

TAKING | PRODUCTIONS

C++ : TALES FROM THE COMPILER

PRODUCTIONS BY TAKING

BASICS AND PRINCIPLES OF C++ PROGRAMMING
FROM THE BEGINNING

فهرست

آشنایی با زبان و ابزارها

صفحه.....	معرفی زبان و کاربرد آن.....
صفحه.....	ساخت و اجرای اولین برنامه.....
	مفاهیم پایه و سینتکس زبان

صفحه.....	متغیرها و انواع داده‌ها.....
صفحه.....	مقداردهی متغیرها و قواعد نام‌گذاری.....
صفحه.....	ورودی و خروجی.....
	عملگرها و عبارات

صفحه.....	عملگرها ریاضی.....
صفحه.....	عملگرها مقایسه‌ای.....
صفحه.....	عملگرها منطقی.....
صفحه.....	عملگرها انتساب.....
	ساختارهای شرطی و تصمیم‌گیری

صفحه.....	شرطها.....
صفحه.....	ساختار switch-case.....
	حلقه ها و کنترل جریان

صفحه.....	حلقه For.....
صفحه.....	حلقه While.....
صفحه.....	حلقه Do-while.....
صفحه.....	دستورات کنترل جریان.....

آرایه‌ها و رشته‌ها

صفحه.....	معرفی آرایه‌ها و کاربرد آن‌ها
صفحه.....	آرایه‌های یکبعدی
صفحه.....	آرایه‌های چندبعدی
صفحه.....	توابع داخلی کار با آرایه‌ها
صفحه.....	پردازش رشته‌ها

توابع

صفحه.....	تعریف و استفاده از توابع
صفحه.....	ارسال و دریافت مقادیر
صفحه.....	مقدار بازگشتی
صفحه.....	انواع متغیرهای سراسری و محلی

اشاره گرها و مدیریت حافظه

صفحه.....	تعریف و مقداردهی اشاره گرها
صفحه.....	عملیات روی اشاره گرها
صفحه.....	تخصیص و آزادسازی حافظه

ساختارها و یونیون‌ها

صفحه.....	تعریف و استفاده از ساختار
صفحه.....	تفاوت ساختار با کلاس در سی پلاس پلاس
صفحه.....	یونیون و کاربرد آن

کار با فایل‌ها

صفحه.....	خواندن و نوشتن در فایل‌ها
صفحه.....	دستکاری داده‌های فایل

معرفی زبان C++ و کاربردهای آن

۱. C++ چیست؟

یک زبان سطح میانی و چندمنظوره است که توسط بجرنه استرووستروپ (Bjarne Stroustrup) در اوایل دهه ۱۹۸۰ توسعه یافت. این زبان از C گرفته شده و ویژگی‌های برنامه‌نویسی شی‌گرا (OOP) را به آن اضافه کرده است. C++ یک زبان کامپایلری است، یعنی کد نوشته شده باید قبل از اجرا توسط یک کامپایلر (مثل MSVC, Clang, g++) یا به کد ماشین تبدیل شود. این کار باعث سرعت بالا و بهینه بودن اجرای برنامه‌های نوشته شده با C++ می‌شود.

۲. ویژگی‌های مهم C++

- ✓ سرعت بالا → چون مستقیماً به کد ماشین کامپایل می‌شود.
- ✓ چندمنظوره → برای برنامه‌نویسی سیستمی، اپلیکیشن، بازی، شبکه و... مناسب است.
- ✓ نزدیکی به سخت‌افزار → امکان مدیریت حافظه و بهینه‌سازی منابع.
- ✓ شی‌گرا (OOP) → پشتیبانی از کلاس‌ها و اشیاء برای مدیریت بهتر کد.
- ✓ قابل حمل (Portable) → روی سیستم‌عامل‌های مختلف مثل ویندوز، لینوکس و مک اجرا می‌شود.
- ✓ انعطاف‌پذیر و قدرتمند → ترکیب ویژگی‌های زبان سطح پایین (C) و زبان سطح بالا.

۳. چرا C++ یاد بگیریم؟

- ✓ بهینه و سریع برای پروژه‌های سنگین.
- ✓ درک بهتر از سخت‌افزار و مدیریت حافظه.
- ✓ یادگیری آسان‌تر زبان‌های دیگر مثل C# و Java.
- ✓ فرصت‌های شغلی زیاد در حوزه‌های مختلف.
- ✓ قدرت بالا در پردازش‌های پیچیده مثل بازی‌سازی و هوش مصنوعی.

♦ نتیجه‌گیری:

C++ یک زبان قدرتمند، سریع و پرکاربرد است که در بسیاری از صنایع استفاده می‌شود. اگر به برنامه‌نویسی حرفه‌ای و ساخت نرم‌افزارهای بهینه علاقه دارید، یادگیری C++ می‌تواند یک انتخاب عالی باشد! 

ساخت و اجرای اولین برنامه C++

حالا که کامپایلر را نصب کردیم و با متغیرها و انواع داده‌ها آشنا شدیم، بباید اولین برنامه خود را در C++ بنویسیم و اجرا کنیم.

۱. ساخت فایل برنامه

ابتدا یک فایل جدید ایجاد کنید و آن را با پسوند `.cpp` ذخیره کنید. مثلاً نام آن را `first_program.cpp` بگذارید.

۲. نوشتن اولین برنامه ("Hello, World")

```
#include <iostream> // کتابخانه برای ورودی و خروجی
using namespace std; // استفاده از فضای نام استاندارد

int main() {
    cout << "Hello, World!" << endl; // چاپ متن روی صفحه
    return 0; // بیان موفقیت‌آمیز برنامه
}
```

توضیح کد:

→ این خط کتابخانه‌ای را اضافه می‌کند که برای ورودی و خروجی (`cout` و `cin`) استفاده می‌شود.

→ این خط باعث می‌شود که نیاز به نوشتن `std::cout` نداشته باشیم.

→ این تابع نقطه شروع اجرای برنامه در C++ است.

→ این دستور متن `"Hello, World"` را روی خروجی نمایش می‌دهد.

→ مقدار `0` به سیستم برمنی‌گرداند که نشان‌دهنده اجرای موفق برنامه است.

۳. متغیرها و انواع داده‌ها در C++

۱. متغیر چیست؟

متغیر (Variable) فضایی در حافظه است که می‌توان مقدار آن را ذخیره و در طول برنامه تغییر داد. هر متغیر یک نام و یک نوع داده دارد که مشخص می‌کند چه مقدار حافظه اشغال کند و چه نوع مقداری را نگه دارد.

۵. نحوه تعریف متغیر در C++

در C++ متغیرها به صورت زیر تعریف می‌شوند:

```
;نوع_داده نام_متغیر = مقدار_ولیه
```

Edit ⌂ Copy ☉

مثال:

```
int age = 25;           // متغیر عدد صحیح
double pi = 3.14;       // متغیر اعشاری
char grade = 'A';       // متغیر کاراکتر
bool isOn = true;        // متغیر بولین
```

Edit ⌂ Copy ☉

۶. انواع داده‌ها در C++

C++ دارای انواع مختلف داده‌ای است که به دو دسته اصلی (Primary) و مشتق شده (Derived) تقسیم می‌شوند:

داده‌های اصلی (Primary Data Types)

مقدار قابل ذخیره	اندازه (بايت)	نوع داده
عدد صحیح ($\pm 2^{31}$ میلیارد)	۴ بايت	int
عدد اعشاری (دقیق ۷-۶ رقم)	۴ بايت	float
عدد اعشاری دقیق‌تر (۱۰-۱۶ رقم)	۸ بايت	double
یک کاراکتر (مثل 'A')	۱ بايت	char
false یا true	۱ بايت	bool
بدون مقدار (مثلاً در توابع)	-	void

داده‌های مشتق شده (Derived Data Types)

این داده‌ها از انواع اصلی ساخته شده‌اند، مثل آرایه‌ها، اشاره‌گرها و ساختارها. بعداً به آن‌ها می‌پردازیم!

۷. مقداردهی متغیرها (Variable Initialization)

می‌توان متغیرها را به روش‌های مختلف مقداردهی کرد:

Edit ⌂ Copy ⌂

```
int a = 10;           // روش معمولی
int b(20);           // مقداردهی با پرانتز
int c{30};           // ` { } ` مقداردهی با (C++11)
```

۵. محدوده (Scope) و طول عمر متغیرها

- متغیرهای محلی (Local Variables): درون یک تابع یا بلاک تعریف شده و فقط در همان قسمت قابل استفاده هستند.
- متغیرهای سراسری (Global Variables): بیرون از همه توابع تعریف شده و در کل برنامه در دسترس اند.

مثال:

Edit ⌂ Copy ⌂

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

int globalVar = 100; // متغیر سراسری

int main() {
    int localVar = 50; // متغیر محلی
    cout << globalVar << " " << localVar;
    return 0;
}
```

۶. تبدیل نوع داده (Type Casting)

C++ اجازه می‌دهد نوع داده‌ها را به یکدیگر تبدیل کنیم:

- ♦ تبدیل ضمنی (Implicit Casting) → وقتی C++ خودش تبدیل را انجام دهد:

Edit ⌂ Copy ⌂

```
int x = 5;
double y = x; // تبدیل خودکار int به double
```

♦ تبدیل صریح (Explicit Casting) → وقتی خودمان مشخص کنیم:

Edit ⌛ Copy ⌂

```
double pi = 3.14;  
int num = (int)pi; // تبدیل دستی double & int
```

۷. ثابت‌ها (Constants) در C++

برای ایجاد متغیرهایی که مقدارشان تغییر نکند، از `const` استفاده می‌کنیم:

Edit ⌛ Copy ⌂

```
const double PI = 3.1416;  
PI = 3.2; // خطأ! مقدار ثابت قابل تغییر نیست
```

نتیجه‌گیری

- ✓ متغیرها در C++ برای ذخیره داده‌ها استفاده می‌شوند.
- ✓ انواع داده‌ای مختلف برای اعداد صحیح، اعشاری، کاراکترها و مقادیر منطقی وجود دارند.
- ✓ متغیرها دارای محدوده و طول عمر مشخصی هستند.
- ✓ از `const` برای تعریف متغیرهای ثابت استفاده می‌شود.

۸. مقداردهی متغیرها و قواعد نامگذاری در C++

۱. مقداردهی متغیرها در C++

مقداردهی یعنی اختصاص مقدار اولیه به متغیر. این کار را می‌توان هنگام تعریف متغیر یا در مراحل بعدی انجام داد.

روش‌های مقداردهی متغیرها

۱. مقداردهی هنگام تعریف متغیر (Initialization)

می‌توان هنگام تعریف متغیر مقدار اولیه به آن داد:

Edit ⌛ Copy ⌂

```
int age = 25; // مقداردهی مستقیم  
double pi = 3.14; // مقداردهی عدد اعشاری  
char grade = 'A'; // مقداردهی کاراکتر  
bool isHappy = true; // مقداردهی مقدار منطقی
```

۵. مقداردهی با پرانتز (C++ Style)

Edit ⌂ Copy ☉

```
int age(25);
double pi(3.14);
char grade('A');
```

۳. مقداردهی با آکولاد (Modern C++ - C++11)

این روش از خطاهای احتمالی جلوگیری می‌کند:

Edit ⌂ Copy ☉

```
int age{25};
double pi{3.14};
char grade{'A'};
```

اگر مقدار نامعتبر بدهید، C++ خطا می‌دهد!

Edit ⌂ Copy ☉

```
int number{3.14}; // ✗ مجاز نیست int خطا! مقدار اعشاری برای
```

۴. مقداردهی بعد از تعریف متغیر

می‌توان متغیر را بدون مقدار اولیه تعریف و بعداً مقداردهی کرد:

Edit ⌂ Copy ☉

```
int number;
number = 10; // مقداردهی بعد از تعریف
```

۵. مقداردهی چند متغیر همزمان

Edit ⌂ Copy ☉

```
int a = 10, b = 20, c = 30; // مقداردهی چند متغیر
```

```
int x, y, z;
x = y = z = 5; // چند متغیر
```

مقداردهی همزمان به چند متغیر

Edit ⌂ Copy ⌂

۳. قواعد نام‌گذاری متغیرها در C++

مجاز است: ✓

- ✓ نام متغیر باید با یک حرف (a-z, A-Z) یا شروع شود.
- ✓ می‌تواند شامل اعداد (0-9) باشد (اما با عدد شروع نشود).
- ✓ ++C به بزرگی و کوچکی حروف حساس است (Age و age دو متغیر متفاوت هستند).
- ✓ بهتر است نام متغیر گویا و مرتب با مقدار آن باشد.

غیرمجاز است: ✗

- ✗ شروع با عدد (1var)
- ✗ استفاده از فاصله (my variable)
- ✗ استفاده از کاراکترهای خاص مثل @, \$, %
- ✗ استفاده از کلمات کلیدی زبان C (int, float, return) و (...)

۴. سبک‌های نام‌گذاری در C++

سبک نام‌گذاری	مثال	توضیح
snake_case	student_age	محبوب در C و پایگاه‌های داده
camelCase	studentAge	محبوب در C++ و جاوا
PascalCase	StudentAge	معمولآً برای نام کلاس‌ها استفاده می‌شود
kebab-case	student-age	در C++ غیرمجاز است!

پیشنهاد: از snake_case یا camelCase برای متغیرها استفاده کنید. ✓

۵. متغیرهای ثابت (const)

برای متغیرهایی که مقدارشان تغییر نکند از const استفاده کنید:

```
const double PI = 3.1416; // مقدار ثابت
PI = 3.2; // خطأ! مقدار ثابت قابل تغییر نیست ✗
```

Edit ⌂ Copy ⌂

نتیجه‌گیری

- ✓ چند روش مختلف مقداردهی را یاد گرفتیم.
- ✓ قواعد نام‌گذاری متغیرها را بررسی کردیم.
- ✓ ثابت‌ها (const) و سبک‌های نام‌گذاری را یاد گرفتیم.

ورودی و خروجی در C++

C++ برای تعامل با کاربر از ورودی (input) و خروجی (output) استفاده می‌کند. برای این کار، از `cin` برای ورودی و `cout` برای خروجی استفاده می‌کنیم.

۱. نمایش خروجی با cout

از کتابخانه `iostream` برای چاپ داده روی صفحه نمایش استفاده می‌کند.

نحوه استفاده از cout ✓

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

int main() {
    cout << "Hello, World!" << endl; // جاب متن و رفتن به خط بعد
    cout << "Welcome to C++ Programming!";
    return 0;
}
```

Edit ⌂ Copy ☉

- ♦ باعث می‌شود که خروجی به خط جدید برود.
- ♦ به جای `\n` می‌توان از `endl` استفاده کرد:

```
cout << "Hello, World!\n";
```

Edit ⌂ Copy ☉

خروجی برنامه: ✓

```
Hello, World!
Welcome to C++ Programming!
```

Edit ⌂ Copy ☉

۲. دریافت ورودی با `cin`

برای گرفتن مقدار از کاربر از `cin` استفاده می‌کنیم.

