = %d \n", x,y,rez);
        break;
    case 5:
        rez = x * y;
        printf(" %d * %d = %d \n", x,y,rez);
        break;
    case 6:
        rez = x / y;
        printf(" %d / %d = %d \n", x,y,rez);
        break;
    case 1337:
        break;
    default:
        printf("Los unos, proverite brojeve validnih opcija!\n");
        break;
    }


}while(izbor!=1337);


return 0;

}
```