Подела функција

У C++ програмима постоји велики број функција попут функције main.
Подела потребних инструкција у програму на више функција смањује комплексности кода.
Функције се често зову и модули.
Постоје предефинисане (уграђене) функције и кориснички дефинисане функције.
Предефинисане функције се налазе практично у свим хедер фајловима.

функција	**хедер**	**употреба**	**тип параметара**	**тип резултата**
abs(x)	<cmath>	abs(-7) = 7	int(double)	int(double)
ceil(x)	<cmath>	ceil(56.34) = 57.0	double	double
exp(x)	<cmath>	exp(1.0) = 2.718	double	double
fabs(x)	<cmath>	fabs(-5.67) = 5.67	double	double
floor(x)	<cmath>	floor(45.67) = 45.00	double	double
islower(x)	<cctype>	islower('h') = 1	int	int
isupper(x)	<cctype>	isupper('K') = 1	int	int
pow(x,y)	<cmath>	pow(2.2, 1.2) = 2.57	double	double
sqrt(x)	<cmath>	sqrt(4.0) = 2.0	double	double
tolower(x)	<cctype>	tolower('A') = 79 ('a')	chr	int
toupper(x)	<cctype>	toupper('a') = 56 ('A')	chr	int

Кориснички дефинисане функције

Кориснички дефинисане функције се деле на функције које враћају вредност и функције које не враћају вредност.

Функције које враћају вредност

Ако се посматра предефинисана функција, прво се у код мора укључити хедер са том функцијом.
Затим се мора знати: име функције, параметри функције (ако их има), тип сваког од параметара, тип резултата функције и код.
Функције које враћају вредност се користе у исказима доделе, као параметри при позивању других функција или као излазни искази.
Код у којем хоћемо да користимо корисничку функцију која враћа вредност мора се састојати од:

```
heder
// prototip funkcije
using namespace std;
int main()
{
        …
        //poziv funkcije
        …
}
//definicija funkcije, zaglavlje funkcije
{
        // telo funkcije
        // vraćena vrednost
}
```

Хедер нема никакве директне везе са кориснички дефинисаном функцијом.
Дефиниција функције се састоји од :
tipFunkcije imeFunkcije(lista formalnih parametara)
{ telo funkcije; return vrednost}
Вредност функције се враћа на место позива функције.
Позив функције (function call) се састоји од:
imefunkcije(lista argumenata)
Листа правих параметара може бити и празна, а тада и листа формалних параметара може бити празна.
Са исказом return izraz; се враћа вредност на место позива функције.
Пре старта кода било које функције се постављају прототипови корисничких фукција, тј заглавље функције без тела функције.

Пример: хоћемо сами да дефинишемо своју функцију која ће да да као резултат апсолутну вредност целог броја:

```
#include<iostream>
int apsolutnaVrednost(int);
using namespace std;

int main()
{
	int x = 0;
	cout << "Unesi ceo broj: ";
	cin >> x;
	int a = apsolutnaVrednost(x);
	cout << "Apsolutna vrednost broja " << x << " je " << a << endl;
	return (0);
}

int apsolutnaVrednost(int broj)
{
	if (broj < 0) broj = -broj;
	return broj;
}
```

Дефиниција функције

```
tipFunkcije imeFunkcije(lista formalnih parametara)  // heder funkcije
{
	//telo funkcije;
	 return vrednost;
}
```

tipFunkcije је било који тип који се користи као тип и за промењиве. Ако функција не враћа никакву вредност тип је void. Када се не наведе тип функције, подразумева се да је int.

imeFunkcije је било које име које се повинује правилима задавања имена промењивих.

lista formalnih parametara се састоји од било којег броја параметара. Сваки параметар има тип и име. Тип и име параметра се повинују правилима за тип и име промењиве. Могуће је и да не постоји нити један параметар.

Тело функције је низ наредби које се извршавају као саставни део функције.

return vrednost је инструкција којом се у главни програм враћа вредност која представља вредност функције. Наведени тип функције је исти као и тип вредности која се враћа у главни програм. Циљ функције је заправо израчунати ову вредност и вратити је у главну функцију.