گرفتن ورودی از کاربر ✓

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

int main() {
    int age;
    cout << "Enter your age: ";
    cin >> age; // دریافت مقدار از کاربر
    cout << "You are " << age << " years old." << endl;
    return 0;
}
```

Edit ⌂ Copy ⌂

مثال ورودی و خروجی: ✓

```
Enter your age: 25
You are 25 years old.
```

Edit ⌂ Copy ⌂

۳. گرفتن چند ورودی از کاربر

می‌توان با یک `cin` چند مقدار از کاربر گرفت:

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

int main() {
    int x, y;
    cout << "Enter two numbers: ";
    cin >> x >> y;
    cout << "Sum: " << (x + y) << endl;
    return 0;
}
```

Edit ⌂ Copy ⌂

خروجی:

Edit ⌛ Copy ⌂

Enter two numbers: 5 10

Sum: 15

۴. گرفتن ورودی رشته‌ای (String)

برای ورودی‌های رشته‌ای تا اولین فاصله را می‌خواند.

مشکل cin با string ◆

Edit ⌛ Copy ⌂

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

int main() {
    string name;
    cout << "Enter your name: ";
    cin >> name; // فقط اولین کلمه را می‌خواند
    cout << "Hello, " << name << "!" << endl;
    return 0;
}
```

ورودی و خروجی:

Edit ⌛ Copy ⌂

Enter your name: Ali Reza

Hello, Ali!

اما cin را می‌خواند و Reza نادیده گرفته می‌شود!!

حل مشکل با getline ◆

برای خواندن کل خط از getline(cin, variable) استفاده کنید:

Edit ⌛ Copy ⌂

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

```

int main() {
    string fullName;
    cout << "Enter your full name: ";
    cin.ignore(); // در برخی موارد ضروری است
    getline(cin, fullName);
    cout << "Hello, " << fullName << "!" << endl;
    return 0;
}

```

وروودی و خروجی:

Edit ⌛ Copy ⌂

```

Enter your full name: Ali Reza Mohammadi
Hello, Ali Reza Mohammadi!

```

۵. دریافت عدد اعشاری (**double** و **float**)

می‌تواند عدد اعشاری را دریافت کند: `cin`

Edit ⌛ Copy ⌂

```

#include <iostream>
using namespace std;

int main() {
    double price;
    cout << "Enter product price: ";
    cin >> price;
    cout << "Price: " << price << endl;
    return 0;
}

```

وروودی و خروجی:

Edit ⌛ Copy ⌂

```

Enter product price: 19.99
Price: 19.99

```

!! نکته: در بعضی زبان‌های محلی، علامت جداگننده اعشار ممکن است `,` باشد، اما `cin` فقط `.` را قبول می‌کند.

۶. ترکیب cout و cin با انواع داده‌های مختلف

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

int main() {
    int age;
    double height;
    char gender;
    bool isStudent;

    cout << "Enter your age: ";
    cin >> age;

    cout << "Enter your height (in meters): ";
    cin >> height;

    cout << "Enter your gender (M/F): ";
    cin >> gender;

    cout << "Are you a student? (1 for Yes, 0 for No): ";
    cin >> isStudent;

    cout << "\nProfile:\n";
    cout << "Age: " << age << "\nHeight: " << height << "\nGender: " << gender;
    cout << "\nstudent: " << (isStudent ? "Yes" : "No") << endl;

    return 0;
}
```

نمونه ورودی و خروجی:

```
Enter your age: 21
Enter your height (in meters): 1.75
Enter your gender (M/F): M
Are you a student? (1 for Yes, 0 for No): 1
```

Profile:

Age: 21
Height: 1.75m
Gender: M
Student: Yes

نتیجه‌گیری

برای نمایش خروجی و `cin` برای گرفتن ورودی استفاده می‌شود.
برای گرفتن ورودی رشته‌ای با فاصله به کار می‌رود.
`getline(cin, variable)` می‌تواند اعداد صحیح، اعشاری، کاراکتر و مقدار منطقی بگیرد.
برای حل مشکل خواندن ناقص ورودی استفاده می‌شود.
`(cin.ignore ())`

عملگرهای ریاضی در C++

در C++ از عملگرهای ریاضی برای انجام محاسبات مختلف استفاده می‌شود. این عملگرها شامل جمع (+)، تفریق (-)، ضرب (*)، تقسیم (/) و باقیمانده (%) هستند.

۱. جمع (+)

عملگر جمع برای اضافه کردن دو عدد به کار می‌رود.

مثال:

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

int main() {
    int a = 10, b = 5;
    int sum = a + b;    // جمع دو عدد
    cout << "Sum: " << sum << endl;    // نمایش نتیجه
    return 0;
}
```

Edit ⚙ Copy ☉

خروجی:

```
Sum: 15
```

Edit ⚙ Copy ☉

۵. تفریق (-)

عملگر تفریق برای کم کردن یک عدد از عدد دیگر استفاده می‌شود.

مثال:

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

int main() {
    int a = 10, b = 5;
    int difference = a - b; // تفریق دو عدد
    cout << "Difference: " << difference << endl; // نمایش نتیجه
    return 0;
}
```

Edit ⚙ Copy ☉

خروجی:

```
Difference: 5
```

Edit ⚙ Copy ☉

۶. ضرب (*)

عملگر ضرب برای ضرب دو عدد استفاده می‌شود.

مثال:

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

int main() {
    int a = 10, b = 5;
    int product = a * b; // ضرب دو عدد
    cout << "Product: " << product << endl; // نمایش نتیجه
    return 0;
}
```

Edit ⚙ Copy ☉

خروجی:

Edit ⌛ Copy ⌂

Product: 50

۵. تقسیم (/)

عملگر تقسیم برای تقسیم یک عدد بر عدد دیگر استفاده می‌شود.

- اگر دو عدد صحیح تقسیم شوند، نتیجه نیز صحیح خواهد بود (یعنی قسمت اعشاری حذف می‌شود).
- برای دریافت قسمت اعشاری، باید از نوع داده‌ی `double` یا `float` استفاده کنید.

مثال با اعداد صحیح:

Edit ⌛ Copy ⌂

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

int main() {
    int a = 10, b = 4;
    int quotient = a / b; // تقسیم دو عدد صحیح
    cout << "Quotient: " << quotient << endl; // نمایش نتیجه
    return 0;
}
```

خروجی:

Edit ⌛ Copy ⌂

Quotient: 2 // قسمت اعشاری حذف می‌شود

مثال با اعداد اعشاری:

Edit ⌛ Copy ⌂

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

```

int main() {
    double a = 10, b = 4;
    double quotient = a / b; // تقسیم دو عدد اعشاری
    cout << "Quotient: " << quotient << endl; // نمایش نتیجه
    return 0;
}

```

خروجی:

Edit ⚙ Copy ☉

Quotient: 2.5

۵. باقیمانده (%)

عملگر باقیمانده یا مدول برای گرفتن باقیمانده تقسیم یک عدد بر عدد دیگر استفاده می‌شود. این عملگر معمولاً برای بررسی زوج یا فرد بودن اعداد، تقسیم مساوی یا کارهای مشابه استفاده می‌شود.

مثال:

```

#include <iostream>
using namespace std;

int main() {
    int a = 10, b = 3;
    int remainder = a % b; // باقیمانده تقسیم
    cout << "Remainder: " << remainder << endl; // نمایش نتیجه
    return 0;
}

```

خروجی:

Edit ⚙ Copy ☉

Remainder: 1

در این مثال، باقیمانده تقسیم 10 بر 3 برابر 1 است.

۶. نکات مهم:

- در تقسیم صحیح (int تقسیم int)، قسمت اعشاری حذف می‌شود. اگر نیاز به دقت بیشتر دارید، باید از نوع داده double یا float استفاده کنید.
- عملگر % تنها برای اعداد صحیح قابل استفاده است.
- در استفاده از عملگرهای ریاضی، اطمینان حاصل کنید که بر صفر تقسیم نکنید چون این عمل باعث خطا در اجرای برنامه می‌شود.

عملگرهای مقایسه‌ای در C++

عملگرهای مقایسه‌ای در C++ برای مقایسه دو مقدار و بررسی رابطه آن‌ها استفاده می‌شوند. این عملگرها نتیجه‌ای از نوع bool (درست یا نادرست) به عنوان خروجی خواهند داشت. در C++, عملگرهای مقایسه‌ای به شرح زیر هستند:

۱. برابر با (==)

این عملگر برای بررسی این‌که آیا دو مقدار برابر هستند یا نه، استفاده می‌شود.

مثال:

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

int main() {
    int a = 10, b = 10;
    if (a == b) {
        cout << "a is equal to b" << endl;
    } else {
        cout << "a is not equal to b" << endl;
    }
    return 0;
}
```

Edit ⌚ Copy ⌂

خروجی:

```
a is equal to b
```

Edit ⌚ Copy ⌂

۳. نابرابر با ($=!$)

این عملگر برای بررسی اینکه آیا دو مقدار برابر نیستند، استفاده می‌شود.

مثال:

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

int main() {
    int a = 10, b = 5;
    if (a != b) {
        cout << "a is not equal to b" << endl;
    } else {
        cout << "a is equal to b" << endl;
    }
    return 0;
}
```

Edit ⚙ Copy ☉

خروجی:

```
a is not equal to b
```

Edit ⚙ Copy ☉

۴. بزرگتر از ($<$)

این عملگر برای بررسی اینکه آیا مقدار اول بزرگتر از مقدار دوم است یا نه، استفاده می‌شود.

مثال:

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

int main() {
    int a = 10, b = 5;
    if (a > b) {
        cout << "a is greater than b" << endl;
    }
}
```

Edit ⚙ Copy ☉

```

} else {
    cout << "a is not greater than b" << endl;
}
return 0;
}

```

خروجی:

Edit ⌂ Copy ☉

a is greater than b

۵. کوچکتر از (>)

این عملگر برای بررسی اینکه آیا مقدار اول کوچکتر از مقدار دوم است یا نه، استفاده می‌شود.

مثال:

```

#include <iostream>
using namespace std;

int main() {
    int a = 5, b = 10;
    if (a < b) {
        cout << "a is less than b" << endl;
    } else {
        cout << "a is not less than b" << endl;
    }
    return 0;
}

```

خروجی:

Edit ⌂ Copy ☉

a is less than b

۶. بزرگتر یا مساوی (=>)

این عملگر برای بررسی اینکه آیا مقدار اول بزرگتر یا مساوی مقدار دوم است، استفاده می‌شود.

:مثال

Edit ⚙ Copy ☉

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

int main() {
    int a = 10, b = 5;
    if (a >= b) {
        cout << "a is greater than or equal to b" << endl;
    } else {
        cout << "a is less than b" << endl;
    }
    return 0;
}
```

خروجی:

Edit ⚙ Copy ☉

```
a is greater than or equal to b
```

۶. کوچکتر یا مساوی (=>)

این عملگر برای بررسی اینکه آیا مقدار اول کوچکتر یا مساوی مقدار دوم است، استفاده می‌شود.

:مثال

Edit ⚙ Copy ☉

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

int main() {
    int a = 5, b = 10;
    if (a <= b) {
        cout << "a is less than or equal to b" << endl;
    } else {
        cout << "a is greater than b" << endl;
    }
    return 0;
}
```

خروجی:

```
a is less than or equal to b
```

Edit ⌂ Copy ⌂

نکات مهم:

- برای مقایسه برابری دو مقدار استفاده می‌شود، نه برای انتساب (که برای انتساب باید از `=` استفاده کرد).
- عملگرهای مقایسه‌ای همیشه `true` یا `false` را برمی‌گردانند. در `C++`، مقادیر `true` به‌طور داخلی معادل ۱ و مقادیر `false` معادل ۰ هستند.
- برای استفاده از این عملگرهای مقایسه‌ها معمولاً در حلقه‌ها یا شرط‌ها استفاده می‌شوند تا تصمیم‌گیری و کنترل جریان برنامه انجام شود.

عملگرهای منطقی در `C++`

عملگرهای منطقی در `C++` برای انجام عملیات منطقی روی عبارات بولی (که نتایج آن‌ها `true` یا `false` است) استفاده می‌شوند. این عملگرهای بیشتر برای کنترل جریان برنامه، به ویژه در شرایط شرطی و حلقه‌ها، به کار می‌روند.

عملگرهای منطقی در `C++`

1. AND منطقی (`&&`)

2. OR منطقی (`||`)

3. NOT منطقی (`!`)

1. AND منطقی (`&&`)

عملگر `&&` زمانی `true` است که هر دو عبارت در سمت چپ و راست آن `true` باشند. اگر یکی از آن‌ها `false` باشد، نتیجه `false` خواهد بود.

مثال:

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

int main() {
    bool a = true, b = false;
```

Edit ⌂ Copy ⌂

```

if (a && b) {
    cout << "Both are true" << endl;
} else {
    cout << "At least one is false" << endl;
}
return 0;
}

```

خروجی:

Edit ⌐ Copy ☉

At least one is false

در این مثال، چون `a` برابر با `true` است و `b` برابر با `false`، نتیجه نهایی `false` است و پیغام "At least one is false" چاپ می‌شود.

۵. OR منطقی (||)

عملگر `||` زمانی `true` است که حداقل یکی از عبارت‌های سمت چپ یا راست آن `true` باشد. اگر هر دو `false` باشند، نتیجه `false` خواهد بود.

