План и програм рада на предмету Програмирање

Теме: Увод у структуре програмског језика C++, Функције и структура програма, Показивачи, Матрице, Стрингови, Технике сортирања низова, Слогови, Датотеке, Динамичке структуре података.

Основни кораци у начину коришћења MS Visual Studio окружења

Почетни екран. Избор Tools, Imort and Export Settings Wizard, Reset All Settings, избор Visual C++.

Повратак на почетни екран. Избор New Project... Са леве стране је картица Project types (VS2008) Installed (VS2015), Templates, Visual C++.

На средишњој картици Win32 Console Application. Испод на линији Location изабрати локацију на којој ће бити смештени сви потребни фајлови за рад са креираним пројектом.

Најбоље је ући у Projects у аутоматско изабраном фолдеру и креирати неколико подфолдера за удобнији рад. Променити име фајла по избору, тако ће се назвати и фолдер за нови пројекат. Отвара се Win32 Application Wizard, Next, Empty project, Finish. Са леве стране се појављује Solution Explorer. Кликнути десним кликом на Source Files, Add, New Item. Кликнути на C++ File (cpp) а по жељи променити назив фајла (не дирати екстензију). Појављује се изабрани фајл под фолдером Source Files и отвара се централни прозор као едитор кода.

```
1     #include<iostream>
2
3     using namespace std;
4
5     int main()
6     {
7         cout << "Zdravo!\n";
8         system("pause");
9         return 0;
10    }
```

Укуцати:

Клик на Build, Clean Solution.

Клик на Build, Build Solution.

Клик на Debug, Start Debugging.

Појављује се конзола са излазним резултатом кода и поруком о чекању на притисак било којег дугмета за завршетак рада.

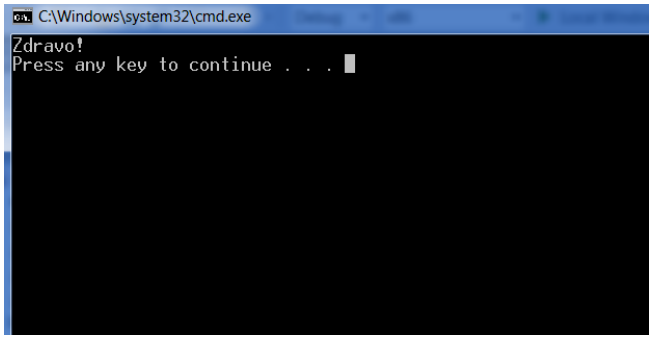
Zdravo!

Press any key to continue...

После првог стартовања преко Start Debugging, могуће је стартовати сорс фајл и као Start Without Debugging. У овом случају се аутоматски после приказа резултата зауставља гашење екрана конзоле (као да постоји system ("pause")).

Ако се код преправи у C++ стилу:

```
1   #include<iostream>
2   using namespace std;
3
4   int main()
5   {
6       cout << "Zdravo!";
7       cout << endl;
8       return 0;
9   }
```



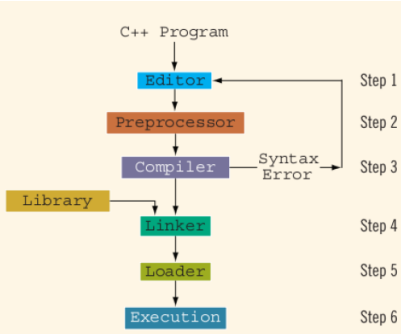
Ако је величина конзоле мала, десни клик на горњи плави део оквира конзоле, избор опције Properties и избор веће величине конзоле.

Кораци у извршавању C++ кода

У едитору се куца код програма у жељеном програмском језику.

Сваки C++ сорс програм се мора снимити са .cpp екстензијом. Исказ који започиње са # се састоје од препроцесорских директива. Оне се обрађују у посебном програму који се назива препроцесор.