Пример: Дефинисати функцију која рачуна збир два цела броја.

```
int zbir (int a, int b)
{
	int z;		// posebno se definise promenljiva z koja predstavlja zbir dva broja
	z=a+b;		// izracunavamo zbir parametara funkcije
	return z;	// u glavni program se vraca vrednost promenljive z
}
```

Други начин:

```
int zbir (int a, int b)
{
	return a + b ;		// u glavni program se vraca vrednost matematickog izraza a+b
}
```

Позив функције
imefunkcije(lista argumenata)
imefunkcije је исто име које се користи при дефинисању функције.
lista argumenata (листа правих параметара) се састоји од истог броја елемената колико је и параметара у дефиницији функције.
Тип аргумената мора да одговара типу параметра који се налази на истом месту у листи формалних параметара.
Име аргумента не мора да одговара имену одговарајућег параметра у листи формалних параметара.
Позив функције се може појавити као операнд у изразу.

Пример: Написати позив већ дефинисане функције zbir (функција која рачуна збир два цела броја).
```
rezultat = zbir(x, y);          // x, y  i rezultat su već deklarisane kao int
```

Пример: Написати програм који коришћењем функције приказује збир два цела броја.
```
#include <iostream>
using namespace std;
int zbir (int a, int b)                 // funkcija se definise pre glavnog programa
{
        return a + b;                   // u glavni program se vraca vrednost funkcije tamo gde je pozvana
}
int main()                              // glavni program
{
        int x = 0, y = 0, rezultat = 0;
        cout << “Unesi dva broja: “;
        cin >> x >> y;
        rezultat = zbir(x, y);          // ovde se poziva funkcija kao operand u izrazu
        cout << “Rezultat sabiranja je: “ << rezultat << endl;
        return 0;
}
```

Могуће је и поједноставити код изостављањем промењиве rezultat и постављањем позива функције у излазни ток података:
```
cout << “Rezultat sabiranja je: “ << zbir(x, y) << endl;
```
Домаћи задатак 01: Написати код за корисничку функцију sabiranjeTriCela која сабира три цела броја.
Домаћи задатак 02: Написати код за корисничку функцију apsolutnoSabiranje која сабира апсолутне вредности два цела броја.

Протутип функције
Протутип функције се састоји од:
tipFunkcije imeFunkcije(lista formalnih parametara);
tipFunkcije и imeFunkcije одговарају типу и имену приликом дефиниције функције.
lista formalnih parametara може састојати или од потпуне листе параметара исто као код дефиниције функције или само од типова параметара (без имена параметара).
Имена параметара не морају да буду иста као имена параметара у дефиницији функције.
Једина велика разлика са дефиницијим функције јесте ; на крају линије.

Пример: Протутип функције која рачуна збир два цела броја.
```
int zbir (int a, int b);
```
Или
```
int zbir (int, int);
```

Пример: Програм који рачуна збир два цела броја.

```
#include<iostream>
int zbirDvaCelaBroja(int a, int b);
using namespace std;

int main()
{
	int x = 0, y = 0, rezultat = 0;
	cout << "Unesi dva cela broja: ";
	cin >> x >> y;
	rezultat = zbirDvaCelaBroja(x, y);
	cout << "Rezultat sabiranja je: " << rezultat << endl;
	return (0);
}

int zbirDvaCelaBroja(int a, int b)
{
	return a + b;
}
```

Редослед функција у структури кода је значајна. Ако се у коду прво појави позив функције а затим дефиниција функције, програм се неће извршити јер није могуће реализовати функцију која није прво дефинисана па затим позвана (исписујем пример добро постављеног редоследа и лоше постављеног редоследа функција у коду).
Зато се користи протутип функције који се поставља испред главне функције. Ако постоји протутип више није битно да ли је у структури кода прво написан дефиниција или позив функције. За протутип је најбитније да редослед и број типова параметара одговара редоследу и броју типова параметара у дефиницији функције.

Пример: Написати програм који исписује већи од два унета броја.

```
#include<iostream>
double veci(double, double);
using namespace std;

int main()
{
	double x = 0, y = 0;
	cout << "Unesi dva broja: ";
	cin >> x >> y;
	cout << "Veci broj je: " << veci(x, y) << endl;
	return (0);
}

double veci(double a, double b)
{
	if (a >= b) return a;
	else return b;
}
```

Домаћи задатак 03: Написати програм који приказује разлику два цела броја.
Домаћи задатак 04: Написати код за корисничку функцију apsolutnoOduzimanje која одузима апсолутне вредности два цела броја.