مثال:

Edit ⌐ Copy ☉

```

#include <iostream>
using namespace std;

int main() {
    bool a = true, b = false;

    if (a || b) {
        cout << "At least one is true" << endl;
    } else {
        cout << "Both are false" << endl;
    }
    return 0;
}

```

خروجی:

Edit ⌂ Copy ☉

At least one is true

در این مثال، چون `a` برابر با `true` است و `b` برابر با `false` است، نتیجه `true` می‌شود و پیغام "true" چاپ می‌شود.

۳. NOT منطقی (!)

عملگر `!` یا NOT زمانی `true` است که عبارت ورودی آن `false` باشد و بالعکس. این عملگر برای معکوس کردن مقدار یک عبارت بولی استفاده می‌شود.

مثال:

Edit ⌂ Copy ☉

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

int main() {
    bool a = true;

    if (!a) {
        cout << "a is false" << endl;
    } else {
        cout << "a is true" << endl;
    }
    return 0;
}
```

خروجی:

Edit ⌂ Copy ☉

a is true

در این مثال، چون `a` برابر با `true` است، نتیجه `!a` برابر با `false` می‌شود، اما چون در عبارت `else` قرار می‌گیرد، پیغام "a is true" چاپ می‌شود.

نکات مهم در مورد عملگرهای منطقی:

- عملگر AND (&&): این عملگر زمانی نتیجه `true` می‌دهد که هر دو عبارت `true` باشند.
- عملگر OR (||): این عملگر زمانی نتیجه `true` می‌دهد که حداقل یکی از عبارت‌ها `true` باشد.
- عملگر NOT (!): این عملگر معکوس کردن مقدار منطقی یک عبارت را انجام می‌دهد. به این معنا که اگر عبارت باشد، آن را به `false` تبدیل می‌کند و برعکس.

تمرین‌های پیشنهادی با عملگرهای منطقی:

- بررسی سن مجاز برای ورود به سایت: برنامه‌ای بنویسید که بررسی کند آیا یک فرد می‌تواند وارد سایت خاصی شود یا خیر. ورودی برنامه باید شامل سن فرد و اطلاعات تایید هویت باشد. اگر سن فرد بیشتر از ۱۸ سال و تایید هویت معتبر باشد، اجازه ورود صادر شود.
- بررسی عضو یا غیر عضو بودن کاربر: برنامه‌ای بنویسید که اگر کاربر عضو سایت باشد و از رمز عبور درست استفاده کند، پیغام خوش‌آمدگویی چاپ شود. در غیر این صورت، پیغام خطا نشان داده شود.
- بررسی وضعیت پرواز: برنامه‌ای بنویسید که وضعیت پرواز را بررسی کند. اگر وضعیت پرواز "آزاد" باشد و ظرفیت خالی باشد، اجازه رزرو بلیط صادر شود.
- عدد مثبت و فرد: برنامه‌ای بنویسید که بررسی کند آیا یک عدد وارد شده از کاربر هم مثبت است و هم فرد.
- چک کردن برای عضویت در باشگاه: برنامه‌ای بنویسید که بررسی کند یک نفر عضو باشگاه است یا خیر و آیا هزینه عضویت را پرداخت کرده است یا نه. اگر هر دو شرایط درست باشد، اجازه دسترسی به امکانات باشگاه داده شود.

جمع‌بندی

- عملگرهای منطقی `&&`، `||` و `!` ابزارهای بسیار مفیدی برای انجام عملیات منطقی در برنامه‌نویسی هستند. آن‌ها در بسیاری از موارد، به ویژه در شرط‌ها و حلقه‌ها، به شما کمک می‌کنند تا تصمیم‌گیری‌های پیچیده‌تری در برنامه خود داشته باشید.

عملگرهای انتساب در C++

عملگرهای انتساب در C++ برای اختصاص دادن مقدار به متغیرها استفاده می‌شوند. این عملگرها برای تغییر مقدار یک متغیر به کار می‌روند و به طور کلی نتایج متفاوتی می‌توانند داشته باشند. در C++, مهم‌ترین عملگرهای انتساب به شرح زیر هستند:

۱. انتساب ساده (=)

عملگر `=` برای انتساب مقدار یک متغیر به متغیر دیگر استفاده می‌شود. این عملگر مقدار موجود در سمت راست را به متغیر سمت چپ اختصاص می‌دهد.

مثال:

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

int main() {
    int a = 10; // a اخنهاص مقدار 10 به متغير
    int b = 5; // b اخنهاص مقدار 5 به متغير
    b = a; // b انتساب مقدار a
    cout << "b: " << b << endl; // چاپ مقدار b
    return 0;
}
```

Edit ⌛ Copy ⌂

خروجی:

```
b: 10
```

Edit ⌛ Copy ⌂

۳. انتساب ترکیبی (=+)

عملگر `=+` برای اضافه کردن یک مقدار به مقدار فعلی متغیر استفاده می‌شود. این عملگر به‌طور خودکار مقدار سمت راست را به مقدار سمت چپ اضافه کرده و نتیجه را در همان متغیر ذخیره می‌کند.

مثال:

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

int main() {
    int a = 10;
    a += 5; // a = a + 5 معادل a = 15 است
    cout << "a: " << a << endl; // چاپ مقدار a
}
```

Edit ⌛ Copy ⌂

```
    return 0;  
}
```

خروجی:

Edit ⌂ Copy ⌂

a: 15

۳. انتساب ترکیبی (-)

عملگر `=-` برای کم کردن یک مقدار از مقدار فعلی متغیر استفاده می‌شود. این عملگر به‌طور خودکار مقدار سمت راست را از مقدار سمت چپ کم کرده و نتیجه را در همان متغیر ذخیره می‌کند.

مثال:

```
#include <iostream>  
using namespace std;  
  
int main() {  
    int a = 10;  
  
    a -= 3; // حالا برابر 7 است a = a - 3 معادل (تبیین می‌کند و a جای مقدار a)  
  
    cout << "a: " << a << endl; //  
  
    return 0;  
}
```

خروجی:

Edit ⌂ Copy ⌂

a: 7

۴. انتساب ترکیبی (*=)

عملگر `=*` برای ضرب کردن مقدار فعلی متغیر با یک مقدار مشخص استفاده می‌شود. این عملگر به‌طور خودکار مقدار سمت راست را در مقدار سمت چپ ضرب کرده و نتیجه را در همان متغیر ذخیره می‌کند.

مثال:

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

int main() {
    int a = 10;

    a *= 2; // a = a * 2 (تغییر می‌کند و حالا برابر 20 است)

    cout << "a: " << a << endl; // جواب مقدار a

    return 0;
}
```

خروجی:

```
a: 20
```

۵. انتساب ترکیبی (/=)

عملگر /= برای تقسیم مقدار فعلی متغیر بر یک مقدار مشخص استفاده می‌شود. این عملگر به‌طور خودکار مقدار سمت راست را از مقدار سمت چپ تقسیم کرده و نتیجه را در همان متغیر ذخیره می‌کند.

مثال:

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

int main() {
    int a = 20;

    a /= 4; // a = a / 4 (تغییر می‌کند و حالا برابر 5 است)
}
```

```

cout << "a: " << a << endl; // چاپ مقدار a

return 0;
}

```

خروجی:

Edit ⌂ Copy ☉

a: 5

۶. انتساب ترکیبی (=%)

عملگر `=%` برای محاسبه باقیمانده تقسیم مقدار فعلی متغیر بر یک مقدار مشخص استفاده می‌شود. این عملگر به‌طور خودکار باقیمانده تقسیم مقدار سمت چپ بر مقدار سمت راست را محاسبه کرده و نتیجه را در همان متغیر ذخیره می‌کند.

مثال:

```

#include <iostream>
using namespace std;

int main() {
    int a = 10;

    a %= 3; // حالا برابر 1 است a = a % 3 معادل (تغییر می‌کند و a = 1 است)

    cout << "a: " << a << endl; // چاپ مقدار a

    return 0;
}

```

خروجی:

Edit ⌂ Copy ☉

a: 1

۷. انتساب ترکیبی ($=^$, $=||$, $=\&\&$)

در C++ می‌توان از عملگرهای منطقی ترکیبی برای انجام عملیات منطقی نیز استفاده کرد. این عملگرها مشابه عملگرهای انتساب ترکیبی ریاضی عمل می‌کنند، اما با مقایسه مقادیر بولی.

مثال:

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

int main() {
    bool a = true, b = false;

    a &&= b; // معادل a = a && b (حالا a برابر false می‌شود)

    cout << "a: " << a << endl; // چاپ مقدار a

    return 0;
}
```

Edit ⚙ Copy ⌂

خروجی:

```
a: 0
```

Edit ⚙ Copy ⌂

نکات مهم:

- عملگر $=$ فقط برای انتساب استفاده می‌شود. برای مقایسه دو مقدار از $==$ استفاده کنید.
- عملگرهای انتساب ترکیبی مانند $=+$, $=-$, $=*$ و غیره بسیار مفید هستند زیرا نیازی به نوشتن مجدد نام متغیر در سمت چپ نیست.
- در عملگرهای انتساب ترکیبی، تغییرات به طور خودکار روی متغیر اصلی اعمال می‌شود.

C++ (if, if-else, else if) شرط‌ها در

در C++, از دستورات شرطی برای کنترل جریان اجرای برنامه استفاده می‌شود. این دستورات به ما این امکان را می‌دهند که بر اساس شرایط مشخص، مسیر اجرای برنامه را تغییر دهیم.

۱. دستور if

ساختار پایه‌ای دستور if به شکل زیر است:

```
if (شرط) {
    کدهایی که اجرا می‌شوند اگر شرط درست باشد //
}
```

Edit ⌂ Copy ⌂

اگر شرط داخل پرانتز true باشد، دستورات داخل {} اجرا می‌شوند. در غیر این صورت، این دستورات نادیده گرفته می‌شوند.

مثال ۱: بررسی عدد مثبت

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

int main() {
    int num;
    cout << "یک عدد وارد کنید";
    cin >> num;

    if (num > 0) {
        cout << ". عدد مثبت است" << endl;
    }

    return 0;
}
```

Edit ⌂ Copy ⌂

وروعدی:

5

Edit ⌂ Copy ⌂

خروجی:

Edit ⌛ Copy ⌂

عدد مثبت است.

اگر عدد منفی یا صفر باشد، هیچ خروجی‌ای چاپ نخواهد شد.

۲. دستور if-else

امکان اجرای یک بلاک کد جایگزین را در صورت نادرست بودن شرط فراهم می‌کند.

Edit ⌛ Copy ⌂

```
if (شرط) {  
    این کد اجرا می‌شود اگر شرط درست باشد //  
} else {  
    این کد اجرا می‌شود اگر شرط غلط باشد //  
}
```

مثال ۲: بررسی عدد مثبت یا منفی

Edit ⌛ Copy ⌂

```
#include <iostream>  
using namespace std;  
  
int main() {  
    int num;  
    cout << "یک عدد وارد کنید: ";  
    cin >> num;  
  
    if (num >= 0) {  
        cout << ". عدد مثبت یا صفر است" << endl;  
    } else {  
        cout << ". عدد منفی است" << endl;  
    }  
  
    return 0;  
}
```

وروودی:

-3

Edit ⌂ Copy ⌂

خروجی:

عدد منفی است.

Edit ⌂ Copy ⌂

۳. دستور if else (شرط چندگانه)

زمانی که بیش از دو حالت داریم، از if برای اضافه کردن شرایط اضافی استفاده می‌کنیم.

```
if (شرط 1) {  
    اگر شرط 1 درست باشد، این کد اجرا می‌شود //  
} else if (شرط 2) {  
    اگر شرط 1 غلط باشد ولی شرط 2 درست باشد، این کد اجرا می‌شود //  
} else {  
    اگر هیچ‌کدام از شرایط بالا درست نبود، این کد اجرا می‌شود //  
}
```

Edit ⌂ Copy ⌂

مثال ۳: بررسی مثبت، منفی یا صفر بودن عدد

```
#include <iostream>  
using namespace std;  
  
int main() {  
    int num;  
    cout << "بک عدد وارد کنید";  
    cin >> num;
```

Edit ⌂ Copy ⌂

```

if (num > 0) {
    cout << "عدد مثبت است" << endl;
} else if (num < 0) {
    cout << "عدد منفی است" << endl;
} else {
    cout << "عدد صفر است" << endl;
}

return 0;
}

```

ورودی :

7

Edit ⌚ Copy ☉

خروجی:

عدد مثبت است.

Edit ⌚ Copy ☉

۴. استفاده از چند شرط با عملگرهای منطقی

گاهی اوقات نیاز داریم که چندین شرط را ترکیب کنیم. این کار با استفاده از عملگرهای منطقی (`&&` و `||`) انجام می‌شود:

مثال ۴: بررسی محدوده عدد

برنامه‌ای بنویسید که بررسی کند آیا عدد ورودی بین 10 تا 50 است یا نه.

```

#include <iostream>
using namespace std;

int main() {
    int num;
    cout << "یک عدد وارد کنید" << " ";
    cin >> num;

    if (num >= 10 && num <= 50) {
        cout << "عدد بین 10 و 50 است" << endl;
    }
}

```

```
    } else {
        cout << "عدد خارج از محدوده است" << endl;
    }

    return 0;
}
```

وروودی:

25

Edit ⌐ Copy ☉

خروجی:

عدد بین 10 و 50 است.

Edit ⌐ Copy ☉

۵. استفاده از شرط‌های تو در تو (**Nested If**)

در بعضی موارد ممکن است نیاز باشد که یک `if` داخل `if` دیگر قرار بگیرد.

مثال ۰: بررسی زوج و مثبت بودن عدد

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

int main() {
    int num;
    cout << "یک عدد وارد کنید: ";
    cin >> num;

    if (num > 0) { // بررسی مثبت بودن عدد
        if (num % 2 == 0) { // بررسی زوج بودن عدد
            cout << "عدد مثبت و زوج است" << endl;
        }
    }
}
```

Edit ⌐ Copy ☉

```

} else {
    cout << "عدد مثبت و فرد است" << endl;
}
} else {
    cout << "عدد منفی است" << endl;
}

return 0;
}

```

وروودی:

8

Edit ⌂ Copy ☉

خروجی:

عدد مثبت و زوج است.