Препроцесорске директиве омогућавају коришћење екстерних библиотека у самом коду. У кораку 3 компајлер проверава да ли сорс програм поштује синтаксна правила и ако нема грешака преводи код у машински језик (програм добијен овим поступком је објекат). У кораку 4 се проверавају библиотеке за потребне програме који помажу извршавање објекта. Ово омогућавају програми звани линекери. Да би је извршни програм могао стартовати треба га учитати у главну меморију а то се ради помоћу програма лоадера.



Структура C++ програмског језика

пример структурисаног кода.

```
#include <iostream>
using namespace std;
int main()
{
	cout << "Moj prvi C++ program."  << endl;
	return 0;
}
```

Указујем на визуелно одвојене делове структуре кода: коментари, позивање библиотека за коришћење предефинисаних објеката, омогућавање скраћеног писања команди, хедер функције main, декларација промењивих, алгоритам функције.

`cout << "Moj prvi C++ program."  << endl;` је излазни израз.

Рачунар проверава израз након пара симбола << и приказује резултат на екрану. Све што је унутар "" је стринг.

Коментари се пишу у једном реду **//** или у више редова **/*...*/**.

Позивање библиотеке **(#include <iostream>**) за рад са предефинисаним објектима за генерисање излаза (**cout**), улаза (**cin**) и коришћење манипулатора (**endl**).

Коришћење исказа (**using namespace std;**) што омогућава коришћење скраћено писање команди без обавезе коришћења префикса std::.

После навођења главне функције, лева витичаста заграда означава почетак тела функције, десна означава завршетак тела функције. Оператор **<<** је оператор уноса тока (stream insertion operator).

Најмања појединачна јединица програма се назива **токен**. У C++ токени се деле на специјалне симболе, специјалне речи и идентификаторе. Специјални симболи: +, -. *, /, blank, ., ;, ?, ,, <=, !=, ==, >=, ++, --. Идентификатори морају почети са словом или _, и састоје се од слова, цифара и _. C++ разликује велика и мала слова.

Зашто ово нису добри идентификатори: plata zaposlenog, Zdravo!!, jedan+dva, 19tog ?

Основни типови података

У C++ типови података се деле у три категорије: основни типови, структурисани типови и поинтери.

Основни типови су : цели бројеви (integral), децимални (floating-point) и енумератори који су кориснички дефинисани.

Цели бројеви се даље могу представити као: char (-128 до 127, бајт), short (-32.768 до 32.767), int (-2.147.483.648 до 2.147.483.647, 4 бајта), long, bool (true, false, бајт), unsigned char, unsigned short, unsigned int, unsigned long, long long (-2^{63} до 2^{63}, 64 бајта), unsigned long long.

Тип података **char** се највише користи за представљање једног карактера кои се може откуцати са тастатуре. Пример овог типа података: 'A', 'a', '5', '*', '!=' . Сваки од ових ASCII карактера има своју вредност од 0 до 128.

За приказ бројева са децималом C++ користи нотацију са покретном тачком (floating-point notation). Примери:

Децимални број	**Научна нотација**	**C++ нотација са покретном тачком**
75.924	$7.5924 * 10^{1}$	7.592400E1
0.18	$1.8 * 10^{-1}$	1.800000E-1
0.0000453	$4.53 * 10^{-5}$	4.530000E-5
-1.482	$-1.482 * 10^{0}$	-1.482000E0
7800.0	$7.8 * 10^{3}$	7.800000E3

Постоје два типа података за рад са децималним бројевима у C++: **float** ($-3.4*10^{38}$ до $3.4*10^{38}$, четири бајта, број значајних цифара је шест или седам) и **double** ($-1.7*10^{308}$ до $1.7*10^{308}$, осам бајтова, број значајних цифара је 15). Понекад се ова два типа називају single precision и double precision.

Тип података **string** је кориснички дефинисан тип података. Мора се позвати библиотека са његовим компонентама. Стринг је низ од нула или више карактера и увек је унутар наводника. Стринг без карактера се зове null или празан стринг. Позиција првог каракетра се обележава са 0. Дужина стринга укључује и бела места у стрингу.