Дефиниције, позив и прототип функције

```
1  // program:     Vezba01primer01
2  // fajl:        SizeOf.cpp
3  // funkcije:    main, sizeof
4  // opis:        odredjuje zauzetu memoriju za razlicite tipove podataka
5  // autor:       Rankovic Nebojsa
6  // revizije:    30.05.2016. v1.00
7  // napomena:    deo 01 casa vezbi funkcije
8
9  #include<iostream>
10 using namespace std;
11
12 int main()
13 {
14     cout << "Tip int zauzima \t" << sizeof(int) << " bajta." << endl;
15     cout << "Tip short int zauzima \t" << sizeof(short) << " bajta." << endl;
16     cout << "Tip long int zauzima \t" << sizeof(long) << " bajta." << endl;
17     cout << "Tip char zauzima \t" << sizeof(char) << " bajt." << endl;
18     cout << "Tip float zauzima \t" << sizeof(float) << " bajta." << endl;
19     cout << "Tip double zauzima \t" << sizeof(double) << " bajtova." << endl;
20     return 0;
21 }
```

На екрану треба да се добије налик следећим резултатима:

```
Tip int zauzima             4 bajta.
Tip short int zauzima       2 bajta.
Tip long int zauzima        4 bajta.
Tip char zauzima            1 bajt.
Tip float zauzima           4 bajta.
Tip double zauzima          8 bajtova.
```

```
// program:   ZbirTriBroja
// fajl:      ZbirTrisaFunkDva.cpp
// funkcije:  main, zbirDvaCelaBroja
// opis:      sabira tri cela broja pomocu funkcije koja sabira dva cela broja
// autor:     Rankovic Nebojsa
// revizije:  07.06.2016. v1.00
// napomena:  deo 05 casa teorije funkcije

#include<iostream>
using namespace std;
int zbirDvaCelaBroja(int a, int b)
{
	return a + b;
}

int main()
{
	int x = 0, y = 0, z = 0;
	cout << "Unesi tri cela broja: ";
	cin >> x >> y >> z;
	cout << "Rezultat sabiranja je: " << //funkcija se poziva kao argument
		zbirDvaCelaBroja(z, zbirDvaCelaBroja(x, y)) << endl;
	return (0);
}
```

```
#include<iostream>
using namespace std;
int sabiranjeBrojevaDon(int);

int main()
{
	int n;
	cout << "Unesti poslednji broj za sabiranje: ";
	cin >> n;
	cout << "Zbir prvih " << n << " brojeva je: " << sabiranjeBrojevaDon(n) << endl;
	return (0);
}

int sabiranjeBrojevaDon(int a)
{
	int zbir = 0, i = 0;
	for (i = 1; i <= a; i++) zbir = zbir + i;
	return zbir;
}
```

Домаћи задатак 05: Написати програм који претвара унете метре у километре и центиметре.
Домаћи задатак 06: Написати програм који рачуна средњу вредност 5 унетих оцена.

Параметри и аргументи функција

Подаци који се наводе при позиву функције (нпр: Funkcija1(argument1, argument2,...)) се називају аргументи функције. Да би се разликовали од других података, понекад се називају стварни аргументи. Након позива функције са стварним аргументима, праве се копије података, тако да приликом извршења функције било каква промена података не мења податак који је прослеђен као аргумент.
Подаци који се наводе у хедеру дефиниције функције (нпр: Funkcija1(parametar1, parametar2, ...)) се називају параметри функције. Они нису стварни подаци. Они само формално представљају податке који ће се навести приликом позивања функције (стварни аргументи) и на основу чијих вредности ће се израчунати вредност функције. Параметри функције понекад се зову и формални аргументи.

```
#include <iostream>
using namespace std;
int Funkcija(int a)
{
	a = 3;
	return a;
}
int main()
{
	int a = 1;
	cout << "Pre poziva funkcije a=/t" << a << endl;
	cout << "Unutar funkcije a=/t" << Funkcija(a) << endl;
	cout << "Posle poziva funkcije a=/t" << a << endl;
	return (0);
}
```

Када постоји потреба да се параметри функције не мењају унутар неке функције, довољно је приликом дефинисања параметара ставити реч const (нпр: int Funkcija(const int a, const double b, const char c)).
Промењиве декларисане унутар функције су приватне, локалне промењиве, за ту функцију. Када се функција позове, компјутер обезбеђује меморијски простор потребан за те промењиве. Када се функција заврши, компјутер ослобађа меморију која се користила за те промењиве. Из тог разлога локална промењива a за функцију main нема исту вредност као локална промењива a за функцију Funkcija.