Edit ⌂ Copy ☉

۶. نکات مهم

- همیشه از {} برای بلاک‌های کد استفاده کنید:
اگر یک if یا else فقط یک دستور داشته باشد، می‌توان {} را حذف کرد، اما بهتر است همیشه از آن استفاده کنید تا خوانایی کد بهتر شود.

Edit ⌂ Copy ☉

```

if (num > 0)
    cout << endl; // مجاز ولی مستعد خطأ در آینده

```

- دقت کنید که else همیشه مربوط به if بالاترین سطح قبل از خودش است.
اگر {} را نگذارید، ممکن است if به else اشتباہی متصل شود.

Edit ⌂ Copy ☉

```

if (x > 10)
if (x < 20)

```

```

cout << "بین 10 و 20 است";
else // else if این متصل شده است
      cout << "بزرگتر از 20 است";

```

◆ ساختار **switch-case** در **++C** ◆

در زبان **++C**, ساختار **switch-case** برای بررسی مقدار یک متغیر با چند مقدار ثابت استفاده می‌شود. این ساختار بهینه‌تر و خواناتر از استفاده‌ی **if-else if** متوالی است.

◆ نحوه‌ی کار **switch-case** ◆

ساختار کلی **switch** به شکل زیر است:

```

Edit ⌗ Copy ⌋

switch (عبارت) {
    case 1: مقدار 1:
        کدهای اجرا شونده اگر مقدار برابر مقدار 1 باشد //
        break;
    case 2: مقدار 2:
        کدهای اجرا شونده اگر مقدار برابر مقدار 2 باشد //
        break;
    ...
    default:
        ها اجرا نشد، این قسمت اجرا می‌شود اگر هیچ‌کدام از //
}

```

◆ نکات مهم:

- ✓ مقدار **عبارت** در **switch** باید از نوع عدد صحیح (**int, char, enum, bool**) باشد.
- ✓ مقدارهای **case** باید ثابت (**const**) باشند و نمی‌توان از متغیر استفاده کرد.
- ✓ دستور **break**; برای خروج از **switch** استفاده می‌شود. در صورت نبود آن، اجرای کد به **case** بعدی ادامه می‌یابد (به این رفتار *fall-through* می‌گویند).
- ✓ بخش **default** اختیاری است، اما بهتر است همیشه وجود داشته باشد تا مقدارهای پیش‌بینی نشده را مدیریت کند.

◆ مثال ۱: بررسی روزهای هفته ◆

```

#include <iostream>
using namespace std;

```

```

int main() {
    int day;
    cout << "یک عدد بین ۱ تا ۷ وارد کنید";
    cin >> day;

    switch (day) {
        case 1:
            cout << "شنبه" << endl;
            break;
        case 2:
            cout << "یکشنبه" << endl;
            break;
        case 3:
            cout << "دوشنبه" << endl;
            break;
        case 4:
            cout << "سهشنبه" << endl;
            break;
        case 5:
            cout << "چهارشنبه" << endl;
            break;
        case 6:
            cout << "پنجشنبه" << endl;
            break;
        case 7:
            cout << "جمعه" << endl;
            break;
        default:
            cout << "عدد نامعتبر!" << endl;
    }

    return 0;
}

```

◆ ورودی و خروجی مثال:

- 3: ورودی: 3 ✓
- خروجی: دوشنبه ✓
- 8: ورودی: 8 ✓
- خروجی: عدد نامعتبر! ✓

استفاده از `break` (fall-through) بدون `switch` ◆

گاهی اوقات عمدآ از استفاده نمیکنیم تا اجرای کد به `case` بعدی منتقل شود.

مثال ۳: بررسی حروف صدادار و بیصدا ◆

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

int main() {
    char letter;
    cout << "یک حرف وارد کنید ";
    cin >> letter;

    switch (letter) {
        case 'a': case 'e': case 'i': case 'o': case 'u':
        case 'A': case 'E': case 'I': case 'O': case 'U':
            cout << "حرف صدادار است" << endl;
            break;
        default:
            cout << "حرف بیصدا است" << endl;
    }

    return 0;
}
```

Edit ⚙ Copy ☉

e: ورودی ✓

خروجی: حرف صدادار است. ✓

b: ورودی ✓

خروجی: حرف بیصدا است. ✓

تفاوت `switch-case` و `if-else if` ◆

switch-case	if-else if	ویژگی
enum و <code>char</code>	هر شرطی را میتوان بررسی کرد	مقایسه متغیرها
سریعتر، بهینه در بررسی مقادیر ثابت	کندتر در مقایسه با <code>switch</code> در شرایط زیاد	سرعت اجرا

خوانایی بیشتر در مقادیر مشخص

خوانایی کمتر در شرایط زیاد

خوانایی کد

- اگر تعداد مقایسه‌ها کم باشد، `if-else` بهتر است.
- اگر مقادیر ثابت زیادی مقایسه شوند، `switch-case` بهینه‌تر است.

آموزش حلقه `for` در C++

حلقه چیست؟

حلقه‌ها در برنامه‌نویسی برای اجرای یک قطعه کد چندین بار استفاده می‌شوند. در C++، `for` یکی از پرکاربردترین حلقه‌ها است که برای تکرار مشخص و کنترل شده استفاده می‌شود.

ساختار کلی حلقه `for`

```
for (شروع; شرط; تغییر) {
    کدهای اجراشونده //
}
```

Edit ⌐ Copy ☰

- شروع (Initialization) → مقداردهی اولیه به متغیر کنترل کننده حلقه
- شرط (Condition) → بررسی می‌شود که آیا حلقه ادامه یابد یا متوقف شود
- تغییر (Update) → مقدار متغیر کنترل کننده تغییر می‌کند

مثال ساده: چاپ اعداد ۱ تا ۵

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

int main() {
    for (int i = 1; i <= 5; i++) {
        cout << i << " ";
    }
    return 0;
}
```

Edit ⌐ Copy ☰

خروجی:

```
1 2 3 4 5
```

Edit ⌂ Copy ⌂

توضیح:

مقدار اولیه است.

حلقه اجرا می‌شود تا زمانی که `i <= 5` باشد.

در هر تکرار، `i` یک واحد (`++i`) افزایش پیدا می‌کند.

◆ شمارش معکوس

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

int main() {
    for (int i = 10; i >= 1; i--) {
        cout << i << " ";
    }
    return 0;
}
```

Edit ⌂ Copy ⌂

خروجی:

```
10 9 8 7 6 5 4 3 2 1
```

Edit ⌂ Copy ⌂

در اینجا مقدار `i` در هر تکرار `1` واحد کاهش می‌یابد (`--i`).

◆ مجموع اعداد ۱ تا N (با دریافت ورودی)

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

Edit ⌂ Copy ⌂

```

int main() {
    int n, sum = 0;
    cout << " عدد را وارد کنید ";
    cin >> n;

    for (int i = 1; i <= n; i++) {
        sum += i;
    }

    cout << " برابر است با " << n << " مجموع اعداد 1 تا " << sum;
    return 0;
}

```

ورودي: 5

خروجي: مجموع اعداد 1 تا 5 برابر است با: 15

◆ ضرب اعداد از 1 تا N

```

#include <iostream>
using namespace std;

int main() {
    int n, fact = 1;
    cout << " عدد را وارد کنید ";
    cin >> n;

    for (int i = 1; i <= n; i++) {
        fact *= i;
    }

    cout << n << " برابر است با " << " فاكتوريل " << fact;
    return 0;
}

```

ورودي: 5

خروجي: فاكتوريل 5 برابر است با: 120

چاپ اعداد زوج از ۲ تا ۲۰

Edit ⌂ Copy ☉

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

int main() {
    for (int i = 2; i <= 20; i += 2) {
        cout << i << " ";
    }
    return 0;
}
```

خروجی:

Edit ⌂ Copy ☉

```
2 4 6 8 10 12 14 16 18 20
```

در اینجا `i` در هر تکرار ۲ واحد افزایش پیدا می‌کند (`i += 2`).

استفاده از `for` برای شمارش معکوس حروف

Edit ⌂ Copy ☉

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

int main() {
    for (char ch = 'Z'; ch >= 'A'; ch--) {
        cout << ch << " ";
    }
    return 0;
}
```

خروجی:

Edit ⌂ Copy ☉

```
Z Y X W V U T S R Q P O N M L K J I H G F E D C B A
```

♦ حلقه (Nested For) تو در تو for

چاپ مربع ستاره‌ای با for تو در تو

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

int main() {
    int n;
    cout << "اندازه مربع را وارد کنید";
    cin >> n;

    for (int i = 1; i <= n; i++) { // کنترل سطرها
        for (int j = 1; j <= n; j++) { // کنترل ستونها
            cout << "* ";
        }
        cout << endl;
    }
    return 0;
}
```

ورودی: 4
خروجی:

```
* * * *
* * * *
* * * *
* * * *
```

در اینجا یک for برای کنترل سطرها و یک for برای کنترل ستونها استفاده شده است. ✓

♦ نکات مهم درباره حلقه for

می‌توان for را بدون مقدار اولیه نوشت: ✓

Edit ⌛ Copy ⌂

```
int i = 1;
for ( ; i <= 5; i++) {
    cout << i << " ";
}
```

✓ می‌توان `for` را بدون افزایش مقدار نوشت (اما باید داخل بدنه مقدار را تغییر داد):

Edit ⌛ Copy ⌂

```
for (int i = 1; i <= 5;) {
    cout << i << " ";
    i++; // تغییر مقدار باید داخل بدنه باشد
}
```

✓ حلقه‌ی بینهایت با `for`:

Edit ⌛ Copy ⌂

```
for (;;) {
    cout << "!"<< " این حلقه هیچ وقت متوقف نمی‌شود";
}
```

⚠️ باید داخل حلقه `break`; یا `return`; قرار داد تا متوقف شود.

چالش برای شما! 🔥

? برنامه‌ای بنویسید که جدول ضرب اعداد ۱ تا ۱۰ را چاپ کند!

💡 می‌توانید از `for` تو در تو استفاده کنید.

نتیجه‌گیری 🎯

- حلقه `for` برای تعداد تکرار مشخص بسیار کارآمد است.
- ساختار آن شامل مقدار اولیه، شرط، و تغییر مقدار متغیر است.
- می‌توان از `for` تو در تو برای ایجاد ساختارهای پیچیده‌تر مانند جداول و الگوها استفاده کرد.
- برای شمارش معکوس یا پرش در مقدار متغیر، می‌توان مقدار افزایش/کاهش را تغییر داد (`i--`, `i+=2` و غیره).

آموزش حلقه while در C++



◆ حلقه چیست؟

حلقه‌ها برای تکرار یک قطعه کد استفاده می‌شوند. حلقه `while` تا زمانی که یک شرط برقرار باشد اجرا می‌شود. برخلاف `for` که تعداد تکرار مشخص دارد، `while` معمولاً برای حلقه‌های نامشخص (مثل دریافت ورودی تا زمانی که مقدار درستی وارد شود) مناسب‌تر است.

◆ ساختار کلی حلقه while

```
while (شرط) {  
    کدهایی که تا زمانی که شرط برقرار است اجرا می‌شوند //  
}
```

Edit ⌂ Copy ⌂

- ✓ شرط (`Condition`) بررسی می‌شود، اگر `true` باشد، بدن حلقه اجرا می‌شود.
- ✓ بعد از اجرای بدن، شرط دوباره بررسی می‌شود.
- ✓ اگر شرط `false` شود، حلقه متوقف می‌شود.

◆ مثال ۱: چاپ اعداد ۱ تا ۵

```
#include <iostream>  
using namespace std;  
  
int main() {  
    int i = 1; // مقدار اولیه  
    while (i <= 5) { // شرط  
        cout << i << " ";  
        i++; // افزایش مقدار  
    }  
    return 0;  
}
```

Edit ⌂ Copy ⌂

خروجی:

```
1 2 3 4 5
```

Edit ⌂ Copy ⌂

توضیح: ۱ مقدار اولیه است.

حلقه اجرا می‌شود تا زمانی که ۲ $i \leq 5$ باشد.

در هر تکرار، ۳ i یک واحد افزایش می‌یابد ($++i$).