Декларација промељиве: tipPodataka identifikator;

Додељивање се изводи преко оператора доделе (**assignment operator**): broj = 5;

Аритметичке операције са целим бројевима: 5 / 2 даје 2, 14 /7 даје 2, 34 % 5 даје 4, 4 % 6 даје 4. Ови примери су са бинарним операторима. Унарни оператор: -5.

Аритметичке операције са негативним бројевима: -34 % 5 даје -4, 34 % -5 даје 4, -34 % -5 даје -4.

Ако су оператори различитих типова, int оператор се претвара у реалан број са децималним делом 0.

preincrement x=5; y=++x; (x=6, y=6), postincrement x=5; y=x++; (y=5, x=6).

Конверзије

Ако се дозволи аутоматска конверзија (имплицитна) једног типа података у други, може доћи до непредвиђених грешака. Да би се спречило, користи се експлицитна конверзија типа, коришћењем каст оператора (type casting):
static_cast<nazivTipaPodataka>(izraz)
Прво се изврши евалуација израза а затим и конверзија његове вредности у тип одређен у <nazivTipaPodataka>.
Примери: **static_cast**<int>(7.8 + **static_cast**<double>(15)/2) = **static_cast**<int>(7.8 + (15.0/2.0)) = **static_cast**<int>(7.8 + 7.5) = **static_cast**<int>(15.3) = 15;
овде се кастовање целог у децимал односи на 15, а у следећем примеру на 15/2:
static_cast<int>(7.8 + **static_cast**<double>(15/2)) = **static_cast**<int>(7.8 + 7.0) = **static_cast**<int>(14.8) = 14
Слично се врши кастовање знаковног типа у целобројни и обр: **static_cast**<int>('A') = 65; **static_cast**<char>(65) = 'A';

Константе

Именована константа је меморијска локација чији садржај се не сме мењати током рада програма:
const double BROJ = 5.469; const int BROJ_PUTNIKA = 32; const char PRAZNO = ' ';
Обтратити пажњу да ако се хоће да константа BROJ буде типа float: const flot BROJ = 5.469f;
За све типове података важи да се први извршава евалуација израза са десне стране једнакости па онда се смешта добијена вредност у меморијску локацију именовану са идентификатором са леве стране једнакости

```
#include<iostream>
#include<string>
using namespace std;
int main()
{
        int broj;
        double cena;
        char prvi;
        string recenica;
        broj = 4;
        cout << “broj = ” << broj << endl;
        cena = 0.02 * 1000;
        cout << “cena = ” << cena << endl;
        prvi = ‘D’;
        cout << “prvi = ” << prvi << endl;
        recenica = “Suncan je dan.”;
        cout << “recenica = ” << recenica << endl;
        return 0;
}
```

На екрану се добија: broj = 4; cena = 20; prvi = D; recenica = Suncan je dan.
Пример: x = y = z; ово значи да је прво y = z а онда да је нова вредност за y : x = y.

Излазни искази (Output)

Стандардни излазни уређај је екран монитора.
cout << izraz ili manipulator << ... и назива се исказ излаза.
<< је оператор уноса тока (stream insertion operator).
Када се исказ евалуира његова вредност се приказује на месту уноса на излазном уређају.
Манипулатор се користи за форматирање излаза. **endl** је манипулатор који чини да се место уноса помери на почетак следеће линије екрана. Исти ефекат се добија:
cout << ‘\n’, cout << “\n”, cout << endl.
Где је \ ескејп карактер а \n је ескејп секвенца нове линије.