За разлику од локалних промењивих, постоје глобалне промењиве које се могу користити у свим функцијама.

```
#include <iostream>
using namespace std;
float a, b;                // definisanje globalnih promenljivih
float pravougaonik()
{
	return a * b;
}
int main () // glavni program
{
	cout << "unesi stranice pravougaonika" << endl;
	cin >> a >> b;
	cout << "povrsina pravougaonika je P=" << pravougaonik() << endl;
	return 0;
}
```

Из примера се види да су промењиве a и b глобалне промењиве и да се са њима може радити у свакој функцији као да су локалне за ту функцију. Функција нема аргументе у позиву јер се вредности у промењивима мењају само у главној функцији. Проблем представља ако се глобалне промењиве користе у великом броју функција што отежава контролу над променама вредности у промењивима па се препоручује коришћење искључиво локалних промењивих.

```
#include<iostream>
using namespace std;
void prviPoziv();
void drugiPoziv();
double a = 1.25, b = 5.75; int c = 11, malo = 100;
int main() {
	cout << a << " b = " << b << " c = " << c << " malo = " << malo << endl;
	prviPoziv();
	cout << a << " b = " << b << " c = " << c << " malo = " << malo << endl;
	drugiPoziv();
	cout << a << " b = " << b << " c = " << c << " malo = " << malo << endl;
	return(0);}

void prviPoziv(){
	double c = 11.05;
	cout << " a = " << a++ << " b = " << b++ << " c = " << c << " malo =" << malo++ << endl;
}

void drugiPoziv(){
	const int malo = 0; double c = (a + b) * malo;
	cout << " a = " << a++ << " b = " << b++ << " c = " << c << " malo =" << malo << endl;
}
```

Домаћи задатак 07: Израчунати површину и запремину базена само са глобалним промењивима и функцијама.
Домаћи задатак 08: Израчунати површину и запремину базена са функцијама и без глобалних промењивих.

<u>Додатне особине функција</u>

Функције се могу поделити у две категорије: оне које не враћају вредност и оне које враћају. Функције које не враћају вредности имају следећи облик:

```
void imeFunkcije(listaParametara)
{
	iskazi;
	return;   //opciono
}
```

Код функција које враћају вредност, вредност може бити константа, промењива или некакав други израз.
Вредност која се враћа не може бити у облику низа.
Обично, функција враћа вредност копирањем вредности у посебан процесорски регистар или меморијску локацију.
Затим позвана функција испита ову локацију због потребе да се све сложи око броја и типа аргумената.
Када се дефинише функција, користи се зарез за одвајање декларисаних параметара у листи параметара.
Иако су параметри истог типа сваки тип се мора посебно декларисати: void xxx(float a, float b).

```
// program:   RazlicitiArgumenti
// fajl:      RazArg.cpp
// funkcije:  main, brojKaraktera
// opis:      prikazuje znak n puta u nizu
// autor:     Rankovic Nebojsa
// revizije:  12.06.2016. v1.00
// napomena:  deo 07 casa teorije funkcije

#include<iostream>
using namespace std;
void brojKaraktera(char, int);

int main()
{
	int ponavljanja;
	char znak;
	cout << "Unesi znak: ";
	cin >> znak;
	while (znak != 'q')
	{
		cout << "Unesi ceo broj: ";
		cin >> ponavljanja;
		brojKaraktera(znak, ponavljanja);
		cout << "\nUnesi sledeci znak ili pritisni q za kraj" << endl;
		cin >> znak;
	}
	cout << "Kraj programa" << endl;
	return(0);
}

void brojKaraktera(char c, int n)
{
	while (n-- > 0) cout << c;
}
```

У коду се користи `cin >> znak` за читање знака са тастатуре.
На овај начин се прескачу знаци као што су белине или ескејп карактери као што је знак за нови ред.
Када би се користио: cin.get() за читање улазних знакова са тастатуре, у улазном току би се нашли и знак за нови ред пошто се притиском на ENTER прелази на следећи ред при уносу.

```
// program:   Faktorijel
// fajl:      Source.cpp
// funkcije:  main, faktorijel
// opis:      racunanje faktorijela celog pozitivnog broja
// autor:     Rankovic Nebojsa
// revizije:  12.06.2016. v1.00
// napomena:  deo 07 casa teorije funkcije

#include<iostream>
using namespace std;
int faktorijel(int);
int main()
{
	int broj = 0;
	cout << "Unesi broj za racunanje faktorijela: ";
	cin >> broj;
	cout << broj << "! = " << faktorijel(broj) << endl;
	return(0);
}

int faktorijel(int n)
{
	int i = 0, proizvod = 1;
	for (i = n; i != 0; i--)
	{
		proizvod *= i;
	}
	return proizvod;
}
```