مثال ۲: شمارش معکوس از ۱۰ تا ۱

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

int main() {
    int i = 10;
    while (i >= 1) {
        cout << i << " ";
        i--;
    }
    return 0;
}
```

Edit ⌂ Copy ⌂

خروجی:

```
10 9 8 7 6 5 4 3 2 1
```

Edit ⌂ Copy ⌂

مقدار i در هر تکرار یک واحد کاهش می‌یابد ($--i$).

مثال ۳: دریافت عدد تا زمانی که مقدار صحیح وارد شود

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

Edit ⌂ Copy ⌂

```

int main() {
    int num;
    cout << "یک عدد مثبت وارد کنید";
    cin >> num;

    while (num <= 0) {
        cout << "اشتباه! عدد باید مثبت باشد. دوباره وارد کنید";
        cin >> num;
    }

    cout << "عدد صحیح وارد شد: " << num;
    return 0;
}

```

3 → 0 → 2- → 5- ورودی:
خروجی:

Edit ⌚ Copy ⌂

ashbeah! عدد باید مثبت باشد. دوباره وارد کنید
 ashbeah! عدد باید مثبت باشد. دوباره وارد کنید
 ashbeah! عدد باید مثبت باشد. دوباره وارد کنید
 عدد صحیح وارد شد: 3

این مثال نشان می‌دهد که `while` تا زمانی که شرط برقرار باشد اجرا می‌شود.

♦ مثال ۴: جمع کردن اعداد تا زمانی که کاربر ۰ وارد کند

Edit ⌚ Copy ⌂

```

#include <iostream>
using namespace std;

int main() {
    int sum = 0, num;

```

```

cout << "اعداد را وارد کنید (برای خروج ۰ بزنید)" ;
cin >> num;

while (num != 0) {
    sum += num;
    cin >> num;
}

cout << "مجموع اعداد: " << sum;
return 0;
}

```

ورودی: ۰ ۲- ۷ ۳ ۵

خروجی:

Edit ⌚ Copy ☉

مجموع اعداد: ۱۳

حلقه تا زمانی اجرا می‌شود که ورودی ۰ نباشد.

♦ مثال ۵: چاپ اعداد زوج تا ۲۰

Edit ⌚ Copy ☉

```

#include <iostream>
using namespace std;

int main() {
    int i = 2;
    while (i <= 20) {
        cout << i << " ";
        i += 2;
    }
    return 0;
}

```

خروجی:

```
2 4 6 8 10 12 14 16 18 20
```

Edit ⌂ Copy ⌂

در اینجا مقدار `i` در هر تکرار ۲ واحد افزایش پیدا می‌کند (`i += 2`).

حلقه‌ی بینهایت با while

```
while (true) {  
    cout << "این حلقه هیچوقت متوقف نمی‌شود"  
}
```

Edit ⌂ Copy ⌂

این حلقه بینهایت اجرا می‌شود و باید داخلش `break` یا `return`؛ یا بگذاریم تا متوقف شود.

تفاوت while و for

استفاده مناسب	حلقه
زمانی که تعداد دفعات تکرار مشخص باشد	<code>for</code>
زمانی که تعداد دفعات تکرار نامشخص باشد (مثلاً منتظر ورودی کاربر)	<code>while</code>

چالش برای شما!

برنامه‌ای بنویسید که تا زمانی که کاربر عددی بین ۱ تا ۱۰۰ وارد نکرده، از او ورودی بگیرد!

راهنمایی: استفاده از `while` برای بررسی شرط ورودی.

نتیجه‌گیری

- برای حلقه‌های نامشخص و شرطی مناسب است.
- تا زمانی که شرط `true` باشد، اجرا می‌شود و وقتی شرط `false` شد، متوقف می‌شود.
- می‌توان آن را برای ورودی نامعتبر، شمارش، محاسبات و موارد دیگر استفاده کرد.
- اگر `while` بدون شرط خروجی باشد، به یک حلقه بینهایت تبدیل می‌شود!

آموزش حلقه do-while در C++



◆ حلقه چیست؟

حلقه‌ها در برنامه‌نویسی برای تکرار اجرای کد تا زمانی که یک شرط برقرار باشد استفاده می‌شوند.
حلقه `do-while` یک نوع حلقه شرطی است که حداقل یک بار اجرا می‌شود، حتی اگر شرط برقرار نباشد!

◆ تفاوت `while` با `do-while`

حلقه `while` → ابتدا شرط بررسی می‌شود و اگر برقرار باشد، کد اجرا می‌شود.

حلقه `do-while` → ابتدا کد اجرا می‌شود و سپس شرط بررسی می‌شود.

يعني `do-while` همیشه حداقل یک بار اجرا می‌شود، حتی اگر شرط نادرست باشد.

◆ ساختار کلی `do-while`

```
do {  
    کدهایی که حداقل یکبار اجرا می‌شوند //  
} while (شرط);
```

Edit ⌂ Copy ☰

◆ دقت کنید که بعد از `while` باید `;` قرار دهید!

◆ مثال ۱: چاپ اعداد ۱ تا ۵

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

Edit ⌂ Copy ☰

```

int main() {
    int i = 1;
    do {
        cout << i << " ";
        i++;
    } while (i <= 5);

    return 0;
}

```

◆ خروجی:

Edit ⌚ Copy ⌂

1 2 3 4 5

◆ در اینجا ابتدا 1 چاپ می‌شود، سپس i افزایش یافته و شرط بررسی می‌شود.
◆ مثال ۲: دریافت عدد تا زمانی که مقدار صحیح وارد شود

Edit ⌚ Copy ⌂

```

#include <iostream>
using namespace std;

int main() {
    int num;
    do {
        cout << "بک عدد مثبت وارد کنید" << endl;
        cin >> num;
    } while (num <= 0);

    cout << "عدد صحیح وارد شد: " << num;
    return 0;
}

```

3 → 0 → 2- → 5- ورودی:

◆ خروجی:

Edit ⌛ Copy ⌂

:پک عدد مثبت وارد کنید:

:پک عدد مثبت وارد کنید:

:پک عدد مثبت وارد کنید:

عدد صحیح وارد شد: 3

✳ حلقه حداقل یکبار اجرا می‌شود، حتی اگر اولین مقدار ورودی اشتباه باشد.

♦ مثال ۳: دریافت پسورد از کاربر تا زمانی که درست وارد کند

Edit ⌛ Copy ⌂

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

int main() {
    string password;
    string correctPass = "1234";

    do {
        cout << "رمز عبور را وارد کنید ";
        cin >> password;
    } while (password != correctPass);

    cout << "ورود موفق!";
    return 0;
}
```

✳ برنامه از کاربر رمز عبور می‌گیرد و تا زمانی که مقدار صحیح وارد نشود، تکرار می‌شود.

♦ مثال ۴: نمایش یکبار پیام حتی اگر شرط برقرار نباشد

Edit ⌛ Copy ⌂

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

```

int main() {
    int x = 10;

    do {
        cout << "این پیام حداقل یکبار چاپ میشود" << endl;
    } while (x < 5);

    return 0;
}

```

خروجی:

Edit ⌚ Copy ☉

این پیام حداقل یکبار چاپ میشود!

حتی با اینکه شرط `x < 5` برقرار نیست، اما `do-while` باعث شده یک بار اجرا شود.

◆ حلقه بینهایت با `do-while`

Edit ⌚ Copy ☉

```

do {
    cout << "این حلقه بینهایت اجرا میشود" << endl;
} while (true);

```

این حلقه همیشه اجرا خواهد شد و باید داخلش `break`; یا `return`; یا بگذاریم تا متوقف شود.

◆ تفاوت `do-while` و `while`

ویژگی	حلقه
ابتدا شرط بررسی میشود، اگر <code>false</code> باشد، اصلاً اجرا نمیشود.	<code>while</code>
ابتدا کد اجرا میشود، سپس شرط بررسی میشود (حداقل یک بار اجرا دارد).	<code>do-while</code>

چالش برای شما!

؟ برنامه‌ای بنویسید که تا زمانی که کاربر عددی بین ۱ تا ۱۰۰ وارد نکرده، از او ورودی بگیرد!

💡 راهنمایی: از `do-while` استفاده کنید تا حداقل یکبار مقدار ورودی گرفته شود.

نتیجه‌گیری

- حداقل یک بار اجرا می‌شود.
- برای ورودی گرفتن از کاربر و بررسی شرط بعد از اجرا کاربرد دارد.
- می‌توان از آن برای کنترل ورودی‌های نامعتبر استفاده کرد.

آموزش دستورات کنترل جریان (continue و break)

در ++C

دستورات کنترل جریان چیست؟

در زبان C++, گاهی اوقات نیاز داریم که جریان اجرای برنامه را کنترل کنیم، یعنی:

- خروج از یک حلقه زودتر از زمان معمول (break)
- رد کردن یک تکرار و رفتن به تکرار بعدی (continue)

برای این کار از دستورات break و continue استفاده می‌کنیم.

۱. دستور break

دستور break باعث خروج فوری از حلقه می‌شود.

اگر break درون یک حلقه باشد، بلافاصله حلقه را متوقف می‌کند و از آن خارج می‌شود.
در switch-case نیز برای خروج از case استفاده می‌شود.

مثال ۱: متوقف کردن یک حلقه for با break

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

int main() {
    for (int i = 1; i <= 10; i++) {
        if (i == 5) {
            break; // حلقه در مقدار 5 متوقف می‌شود
        }
        cout << i << " ";
    }
}
```

Edit ⌂ Copy ⌂

```
    }
    return 0;
}
```

Edit ⚙ Copy ☉

خروجی: ♦

```
1 2 3 4
```

Edit ⚙ Copy ☉

در اینجا، وقتی `break` باعث توقف حلقه می‌شود. ✨

مثال ۲: جستجوی عددی در یک `while` با `break` ♦

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

int main() {
    int num;
    while (true) {
        cout << "یک عدد مثبت وارد کنید";
        cin >> num;

        if (num > 0) {
            cout << " عدد معتبر است" << endl;
            break; // خروج از حلقه پس از دریافت مقدار صحیح
        }
        cout << ". عدد نامعتبر! دوباره تلاش کنید" << endl;
    }
    return 0;
}
```

Edit ⚙ Copy ☉

در این مثال، تا زمانی که کاربر عدد مثبت وارد نکند، حلقه ادامه دارد. اما اگر عدد معتبر باشد، `break` باعث خروج از حلقه می‌شود. ✨

♦ مثال ۳: خروج زودهنگام از switch-case

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

int main() {
    int choice;
    cout << "پک عدد بین ۱ تا ۳ وارد کنید";
    cin >> choice;

    switch (choice) {
        case 1:
            cout << "انتخاب ۱" << endl;
            break;
        case 2:
            cout << "انتخاب ۲" << endl;
            break;
        case 3:
            cout << "انتخاب ۳" << endl;
            break;
        default:
            cout << " عدد نامعتبر" << endl;
    }
    return 0;
}
```

باعث خروج از هر `case` می‌شود. بدون `break`، اجرای برنامه به `case` بعدی ادامه پیدا می‌کند.

۲. دستور continue

دستور `continue` باعث می‌شود که باقی‌مانده کد در آن تکرار از حلقه اجرا نشود و مستقیماً به تکرار بعدی برود. بخلاف `break` که حلقه را متوقف می‌کند، `continue` فقط آن تکرار خاص را رد می‌کند!

♦ مثال ۴: رد کردن اعداد زوج در یک حلقه for با continue

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

```

int main() {
    for (int i = 1; i <= 10; i++) {
        if (i % 2 == 0) {
            continue; // اگر عدد زوج باشد، از اجرای دستورات بعدی رد می‌شود
        }
        cout << i << " ";
    }
    return 0;
}

```

Edit ⌂ Copy ☉

◆ خروجی:

```
1 3 5 7 9
```

Edit ⌂ Copy ☉

هر عدد زوجی، cout را اجرا می‌کند و continue را اجرا نمی‌شود. *

◆ مثال ۵: رد کردن اعداد منفی در while

```

#include <iostream>
using namespace std;

int main() {
    int num;

    while (true) {
        cout << "یک عدد وارد کنید (0 برای خروج) ";
        cin >> num;

        if (num < 0) {
            cout << ". اعداد منفی مجاز نیستند! دوباره امتحان کنید" << endl;
            continue; // از اجرای بقیه کدهای داخل حلقه رد می‌شود و عدد بعدی را دریافت می‌کند
        }
    }
}

```

Edit ⌂ Copy ☉

```

if (num == 0) {
    break; // خروج از حلقه
}

cout << "عدد معنیز: " << num << endl;
}

return 0;
}

```

در این مثال: *

- اگر کاربر عدد منفی وارد کند، `continue` باعث می‌شود که عدد معنیز چاپ نشود و حلقه مستقیماً به مرحله گرفتن عدد جدید برود.
- اگر عدد ۰ باشد، `break` باعث خروج از حلقه می‌شود.

تفاوت `continue` و `break`

عملکرد	دستور
کامل از حلقه خارج می‌شود	<code>break</code>
فقط تکرار جاری را رد می‌کند و به تکرار بعدی می‌رود	<code>continue</code>

چالش برای شما!

- برنامه‌ای بنویسید که از عدد ۱ تا ۳۰ را چاپ کند اما:
- اعداد زوج را چاپ نکند (از `continue` استفاده کنید).
- اگر به عدد ۱۵ رسید، حلقه را متوقف کند (از `break` استفاده کنید).

نتیجه‌گیری

- برای خروج فوری از حلقه‌ها و `switch-case` استفاده `break` می‌شود.
- برای رد کردن یک تکرار از حلقه و رفتن به تکرار بعدی مفید است.
- با استفاده صحیح از این دو دستور می‌توان کنترل دقیق‌تری روی اجرای برنامه داشت.

❖ معرفی آرایه‌ها و کاربرد آن‌ها در C++

◆ آرایه چیست؟

آرایه (Array) یک مجموعه‌ای از متغیرهای همنوع است که در حافظه به صورت پشت‌سرهم ذخیره می‌شوند. هر مقدار در آرایه با استفاده از اندیس (Index) قابل دسترسی است.

ویژگی‌های آرایه: ✓

همه عناصر آرایه از یک نوع داده‌ای هستند. ✓

مقداردهی و دسترسی به عناصر با اندیس انجام می‌شود. ✓

فضای حافظه‌ای پیوسته‌ای را اشغال می‌کند. ✓

◆ تعریف آرایه در C++

```
datatype arrayName[size];
```

Edit ⓘ Copy ⬤

(int , float , char) → نوع داده (مثلاً datatype ✓

→ نام آرایه ✓

→ تعداد عناصر آرایه ✓

مثال: تعریف یک آرایه از اعداد صحیح با ۵ عنصر ❖

```
int numbers[5]; // آرایه‌ای از ۵ عدد صحیح
```

Edit ⓘ Copy ⬤

◆ مقداردهی آرایه

مقداردهی مستقیم هنگام تعریف: ✓

```
int numbers[5] = {10, 20, 30, 40, 50};
```

Edit ⓘ Copy ⬤

مقداردهی جداگانه به هر خانه: ✓

Edit ⌂ Copy ⌂

```
numbers[0] = 10;
numbers[1] = 20;
numbers[2] = 30;
numbers[3] = 40;
numbers[4] = 50;
```

♦ دسترسی به عناصر آرایه

هر عنصر با اندیس از ۰ شروع می‌شود.

♦ دسترسی به عناصر آرایه

هر عنصر با اندیس از ۰ شروع می‌شود.

Edit ⌂ Copy ⌂

```
cout << numbers[0]; // 10
cout << numbers[2]; // 30
```

♦ پیمایش آرایه با حلقه for

Edit ⌂ Copy ⌂

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

int main() {
    int numbers[5] = {10, 20, 30, 40, 50};

    for (int i = 0; i < 5; i++) {
        cout << "numbers[" << i << "] = " << numbers[i] << endl;
    }

    return 0;
}
```

خروجی:

Edit ⌛ Copy ⌂

```
numbers[0] = 10  
numbers[1] = 20  
numbers[2] = 30  
numbers[3] = 40  
numbers[4] = 50
```

◆ کاربردهای آرایه‌ها

- ✓ ذخیره و پردازش مجموعه‌ای از داده‌ها مثل نمرات دانشآموزان
- ✓ جستجو و مرتب‌سازی داده‌ها
- ✓ مدیریت لیست‌ها مانند لیست خرید یا اسامی
- ✓ استفاده در ساختارهای داده‌ای پیچیده‌تر مثل ماتریس، رشته، و لیست‌ها

خلاصه

- ✓ آرایه مجموعه‌ای از عناصر همنوع است.
- ✓ اندیس‌ها از ۰ شروع می‌شوند.
- ✓ دسترسی و تغییر مقدار با اندیس امکان‌پذیر است.
- ✓ می‌توان با حلقه‌ها روی آرایه پردازش انجام داد.