Узлазни искази (Input)

Уношење података у промењиве се изводи се на два начина: коришћењем израза доделе или коришћењем улазног исказа input. C++ омогућава истовремену декларацију и иницијализацију промењивих:
int first = 13, second = 10; char x = ‘ ’;
Смештање података у промењиве преко стандардних улазних уређаја се врши преко исказа:
cin >> promenjiva>>...;
Овде се користи оператор **>>** (stream extraction operator) оператор екстракције тока.

```
#include <iostream>
#include <string>
using namespace std;
int main()
{
	string ime, prezime;
	int starost;
	double tezina;
	cout << "Unesi ime, prezime, godine starosti i tezinu, "
		<< "odvojene belim mestima." << endl;
	cin >> ime >> prezime;
	cin >> starost >> tezina;
	cout << "Ime i prezime: " << ime << " " << prezime << endl;
	cout << "Godine starosti: " << starost << endl;
	cout << "Tezina: " << tezina << endl;
	return 0;
}
```

```
Unesi ime, prezime, godine starosti i tezinu, odvojene belim mestima.
Pera Peric 16 45.5
Ime i prezime: Pera Peric
Godine starosti: 16
Tezina: 45.5
Press any key to continue . . .
```

Стрим

Stream је низ карактера од извора према циљу.

Хедер фајл **iostream** садржи и дефиниције два типа података: istream (input stream) и ostream (output stream).

Као и две декларације промењивих: cin – common input, cout – common output.

Оператор екстракције **>>** је бинаран и тражи два операнда, леви је input stream промењива попут cin а оператор десне стране је промењива.

Када >> скенира за следеће инпут, прескаче све карактере до празног места.

Примери:

Нека је декларисано

int a,b;

double z;

char ch;

iskaz	**ulaz**	**vrednost u memoriji**
cin >> ch;	A	ch = 'A'
cin >> ch;	AB	ch = 'A', 'B' se čuva za sledeći ulaz
cin >> a;	48	a = 48
cin >> a;	46.35	a = 46, .35 se čuva za sledeći ulaz
cin >> z;	74.35	z = 74.35
cin >> z;	39	z = 39.0
cin >> z >> a;	65.78.38	z = 65.78, a = 38
cin >> a >> b;	4 60	a = 4, b = 60
cin >> a >> z;	46 32.3 68	a = 46, z = 32.3, 68 se čuva za sledeći ulaz

Предефинисане функције

Функција main се аутоматски извршава када се стартује програм.

Друге функције се извршавају само када су позване.

Већ написане функције се називају предефинисане функције.

Оне се налазе организоване у колекције библиотека, званих хедер фајлови.

Нпр. за коришћење предефинисане функције pow, мора се укључити у код позивање библиотеке cmath. pow(2.0, 3.0) = $2.0^{3.0}$ = 8.0, и овде су вредности у загради аргументи или параметри функције pow.

pow(2.0, 3.0) се назива позив функције.

```
#include <iostream>
#include <cmath>
#include <string>
using namespace std;
```

```
const double PI = 3.1416;
int main()
{
	double poluprecnik, zapremina, x1, y1, x2, y2, rastojanje;
	string recenica;
	cout << "Unesi poluprecnik sfere: ";
	cin >> poluprecnik;
	cout << endl;
	zapremina = (4.0 / 3) * PI * pow(poluprecnik, 3);
	cout << "Zapremina sfere je: " << zapremina << endl << endl;
	cout << "Unesi koordinate dve tacke u X-Y povrsi: ";
	cin >> x1 >> y1 >> x2 >> y2;
	cout << endl;
	rastojanje = sqrt(pow(x2 - x1, 2) + pow(y2 - y1, 2));
	cout << "Rastojanje izmedju tacaka "
		<< "sa unetim koordinatama je: "
		<< rastojanje << endl << endl;
	recenica = "Programiranje u C++";
	cout << "Broj znakova, ukljucujuci prazna mesta "
		<< "u \"" << recenica << "\" je: "
		<< recenica.length() << endl;
	return 0;
}
```

```
Unesi poluprecnik sfere: 3

Zapremina sfere je: 113.098

Unesi koordinate dve tacke u X-Y povrsi: 4 7 9 -5

Rastojanje izmedju tacaka sa unetim koordinatama je: 13

Broj znakova, ukljucujuci prazna mesta u "Programiranje u C++" je: 19
Press any key to continue . . .
```

Узлазне функције

пример: ако је улаз : A 25 и хоћемо да ch1 прими знак A, да ch2 прими празно место, и да num прими број 25

```
char ch1, ch2;
int num;
cin >> ch1 >> ch2 >> num;
```

Код ће се извршити тако што је A ушло у ch1, празно место је прескочено од стране >> оператора екстракције, 2 је смештена у ch2 а 5 је ушло у num. Ако то нисмо хтели, закључујемо да >> се не може користити за овакве податке. Када је потребно процесуирати цео улаз укључујући и празна места и карактер нове линије, користи се функција **get**.

cin.get(varChar); // где је varChar промењива типа char и преставља аргумент (параметар) функције.