Примена претходног кода у следећем задатку:

```
// program:   LotoSanse
// fajl:      Source.cpp
// funkcije:  main, verovatnoca
// opis:      racunanje verovatnoce dobijanja na loto-u
// autor:     Rankovic Nebojsa
// revizije:  12.06.2016. v1.00
// napomena:  deo 07 casa teorije funkcije

#include<iostream>
using namespace std;
long double verovatnoca(unsigned, unsigned);

int main()
{
	double brojMogucnosti = 0.00, brojIzvlacenja = 0.00;
	cout << "Unesi broj mogucih brojeva koji se izvlace ";
	cout << "i koliko se brojeva izvlaci: ";
	while ((cin >> brojMogucnosti >> brojIzvlacenja) && (brojIzvlacenja <= brojMogucnosti))
	{
		cout << "Imas jednu sansu u ";
		cout << verovatnoca(brojMogucnosti, brojIzvlacenja);
		cout << " da dobijes." << endl;
		cout << "Sledece vrednosti (q za kraj): ";
	}
	cout << "Kraj programa" << endl;
	return(0);
}
```

```
long double verovatnoca(unsigned a, unsigned b)
{
	long double rezultat = 1.0, n = 0.00;
	unsigned p;

	for (n = a, p = b; p > 0; n--, p--)
	{
		rezultat = rezultat * n / p;
	}
	return rezultat;
}
```

Види се у примеру да функција verovatnoca приказује две врсте локалних промењивих које се могу имати у функцији. Прво постоје формални параметри (a, b) који су декларисани у хедеру дефиниције функције а постоје и локалне промењиве (rezultat, n, p) које постоје само док се сама функција извршава и које добијају своје вредности унутар саме функције. Кликом на q се неиспуњава услов опстанка у петљи (кликом на било који друго слово или давањем мање вредности за први параметар од другог).
Домаћи задатак 09: Исписати изабрани карактер по једном у жељени број редова.
Домаћи задатак 10: Испитати који је број од три унета броја највећи.

Рекурзивне функције
Рекурзивне функције су функције које непосредно или посредно позивају саме себе.
Све рекурзивне функције се своде на циклус понављања.
Оне позивају себе саме али са измењеним аргументима.
Проблем је што извршавање ових функција траје веома дуго и што троше превише меморијских ресурса за рад (RAM меорија).

Пример рачунање факторијела коришћењем рекурзивне функције:

```
// program:   RekurzFaktor
// fajl:      RekurzivniFaktorijel.cpp
// funkcije:  main, rekurzija
// opis:      racunanje faktorijela preko rekurzije
// autor:     Rankovic Nebojsa
// revizije:  12.06.2016. v1.00
// napomena:  deo 08 casa teorije funkcije

#include<iostream>
using namespace std;
int rekurzija(int);

int main()
{
	int broj;
	cout << "Uneti broj za racunanje faktorijela: ";
	cin >> broj;
	cout << "Faktorijel od " << broj << " je " << rekurzija(broj) << endl;
	return(0);
}

int rekurzija(int a)
{
	return (a > 0) ? (a * rekurzija(a - 1)) : 1;
}
```

Вредност функције се добија условним изразом којим се или израчунава факторијел по формули или се узима константа 1.
Израчунавање формуле подразумева поновно позивање функције факторијел са аргументом a-1 и то се понавља све док параметар не постане нула.
Када параметар постане нула бира се константа 1.
Ово је поступак за израчунавање међурезултата који се понавља све док се не добије вредност функције за провобитни позив (rekurzija (a)).

Пример са исписивањем унетог броја уназад помоћу рекурзије:

```
#include<iostream>
using namespace std;
void rekurzija(int);

int main()
{
	int n;
	cin >> n;
	rekurzija(n);
	return(0);
}

void rekurzija(int x)
{
	cout << x % 10;
	if (x / 10 > 0) rekurzija(x / 10);
}
```

Пример сабирања целих бројева од 1 до n помоћу рекурзије:

```
#include<iostream>
using namespace std;
int rekurzija(int);

int main()
{
	int n = 0;
	cin >> n;
	cout << rekurzija(n) << endl;
	return(0);
}

int rekurzija(int x)
{
	if (x == 0) return(0);
	else return (x + rekurzija(x - 1));
}
```

Домаћи 11: Написати програм који исцртава странице квадрата помоћу низа карактера.
Домаћи 12: Написати програм који прерачунава Целзијусове у Фахренхајтове степене и обрнуто по потреби.