◆ آرایه‌های یکبعدی در C +

◆ آرایه یکبعدی چیست؟

آرایه یکبعدی در C + مجموعه‌ای از مقادیر همنوع است که در یک ردیف پیوسته از حافظه ذخیره می‌شوند و با اندیس (index) قابل دسترسی هستند.

- ✓ ویژگی‌های آرایه یکبعدی:
- ✓ همه عناصر از یک نوع داده‌ای هستند.
- ✓ اندیس از ۰ شروع می‌شود.
- ✓ مقداردهی و دسترسی با اندیس انجام می‌شود.
- ✓ پیمایش آرایه با حلقه‌ها بسیار رایج است.

◆ تعریف آرایه یکبعدی در C++

ساختار کلی تعریف آرایه:

Edit ⌂ Copy ☒

```
datatype arrayName[size];
```

مثال: تعریف آرایه‌ای از ۵ عدد صحیح

Edit ⌂ Copy ☒

```
int numbers[5];
```

نکته: مقدار size تعداد عناصر را مشخص می‌کند و ثابت است (نمی‌توان آن را تغییر داد).

◆ مقداردهی آرایه یکبعدی

روش ۱: مقداردهی مستقیم هنگام تعریف ✓

Edit ⌂ Copy ☒

```
int numbers[5] = {10, 20, 30, 40, 50};
```

روش ۲: مقداردهی جداگانه به هر خانه ✓

Edit ⌂ Copy ☒

```
numbers[0] = 10;  
numbers[1] = 20;  
numbers[2] = 30;  
numbers[3] = 40;  
numbers[4] = 50;
```

روش ۳: مقداردهی بدون تعیین سایز (C++ سایز را تشخیص می‌دهد) ✓

Edit ⌂ Copy ☒

```
int numbers[] = {10, 20, 30, 40, 50};
```

◆ دسترسی به عناصر آرایه

برای دسترسی به هر مقدار، از نام آرایه و اندیس آن استفاده می‌شود.

Edit ⌚ Copy ☉

```
cout << numbers[0]; // 10  
cout << numbers[3]; // 40
```

◆ پیمایش آرایه با for

Edit ⌚ Copy ☉

```
#include <iostream>  
using namespace std;  
  
int main() {  
    int numbers[5] = {10, 20, 30, 40, 50};  
  
    for (int i = 0; i < 5; i++) {  
        cout << "numbers[" << i << "] = " << numbers[i] << endl;  
    }  
  
    return 0;  
}
```

◆ خروجی:

Edit ⌚ Copy ☉

```
numbers[0] = 10  
numbers[1] = 20  
numbers[2] = 30  
numbers[3] = 40  
numbers[4] = 50
```

◆ گرفتن مقادیر آرایه از ورودی

Edit ⌚ Copy ☉

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

int main() {
    int numbers[5];

    // گرفتن ورودی از کاربر
    cout << "Enter 5 numbers: ";
    for (int i = 0; i < 5; i++) {
        cin >> numbers[i];
    }

    // نمایش مقادیر وارد شده
    cout << "You entered: ";
    for (int i = 0; i < 5; i++) {
        cout << numbers[i] << " ";
    }

    return 0;
}
```

ورودی: ◆

Edit ⌚ Copy ☉

10 20 30 40 50

خروجی: ◆

Edit ⌚ Copy ☉

You entered: 10 20 30 40 50

◆ آرایه به عنوان متغیر مستقل در عملیات ریاضی

می‌توان جمع، میانگین و بیشترین مقدار را از آرایه محاسبه کرد.

♦ مثال: محاسبه مجموع عناصر آرایه

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

int main() {
    int numbers[] = {10, 20, 30, 40, 50};
    int sum = 0;

    for (int i = 0; i < 5; i++) {
        sum += numbers[i];
    }

    cout << "Sum: " << sum;
    return 0;
}
```

Edit ⌛ Copy ⌂

◆ خروجی:

```
Sum: 150
```

Edit ⌛ Copy ⌂

◆ کاربردهای آرایه‌های یکبعدی

✓ ذخیره و پردازش لیست نمرات دانشآموزان

✓ پردازش داده‌های عددی و آماری

✓ ذخیره اطلاعات کاربران مانند شماره تماس‌ها

✓ پیاده‌سازی ساختارهای داده‌ای پیشرفته‌تر مانند پشته و صف

جمع‌بندی

- ✓ آرایه یک مجموعه از مقادیر همنوع است.
- ✓ مقداردهی مستقیم، جداگانه یا از ورودی امکان‌پذیر است.
- ✓ حلقه‌های `for` و `while` برای پردازش آرایه کاربرد دارند.
- ✓ آرایه‌ها در بسیاری از الگوریتم‌ها نقش کلیدی دارند.

آرایه‌های چندبعدی در C++

◆ آرایه چندبعدی چیست؟

آرایه‌های چندبعدی مجموعه‌ای از آرایه‌های یکبعدی هستند که در چندین بُعد سازمان‌دهی شده‌اند.

ویژگی‌های آرایه چندبعدی:

- ✓ آرایه‌ای از آرایه‌ها است.
- ✓ اندیس‌گذاری در چند سطح انجام می‌شود.
- ✓ متداول‌ترین نوع آن آرایه‌های دو بعدی (ماتریس) هستند.

◆ تعریف آرایه چندبعدی در C++

ساختار کلی:

```
datatype arrayName[size1][size2]...[sizeN];
```

Edit ⌂ Copy ☒

(`int`, `float` → نوع داده (مثل `datatype`) ◆

تعداد عناصر در هر بعد → `size1, size2, ..., sizeN` ◆

◆ آرایه‌های دو بعدی (ماتریس)

◆ تعریف یک آرایه 2×3 (۲ سطر، ۳ ستون)

```
int matrix[2][3];
```

Edit ⌂ Copy ☒

روش مقداردهی مستقیم: ✓

Edit ⌂ Copy ☉

```
int matrix[2][3] = {  
    {1, 2, 3},  
    {4, 5, 6}  
};
```

دسترسی به عناصر: ✓

Edit ⌂ Copy ☉

```
cout << matrix[0][1]; // خروجی: 2  
cout << matrix[1][2]; // خروجی: 6
```

◆ پیمایش آرایه دو بعدی با `for`

Edit ⌂ Copy ☉

```
#include <iostream>  
using namespace std;  
  
int main() {  
    int matrix[2][3] = {  
        {1, 2, 3},  
        {4, 5, 6}  
    };  
  
    for (int i = 0; i < 2; i++) { // پیمایش سطرها  
        for (int j = 0; j < 3; j++) { // پیمایش ستونها  
            cout << matrix[i][j] << " ";  
        }  
        cout << endl;  
    }  
  
    return 0;  
}
```

خروجی:

Edit ⌛ Copy ⌂

```
1 2 3  
4 5 6
```

گرفتن مقادیر آرایه دو بعدی از ورودی

Edit ⌛ Copy ⌂

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

int main() {
    int matrix[2][2];

    cout << "Enter 4 numbers: ";
    for (int i = 0; i < 2; i++) {
        for (int j = 0; j < 2; j++) {
            cin >> matrix[i][j];
        }
    }

    cout << "Matrix: \n";
    for (int i = 0; i < 2; i++) {
        for (int j = 0; j < 2; j++) {
            cout << matrix[i][j] << " ";
        }
        cout << endl;
    }

    return 0;
}
```

ورودی:

Edit ⌛ Copy ⌂

```
1 2 3 4
```

◆ خروجی:

Matrix:

```
1 2  
3 4
```

Edit ⌂ Copy ☒

◆ آرایه سه بعدی در C++

◆ ساختار کلی:

```
datatype arrayName[size1][size2][size3];
```

Edit ⌂ Copy ☒

◆ مثال: تعریف آرایه $3 \times 2 \times 2$

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

Edit ⌂ Copy ☒

◆ دسترسی به مقدار خاص

```
cout << arr[1][0][2]; // 9
```

Edit ⌂ Copy ☒

◆ کاربردهای آرایه‌های چندبعدی

پردازش ماتریس‌ها (تصاویر، گرافیک، فیلترها)

مدیریت جداول (صفحات گسترده، پایگاه‌داده‌ها)

حل مسائل مرتبط با فیزیک و ریاضیات

برنامه‌نویسی بازی‌ها (سیستم‌های مختصات، نقشه‌ها)

خلاصه

- ✓ آرایه‌های دوبعدی (ماتریس) متداول‌ترین نوع آرایه‌های چندبعدی هستند.
- ✓ حلقه‌های تودرتو (for) برای پیمایش آرایه‌های چندبعدی استفاده می‌شود.
- ✓ مقداردهی مستقیم و دریافت ورودی از کاربر ممکن است.
- ✓ برای ذخیره و پردازش داده‌های پیچیده مانند تصاویر و جداول استفاده می‌شوند.

+ توابع داخلی کار با آرایه‌ها در C++

۱. به دست آوردن اندازه آرایه (`sizeof`) ✓
در C++ تابعی مانند `len()` در پایتون وجود ندارد، اما می‌توان از `sizeof` برای محاسبه تعداد عناصر آرایه استفاده کرد.

Edit ⌛ Copy ⌂

```
int arr[] = {10, 20, 30, 40, 50};  
int size = sizeof(arr) / sizeof(arr[0]);  
cout << "Size: " << size; // 5 خروجی:
```

۲. مقداردهی به آرایه (`fill`) ✓

تابع `fill()` مقدار تمام خانه‌های آرایه را مقداردهی می‌کند.

<`algorithm`> نیازمند

Edit ⌛ Copy ⌂

```
#include <iostream>  
#include <algorithm>  
using namespace std;  
  
int main() {  
    int arr[5];  
    fill(arr, arr + 5, 100); // مقدار تمام خانه‌ها 100 می‌شود  
    // خروجی: 100 100 100 100 100  
}
```

۳. یافتن کوچکترین و بزرگترین مقدار ()

<algorithm> نیازمند

Edit ⌂ Copy ⌂

```
#include <iostream>
#include <algorithm>
using namespace std;

int main() {
    int arr[] = {5, 10, 2, 8, 1};
    int size = sizeof(arr) / sizeof(arr[0]);

    cout << "Min: " << *min_element(arr, arr + size) << endl;
    cout << "Max: " << *max_element(arr, arr + size) << endl;
}
```

خروجی:

Edit ⌂ Copy ⌂

```
Min: 1
Max: 10
```

۴. محاسبه مجموع عناصر ()

<numeric> نیازمند

Edit ⌂ Copy ⌂

```
#include <iostream>
#include <numeric>
using namespace std;

int main() {
    int arr[] = {1, 2, 3, 4, 5};
    int size = sizeof(arr) / sizeof(arr[0]);

    int sum = accumulate(arr, arr + size, 0);
    cout << "Sum: " << sum; // 15 خروجی:
}
```

(find) .جستجوی مقدار ✓

<algorithm> نیازمند ✘

Edit ⌂ Copy ⌂

```
#include <iostream>
#include <algorithm>
using namespace std;

int main() {
    int arr[] = {10, 20, 30, 40, 50};
    int size = sizeof(arr) / sizeof(arr[0]);

    int* pos = find(arr, arr + size, 30);
    if (pos != arr + size)
        cout << "Found at index: " << (pos - arr);
    else
        cout << "Not found";
}
```

خروجی: ♦

Edit ⌂ Copy ⌂

Found at index: 2

(sort) ۶. مرتبسازی ✓

<algorithm> نیازمند ✘

Edit ⌂ Copy ⌂

```
#include <iostream>
#include <algorithm>
using namespace std;

int main() {
    int arr[] = {5, 2, 8, 1, 3};
    int size = sizeof(arr) / sizeof(arr[0]);
```

```
sort(arr, arr + size); // مرتبسازی صعودی
for (int x : arr) {
    cout << x << " ";
}
}
```

خروجی:

```
1 2 3 5 8
```

Edit ⌐ Copy ☒

مرتبسازی نزولی:

```
sort(arr, arr + size, greater<int>());
```

۷. بررسی آرایه مرتب شده (is_sorted) ✓

<algorithm> نیازمند ↗

```
if (is_sorted(arr, arr + size)) {
    cout << "Array is sorted!";
} else {
    cout << "Array is not sorted!";
}
```

Edit ⌐ Copy ☒

۸. معکوس کردن آرایه (reverse) ✓

<algorithm> نیازمند ↗

```
reverse(arr, arr + size);
```

Edit ⌐ Copy ☒

۹. شمارش تعداد تکرار یک مقدار (count) ✓

<algorithm> نیازمند ↗

```
int count_3 = count(arr, arr + size, 3);
cout << "Number 3 appears: " << count_3 << " times";
```

Edit ⌂ Copy ⌂

۱۰. چرخش آرایه ✓

<algorithm> ✅