Ефекат је да се следећи улазни карактер смешта у промењиву varChar.

За правилно учитавање из претходног примера:

```
cin >> ch1;
cin.get(ch2);
cin >> num;
```

Када је потребно обрадити само део улазних података користи се ток функција **ignore** за одбацивање дела улаза:

cin.ignore(intExp, chExp); // intExp је цео број који показује број карактера у линији улаза који ће бити игнорисани од стране или док се не наиђе на карактер chExp у том опсегу карактера са улаза.

Пример:

```
int a, b;
cin >> a;
cin.ignore(100, '\n');
cin >> b;
```

ако је улаз 25 67 89
12 78 34

прва линија cin >> a смешта 25 у a, друга линија cin.ignore(100, '\n') игнорише све бројеве и празна места све док не наиђе карактер за нови ред, трећа линија смешта 12 у b.
Са **cin.ignore()**; се прескаче било који следећи карактер и најчешће се користи за прескакање карактера нове линије.
Када је потребно имати у улазном току и слова и бројеве треба правити доста кода за правилну конверзију.
У оваквим ситуацијама је олакшано коришћењем функција **putback** и **peek**: **cin.putback(ch); ch = cin.peek()**; где је ch промењива типа char.
Функција putback враћа последњи карактер који је извучен из улазног тока са get функцијом назад у улазни ток. Функција peek гледа који је следећи карактер у улазном току али га не извлачи из тока.
Ако је из било ког разлога дошло до прекида улазног тока, систем ће игнорисати сваки следећи улаз или излаз. Тада се користи функција **clear** за поправљање улазног тока: **cin.clear()**; најчешће се у следећој линији ставља cin.ignore(intExp, chExp); чиме се чисти улазни бафер од неконтролисаних података.