```
rotate(arr, arr + 2, arr + size); // چرخش به اندازه ۲ خانه به چپ
```

Edit ⌂ Copy ⌂

خلاصه

در C++ توابع داخلی کمی برای کار با آرایه‌ها وجود دارد، اما با کتابخانه‌های `<numeric>` و `<algorithm>` می‌توان بسیاری از عملیات را انجام داد.
برای محاسبه تعداد عناصر استفاده `sizeof` می‌شود.
توابعی مانند `max_element()`، `sort()`، `find()`، `reverse()`، `accumulate()`، `min_element()` بسیار مفید هستند.

پردازش رشته‌ها در C++

در C++ پردازش رشته‌ها با استفاده از کلاس `string` از کتابخانه `<string>` انجام می‌شود. این کلاس توابع زیادی برای کار با رشته‌ها ارائه می‌دهد. در ادامه، مهمترین توابع و متدهای پردازش رشته‌ها را بررسی می‌کنیم.

۱ مقداردهی و تعریف رشته

در C++ می‌توان به چند روش یک رشته را مقداردهی کرد:

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

int main() {
    string str1 = "Hello";
    string str2("World");
    string str3(5, '*'); // *****
    cout << str1 << " " << str2 << " " << str3;
}
```

Edit ⌂ Copy ⌂

خروجی:

Edit ⌛ Copy ⌂

```
Hello World *****
```

گرفتن ورودی رشته (getline و cin) 2

cin فقط تا اولین فاصله ورودی می‌گیرد، اما getline کل خط را دریافت می‌کند:

Edit ⌛ Copy ⌂

```
string name;  
cin >> name; // فقط یک کلمه دریافت می‌کند!  
  
string fullName;  
getline(cin, fullName); // کل خط را دریافت می‌کند.
```

به دست آوردن طول رشته (size و length) 3

Edit ⌛ Copy ⌂

```
string str = "Hello";  
cout << str.length(); // 5  
cout << str.size(); // 5
```

دسترسی به کاراکترهای رشته (at و []) 4

Edit ⌛ Copy ⌂

```
string str = "Hello";  
cout << str[1]; // خروجی: e  
cout << str.at(1); // خروجی: e
```

تغییر مقدار کاراکتر رشته 5

```
string str = "Hello";
str[0] = 'M';
cout << str; // Melllo
```

Edit ⌂ Copy ⌂

الحق (اتصال) رشته‌ها (+ و ()append 6

```
string str1 = "Hello";
string str2 = " World";
string result = str1 + str2;
cout << result; // Hello World

str1.append(str2);
cout << str1; // Hello World
```

Edit ⌂ Copy ⌂

(()compare مقایسه رشته‌ها 7

```
string str1 = "apple";
string str2 = "banana";

if (str1.compare(str2) == 0)
    cout << "Equal";
else if (str1.compare(str2) < 0)
    cout << "str1 comes first";
else
    cout << "str2 comes first";
```

Edit ⌂ Copy ⌂

خروجی:

Edit ⌛ Copy ⌂

```
str1 comes first
```

(()find) 8 بررسی وجود زیررشته

Edit ⌛ Copy ⌂

```
string str = "Hello, World!";
int pos = str.find("World");

if (pos != string::npos) // میکنیم که پیدا شده یا نه
    cout << "Found at: " << pos;
else
    cout << "Not found";
```

خروجی:

Edit ⌛ Copy ⌂

```
Found at: 7
```

(()replace) 9 جایگزینی زیررشته

Edit ⌛ Copy ⌂

```
string str = "Hello, World!";
str.replace(7, 5, "C++");
cout << str; // Hello, C++!
```

(()substr) 10 استخراج زیررشته

Edit ⌛ Copy ⌂

```
string str = "Hello, World!";
string sub = str.substr(7, 5);
cout << sub; // World
```

۱ ۲ حذف فضای خالی ابتدا و انتهای رشته (trim - روش دستی)

Edit ⌛ Copy ⌂

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

string trim(string s) {
    size_t start = s.find_first_not_of(" \t");
    size_t end = s.find_last_not_of(" \t");
    return s.substr(start, end - start + 1);
}

int main() {
    string str = "Hello, World!";
    cout << "|" << trim(str) << "|"; // |Hello, World!|
}
```

۱ ۲ تبدیل حروف کوچک و بزرگ (tolower و toupper)

Edit ⌛ Copy ⌂

```
#include <iostream>
#include <string>
#include <cctype> // برای توابع کار با کاراکترها
using namespace std;

int main() {
    string str = "Hello";

    for (char &c : str) c = toupper(c);
    cout << str; // HELLO

    for (char &c : str) c = tolower(c);
    cout << str; // hello
}
```

(()erase) حذف یک کاراکتر از رشته ۱ ۳

Edit ⌛ Copy ⌂

```
string str = "abcdef";  
str.erase(2, 1); // حذف کاراکتر سوم  
cout << str; // abdef
```

(()to_string) تبدیل عدد به رشته ۱ ۴

Edit ⌛ Copy ⌂

```
int num = 123;  
string str = to_string(num);  
cout << str + "4"; // 1234
```

(()stoi(), stof(), stod) تبدیل رشته به عدد ۱ ۵

Edit ⌛ Copy ⌂

```
string str = "123";  
int num = stoi(str);  
cout << num + 1; // 124
```

خلاصه

- طول رشته → ()size و ()length ✓
- الحاق رشته‌ها → ()append و (+) ✓
- مقایسه رشته‌ها → ()compare ✓
- پیدا کردن زیررشته → ()find ✓
- جایگزینی زیررشته → ()replace ✓
- برش زیررشته → ()substr ✓
- تبدیل حروف → ()tolower و ()toupper ✓
- تبدیل بین عدد و رشته → ()stoi و ()to_string ✓

توابع در C++

توابع در C++ به ما این امکان را می‌دهند که کد را سازماندهی کنیم، از تکرار جلوگیری کنیم و خوانایی برنامه را افزایش دهیم. در ادامه، با توابع و ویژگی‌های مختلف آن‌ها آشنا می‌شویم.

1 تعریف تابع در C++

ساختار کلی یک تابع:

```
{ نوع_بازگشتی نام_تابع(پارامترها)  
// بدن تابع  
return لازم نیست void مقدار; // اگر نوع بازگشتی  
}
```

مثال تابعی که دو عدد را جمع می‌کند:

```
#include <iostream>  
using namespace std;  
  
int sum(int a, int b) { // تعریف تابع  
    return a + b;  
}  
  
int main() {  
    cout << sum(3, 5); // فراخوانی تابع  
}
```

خروجی:

8

2 تابع بدون مقدار بازگشتی (void)

مثال:

Edit ⌛ Copy ⌂

```
void printMessage() {
    cout << "Hello, C++!";
}

int main() {
    printMessage(); // خروجی: Hello, C++
}
```

3 مقدار پیشفرض برای پارامترها

❖ می‌توان مقدار پیشفرض برای ورودی‌ها تعیین کرد:

Edit ⌛ Copy ⌂

```
void greet(string name = "Guest") {
    cout << "Hello, " << name << "!";
}

int main() {
    greet();      // خروجی: Hello, Guest!
    greet("Ali"); // خروجی: Hello, Ali!
}
```

4 تابع بازگشتی (Recursive Function)

❖ مثال: فاکتوریل عدد

Edit ⌛ Copy ⌂

```
int factorial(int n) {
    if (n == 0) return 1;
    return n * factorial(n - 1);
}

int main() {
    cout << factorial(5); // خروجی: 120
}
```

تابع با آرایه به عنوان ورودی 5

آرایه‌ها به صورت مرجع (Reference) به تابع ارسال می‌شوند:

```
void printArray(int arr[], int size) {  
    for (int i = 0; i < size; i++) {  
        cout << arr[i] << " ";  
    }  
}  
  
int main() {  
    int nums[] = {1, 2, 3, 4, 5};  
    printArray(nums, 5);  
}
```

Edit ⌚ Copy ☉

خروجی:

```
1 2 3 4 5
```

Edit ⌚ Copy ☉

تابع با مقدار بازگشته چندگانه (با استفاده از tuple و pair) 6

مثال: تابعی که کوچکترین و بزرگترین مقدار را برمی‌گرداند

```
#include <iostream>  
#include <tuple>  
using namespace std;  
  
pair<int, int> minMax(int a, int b) {  
    return {min(a, b), max(a, b)};  
}  
  
int main() {  
    pair<int, int> result = minMax(10, 5);  
    cout << "Min: " << result.first << ", Max: " << result.second;  
}
```

Edit ⌚ Copy ☉

خروجی:

Edit ⌛ Copy ⌂

Min: 5, Max: 10

7 ارسال متغیر به صورت مقدار و مرجع (pass by value & pass by reference)

رسال مقدار (pass by value): مقدار اصلی تغییر نمی‌کند

Edit ⌛ Copy ⌂

```
void changeValue(int x) {
    x = 100;
}

int main() {
    int num = 10;
    changeValue(num);
    cout << num; // خروجی: 10 (تغییر نکرد!)
}
```

رسال با مرجع (pass by reference): مقدار اصلی تغییر می‌کند

Edit ⌛ Copy ⌂

```
void changeValue(int &x) {
    x = 100;
}

int main() {
    int num = 10;
    changeValue(num);
    cout << num; // خروجی: 100
}
```

8 تابعی که یک رشته را پردازش می‌کند

مثال: تابعی که حروف کوچک را بزرگ می‌کند

Edit ⚙ Copy ☉

```
#include <iostream>
#include <cctype> // برای توابع تغییر حروف
using namespace std;

void toUpperCase(string &s) {
    for (char &c : s) c = toupper(c);
}

int main() {
    string text = "hello";
    toUpperCase(text);
    cout << text; // خروجی: HELLO
}
```

۹ تابعی برای معکوس کردن آرایه

مثال: ❤️

Edit ⚙ Copy ☉

```
void reverseArray(int arr[], int size) {
    for (int i = 0; i < size / 2; i++) {
        swap(arr[i], arr[size - i - 1]);
    }
}

int main() {
    int nums[] = {1, 2, 3, 4, 5};
    reverseArray(nums, 5);

    for (int n : nums) cout << n << " ";
}
```

خروجی: ♦

Edit ⚙ Copy ☉

5 4 3 2 1

10 تابعی که بررسی کند یک عدد اول است یا نه

Edit ⌛ Copy ⌂

```
bool isPrime(int n) {
    if (n < 2) return false;
    for (int i = 2; i * i <= n; i++) {
        if (n % i == 0) return false;
    }
    return true;
}

int main() {
    cout << isPrime(7); // 1 (عنی) خروجی: True
}
```

خلاصه‌ی توابع در C++

- ✓ تعریف تابع: نوع_بازگشتی نام_تابع(پارامترها) { بدن تابع }
- ✓ تابع بدون مقدار بازگشتی: void
- ✓ مقدار پیش‌فرض پارامترها (Recursion)
- ✓ ارسال آرایه به تابع
- ✓ چند مقدار بازگشتی با tuple و pair
- ✓ ارسال مقدار و مرجع (pass by value & reference)

ارسال و دریافت مقادیر در C++

در C++, روش‌های مختلفی برای ارسال و دریافت مقادیر بین توابع و برنامه وجود دارد. در این بخش، تمام روش‌های مهم را بررسی می‌کنیم.

1 دریافت مقدار از ورودی (cin)

دریافت مقدار عددی از کاربر و نمایش آن:

Edit ⌛ Copy ⌂

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

```
int main() {
    int num;
    cout << "Enter a number: ";
    cin >> num;
    cout << "You entered: " << num;
}
```

وروودی: 10

خروجی: You entered: 10

دريافت چند مقدار همزمان:

Edit ⌚ Copy ☉

```
int a, b;
cin >> a >> b;
```

دريافت مقدار از ورودی و ذخيره در يك رشته:

Edit ⌚ Copy ☉

```
string name;
cin >> name;
```

نکته: **cin** تنها یک کلمه را دریافت می‌کند. اگر بخواهید یک جمله کامل دریافت کنید، باید از **getline** استفاده کنید:

Edit ⌚ Copy ☉

```
string sentence;
getline(cin, sentence);
```

ارسال مقدار به خروجی (**cout**) **2**

چاپ مقدار در خروجی:

```
cout << "Hello, C++!" << endl;
```

Edit ⌂ Copy ☉

خروجی: !++Hello, C

چاپ چند مقدار به صورت همزمان:

```
cout << "Sum of " << a << " and " << b << " is " << (a + b);
```

Edit ⌂ Copy ☉

فرمت دهی خروجی با `setprecision` و `setw`

```
#include <iostream>
#include <iomanip> // برای setw و setprecision
using namespace std;

int main() {
    double pi = 3.1415926535;
    cout << fixed << setprecision(2) << pi; // 3.14
}
```

Edit ⌂ Copy ☉

ارسال مقدار به تابع (Pass by Value & Pass by Reference) 3

ارسال مقدار به تابع (Pass by Value) ✓

در این روش، مقدار ورودی کپی شده و تغییرات داخل تابع روی مقدار اصلی اعمال نمی‌شود.