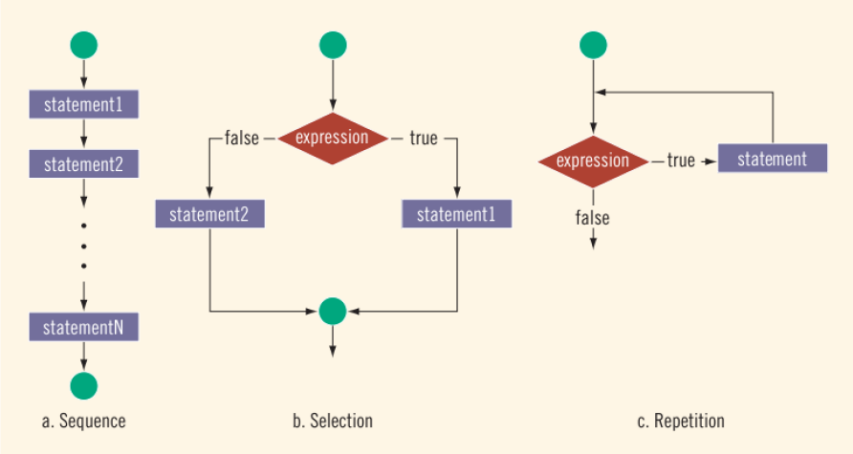
Излазни манипулатори

Користи се за контролу излаза бројева са покретном тачком.
Да би се одредило колико места после тачке хоћемо у бројевима користи се **setprecision** манипулатор:
setprecision (n) gde je n broj decimalnih mesta.
cout << setpreciosion (2); форматира излаз децималних бројева на два места иза тачке.
Prethodno treba uključiti heder fajl iomanip u kod : **#include<iomanip>.**
Ако хоћемо да се сви унети децимални бројеви појављују на излазу са истим форматом користи се **fixed** манипулатор: cout << fixed; што ће да важи све док се овај манипулатор не искључи: **cout.unsetf(ios::fixed)**; са **unsetf** манипулатором. После овога бројеви на излазу се појављују у подразумеваној поставци.
Манипулатор **scentific** се користи за излаз бројева у научном формату.
Ако је децимални део броја све нула и ако се треба показати децимални део у фиксном формату, излаз можда неће показати децималну тачку и децималне нуле. Да би се они сигурно показали користи се манипулатор **showpoint**.
cout << showpoint; pokazuje na standardnom izlazu decimalnu tačku i nule decimale
cout << fixed << showpoint; pokazuje u fiksnom decimalnom formatu sve

Излаз са стрингом

Cin се може користити за смештање string типа података у промењиве. Проблем је што унос са улаза се врши све док се не наиђе на празно место.
Да би се унело и празно место: **getline(cin, strVar)**; где је **strVar** промењива типа стринга.
Овакво читање је ограничено појавом \n. Функција getline чита све док не стигне крај тренутне линије.
Карактер \n се такође чита али се не смешта у string промењиву.
string myString; ово је ток са улаза: Hello there. How are you ?
getline(cin, myString);
Добија се:
mystring = " Hello there. How are you ?"

Селекција



Селекције се заснивају на упоређивању вредности путем релацијских оператора (==, !=, <, <=, >, >=) чиме се добија резултат true или false. Могу да се упоређују бројеви било којег типа, цели бројеви и знаковни тип (помоћу ASCII кода за сваки знак).
Селекција може бити једносмерна:
if (izraz) где израз даје резултат буловог типа. а исказ се извршава само ако је израз тачан.
  iskaz Ако се исказ састоји од више линија, оне морају бити у блоку {…}
Селекција може бити двосмерна:
if (izraz) ако је израз тачан извршава се исказ1, ако је израз нетачан извршава се исказ2
  iskaz1 Ако се исказ састоји од више линија, оне морају бити у блоку {…}
else
  iskaz2
При комбиновању више логичких израза користе се логички оператори (! не, && и, || или).
Упоређивање сртрингова се своди на упоређивање ASCII кодовне вредности првих карактера оба стринга па ако су истих вредности следећих итд.
Оператор услова
Тернарни оператор (?:) се назива оператор услова: **izraz1 ? izraz2 : izraz3**.
Ако је израз1 тачан или једнак вредности која није једнака 0, резултат израза1 је израз2 иначе резултат је израз3.
Пример: max = (a >= b) ? a : b; tj. if (a >= b) max = a; else max = b;
Структура switch

```
switch (izraz)
case vrednost1:   case vrednost2:   ...        default:
  iskaz1 break;      iskaz2 break;                 iskaz
```

while петља

```
while(izraz) iskaz              док је израз тачан извршаваће се исказ
// inicijalizacija promenjive za kontrolu petlje
while (izraz)       // izraz testira vrednost promenjive za kontrolu petlje
{
        …
        // update promenjivu za kontrolu petlje
        …
}
```

Постоје три форме while петље: контрола бројањем циклуса, контрола улазом и контрола флегом. Прва броји пролазак циклуса и реагује на достизање одређеног броја циклуса, друга реагује на унету вредност са улаза, трећа реагује на промену буловог логичког резултата.
for петља
Служи за поједностављење контроле бројањем циклуса: for(inicijalizacija; uslov petlje; update) iskaz
Петља: for(;;) је бесконачна јер је увек тачна
do...while петља
Ако желимо да се сигурно једном уђе у петљу без обзира на тачност услова користи се:

```
do
  iskaz
while (izaraz);
```

Команде break и continue
Команда **break** се користи за безусловни излазак из петље. На тај начин се прескаче остатак наредби унутар исказа у петљи.
Команда **continue** се користи за прекидање извршења циклуса са тренутним вредностима и безусловни почетак новог циклуса петље.