```
void changeValue(int x) {
    x = 100;
}

int main() {
    int num = 10;
    changeValue(num);
    cout << num; // تغییری نکرد!
}
```

Edit ⌂ Copy ☉

ارسال مقدار با مرجع (Pass by Reference)

در این روش، مقدار اصلی به تابع ارسال شده و تغییرات روی مقدار اصلی اعمال می‌شود.

Edit  Copy 

```
void changeValue(int &x) {
    x = 100;
}

int main() {
    int num = 10;
    changeValue(num);
    cout << num; // 100
}
```

ارسال آرایه به تابع

آرایه‌ها همیشه به صورت مرجع (Reference) ارسال می‌شوند و مقدار اصلی تغییر می‌کند:

Edit  Copy 

```
void doubleArray(int arr[], int size) {
    for (int i = 0; i < size; i++) {
        arr[i] *= 2;
    }
}

int main() {
    int nums[] = {1, 2, 3, 4, 5};
    doubleArray(nums, 5);

    for (int n : nums) cout << n << " "; // 10 8 6 4 2
}
```

ارسال رشته (String) به تابع

ارسال مقدار به صورت کپی (تغییرات در مقدار اصلی تأثیر ندارد)

Edit ⚙ Copy ☉

```
void changeString(string str) {
    str = "New Text";
}

int main() {
    string text = "Old Text";
    changeString(text);
    cout << text; // خروجی: Old Text (تغییری نکرد)
}
```

◆ ارسال مقدار به صورت مرجع (تغییرات روی مقدار اصلی اعمال می‌شود)

Edit ⚙ Copy ☉

```
void changeString(string &str) {
    str = "New Text";
}

int main() {
    string text = "Old Text";
    changeString(text);
    cout << text; // خروجی: New Text
}
```

◆ 6 ارسال مقدار بازگشتی از تابع (return)

◆ مثال: تابعی که دو عدد را جمع می‌کند و مقدار را برمی‌گرداند

Edit ⚙ Copy ☉

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

int main() {
    cout << sum(3, 7); // خروجی: 10
}
```

مثال: تابعی که بزرگترین مقدار را برمی‌گرداند

Edit ⌛ Copy ⌂

```
int findMax(int a, int b) {
    return (a > b) ? a : b;
}

int main() {
    cout << findMax(5, 10); // 10 : خروجی
}
```

7 چند مقدار بازگشتی با tuple و pair

برگرداندن دو مقدار با pair

Edit ⌛ Copy ⌂

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

pair<int, int> minMax(int a, int b) {
    return {min(a, b), max(a, b)};
}

int main() {
    pair<int, int> result = minMax(10, 5);
    cout << "Min: " << result.first << ", Max: " << result.second;
}
```

برگرداندن چند مقدار با tuple

Edit ⌛ Copy ⌂

```
#include <iostream>
#include <tuple>
using namespace std;

tuple<int, double, string> getData() {
    return {10, 3.14, "Hello"};
}
```

```

int main() {
    int x;
    double y;
    string z;
    tie(x, y, z) = getData();

    cout << x << " " << y << " " << z;
}

```

خروجی:

Edit ⌛ Copy ☉

10 3.14 Hello

خلاصه ارسال و دریافت مقادیر در C++

- ✓ دریافت مقدار از ورودی: `cin, getline`
- ✓ ارسال مقدار به خروجی: `cout, setw, setprecision`
- ✓ ارسال مقدار به تابع: `pass by reference` و `pass by value`
- ✓ ارسال آرایه و رشته به تابع
- ✓ مقدار بازگشتی از تابع با `return`
- ✓ برگرداندن چند مقدار با `tuple` و `pair`

C++ (return) مقدار بازگشتی در

در زبان C++, مقدار بازگشتی (Return Value) مقدار یا نتیجه‌ای است که یک تابع پس از اجرا برمی‌گرداند. از دستور `return` برای برگرداندن مقدار از تابع استفاده می‌شود.

1 مقدار بازگشتی در توابع عددی

مثال ساده: تابعی که مجموع دو عدد را برمی‌گرداند 

```

#include <iostream>
using namespace std;

int sum(int a, int b) {
    return a + b; // مقدار a + b می‌شود
}

```

```

int main() {
    int result = sum(5, 10);
    cout << "Sum is: " << result; // خروجی: Sum is: 15
}

```

عملکرد:

- مقدار در `return a + b` محاسبه شده و به تابع `main()` بازگردانده می‌شود.
- مقدار بازگشتی در `result` ذخیره و نمایش داده می‌شود.

2 مقدار بازگشتی در توابع شرطی

مثال: تابعی که بزرگ‌ترین مقدار بین دو عدد را برمی‌گرداند

```

int max(int a, int b) {
    if (a > b)
        return a; // بازگرداند a بزرگ‌تر باشد، مقدار a اگر
    else
        return b; // بازگرداند b در غیر این صورت مقدار
}

```

```

int main() {
    cout << "Max is: " << max(10, 20); // خروجی: Max is: 20
}

```

Edit ⚙ Copy ☉

3 مقدار بازگشتی در توابعی که مقدار بولی برمی‌گردانند (`bool`)

مثال: بررسی عدد زوج یا فرد

```

bool isEven(int num) {
    return num % 2 == 0; // اگر num زوج باشد، برمی‌گرداند true
}

int main() {
    cout << isEven(10); // 1 (یعنی) خروجی: true
    cout << isEven(7); // 0 (یعنی) خروجی: false
}

```

Edit ⚙ Copy ☉

◆ مقدار بازگشتی `false` یا `true` است.

4 مقدار بازگشتی در توابعی که رشته (string) برمی‌گردانند

مثال: تابعی که سلام را با نام ترکیب کرده و برمی‌گرداند 

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

string greet(string name) {
    return "Hello, " + name + "!" // مقدار رشته جدید بازگردانده می‌شود
}

int main() {
    cout << greet("Ali"); // خروجی: Hello, Ali!
}
```

Edit ⌛ Copy ⌂

5 مقدار بازگشتی در آرایه‌ها

در C++ نمی‌توان آرایه را مستقیماً بازگرداند، ولی می‌توان اشاره‌گر (Pointer) یا `vector` را برگرداند. 

روش ۱: استفاده از اشاره‌گر (Pointer)

```
int* getArray() {
    static int arr[3] = {1, 2, 3}; // آدرس آرایه باید static باشد
    return arr; // آدرس آرایه را برمی‌گرداند
}

int main() {
    int* myArr = getArray();
    cout << myArr[0] << " " << myArr[1] << " " << myArr[2]; // خروجی: 3 2 1
}
```

Edit ⌛ Copy ⌂

چرا ؟ static

چون متغیرهای محلی پس از پایان تابع از بین می‌روند، برای جلوگیری از حذف شدن آرایه، باید static باشد.

روش ۳: استفاده از vector ✓

Edit ⌐ Copy ☉

```
#include <vector>
vector<int> getVector() {
    return {1, 2, 3};
}

int main() {
    vector<int> myVec = getVector();
    cout << myVec[0] << " " << myVec[1] << " " << myVec[2]; // خروجی: 3 2 1
}
```

استفاده از vector ایمن‌تر است.

۶ مقدار بازگشتی چندگانه (با tuple و pair)

گاهی لازم است بیش از یک مقدار را از تابع برگردانیم.

برگرداندن دو مقدار با pair ✓

Edit ⌐ Copy ☉

```
#include <iostream>
#include <utility> // برای استفاده از pair
using namespace std;

pair<int, int> minMax(int a, int b) {
    return {min(a, b), max(a, b)};
}

int main() {
    pair<int, int> result = minMax(5, 10);
    cout << "Min: " << result.first << ", Max: " << result.second;
    // خروجی: Min: 5, Max: 10
}
```

مقدارهای برگردانده شده را نشان می‌دهند.

برگرداندن چند مقدار با tuple

Edit ⌛ Copy ⌂

```
#include <iostream>
#include <tuple> // برای استفاده از tuple
using namespace std;

tuple<int, double, string> getData() {
    return {10, 3.14, "Hello"};
}

int main() {
    int x;
    double y;
    string z;
    tie(x, y, z) = getData(); // دریافت مقدارها
    cout << x << " " << y << " " << z;
    // 3.14 10 Hello
}
```

از tie(x, y, z) برای گرفتن مقدارهای tuple استفاده شده است.

7 مقدار بازگشتی void (بدون مقدار)

اگر تابع قرار نیست مقدار بازگرداند، باید void باشد: 

Edit ⌛ Copy ⌂

```
void sayHello() {
    cout << "Hello, world!";
}

int main() {
    sayHello(); // خروجی: Hello, world!
}
```

یعنی تابع هیچ مقدار بازگشتی ندارد.

❖ خلاصه‌ی مقدار بازگشته در C++

✓ عدد برمی‌گردانیم:
int func() { return 10; }

✓ مقدار شرطی برمی‌گردانیم:
bool func() { return true; }

✓ رشته برمی‌گردانیم:
string func() { return "Hi"; }

✓ آرایه برمی‌گردانیم:
vector<int> func() { return {1,2,3}; }

✓ چند مقدار برمی‌گردانیم:
vector<pair<int, int>>

- برای دو مقدار <pair<int, int>>
- برای چند مقدار <tuple<int, double, string>>
- ✓ تابع بدون مقدار بازگشتی: void

❖ متغیرهای سراسری و محلی در C++

در زبان C++, متغیرها بر اساس دامنه (Scope) و طول عمر (Lifetime) به دو نوع اصلی تقسیم می‌شوند:

1 متغیرهای محلی (Local Variables)

2 متغیرهای سراسری (Global Variables)

- ♦ دامنه (Scope) یعنی در کجا برنامه می‌توان به متغیر دسترسی داشت.
- ♦ طول عمر (Lifetime) یعنی چه مدت متغیر در حافظه باقی می‌ماند.

1 متغیرهای محلی (Local Variables)

- ❖ متغیرهای محلی فقط در همان بلوک {} که تعریف شده‌اند قابل دسترسی هستند.
- ❖ وقتی اجرای برنامه از بلوک خارج شود، متغیر از بین می‌رود.
- ♦ مثال: متغیر محلی درون تابع

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

void myFunction() {
    int x = 10; // متغیر محلی
    cout << "x inside function: " << x << endl;
}

int main() {
    myFunction();
    // cout << x; // ❌ در این قسمت ناشناخته است x: خطای
}
```

Edit ⚙ Copy ☉

فقط درون `() myFunction` قابل استفاده است. ✓

خارج از `() myFunction` ، این متغیر وجود ندارد. ✓

مثال: متغیر محلی درون حلقه ◆

Edit ⌂ Copy ⌂

```
for (int i = 0; i < 5; i++) {  
    cout << i << " "; // درون حلقه معتبر است i مقدار  
}  
// cout << i; // ✗ خارج از حلقه تعریف نشده است i: خطای
```

فقط داخل حلقه `for` `i` معتبر است. ✓

متغیرهای سراسری (Global Variables) 2

متغیرهای سراسری خارج از تمام توابع تعریف می‌شوند و در همه جای برنامه قابل دسترسی هستند. ✨

طول عمر آنها تا پایان اجرای برنامه ادامه دارد. ✨

مثال: متغیر سراسری ◆

Edit ⌂ Copy ⌂

```
#include <iostream>  
using namespace std;  
  
int globalVar = 100; // متغیر سراسری  
  
void myFunction() {  
    cout << "Global variable: " << globalVar << endl;  
}  
  
int main() {  
    cout << "Global variable in main: " << globalVar << endl;  
    myFunction();  
}
```

در هر دو تابع `() myFunction` و `() main` `globalVar` در دسترس است. ✓

⚠ تفاوت‌های کلیدی بین متغیرهای محلی و سراسری

متغیر سراسری (Global)	متغیر محلی (Local)	ویژگی
خارج از تمام توابع	داخل یک تابع یا بلوک {}	محل تعریف
در تمام برنامه	فقط داخل همان تابع یا بلوک	(Scope) دامنه
تا پایان برنامه	تا زمانی که تابع اجرا می‌شود	(Lifetime) طول عمر
(Data Segment)	(Stack) در حافظه استک	حافظه
قابل دسترسی از هر تابع <input checked="" type="checkbox"/>	فقط در همان تابع یا بلوک <input checked="" type="checkbox"/>	دسترسی توسط توابع دیگر

3 اولویت متغیرهای سراسری و محلی

- اگر یک متغیر محلی و سراسری همنام باشند، متغیر محلی اولویت دارد.
- مثال: اولویت متغیر محلی 

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

int num = 50; // متغیر سراسری

int main() {
    int num = 10; // متغیر محلی (اولویت دارد)
    cout << "Local num: " << num << endl; // مقدار 10 چاپ می‌شود
    cout << "Global num: " << ::num << endl; // مقدار 50 چاپ می‌شود
}
```

Edit ⌛ Copy ⌂

استفاده از :: (عملگر محدوده سراسری) برای دسترسی به متغیر سراسری. 

4 متغیرهای static (ثابت)

متغیرهای static مقدارشان را حفظ می‌کنند، حتی بعد از خروج از تابع. 

مثال: متغیر static در تابع 

Edit ⌂ Copy ☉

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

void counter() {
    static int count = 0; // مقدار حفظ می‌شود
    count++;
    cout << "Count: " << count << endl;
}

int main() {
    counter(); // خروجی: Count: 1
    counter(); // خروجی: Count: 2
    counter(); // خروجی: Count: 3
}
```

بدون `count`، مقدار `static` همیشه از ۰ شروع می‌شد. ✓

5 متغیرهای `extern` (متغیر خارجی)

✗ متغیر `extern` زمانی استفاده می‌شود که بخواهیم از یک متغیر سراسری که در فایل دیگری تعریف شده استفاده کنیم.

♦ مثال: استفاده از `extern` ✗

Edit ⌂ Copy ☉

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

extern int num; // اعلام وجود متغیر سراسری (در فایل دیگری تعریف شده)

int main() {
    cout << num; // مقدار متغیر سراسری جاپ می‌شود
}
```

✓ متغیر `num` باید در فایل جداگانه‌ای تعریف شود.

- ✓ متغیرهای محلی (Local) درون یک تابع یا بلوک {} تعریف می‌شوند و بعد از خروج از آن از بین می‌روند.
- ✓ متغیرهای سراسری (Global) خارج از تمام توابع تعریف می‌شوند و تا پایان برنامه در دسترس‌اند.
- ✓ اگر نام متغیر محلی و سراسری یکسان باشد، متغیر محلی اولویت دارد.
- ✓ با استفاده از :: (عملگر محدوده سراسری) می‌توان به متغیر سراسری دسترسی داشت.
- ✓ متغیر static مقدار خود را حفظ می‌کند، حتی بعد از خروج از تابع.
- ✓ متغیر extern برای استفاده از متغیرهای تعریف شده در فایل‌های دیگر به کار می‌رود.

◆ تعریف و مقداردهی اشاره‌گرها در C++

یک اشاره‌گر (Pointer) متغیری است که آدرس یک متغیر دیگر را در خود ذخیره می‌کند. با استفاده از اشاره‌گرها می‌توان به طور مستقیم حافظه را مدیریت کرد.

◆ ۱. تعریف یک اشاره‌گر

ساختار کلی تعریف:

Edit ⓘ Copy ⌂

```
;نوع_داده *نام_اشاره‌گر
```

- ◆ علامت * نشان‌دهنده‌ی یک اشاره‌گر است.
- ◆ نوع_داده باید همان نوع متغیری باشد که اشاره‌گر به آن اشاره می‌کند.

◆ مثال: تعریف اشاره‌گر

Edit ⓘ Copy ⌂

```
int *ptr; // int
double *dptr; // double
```

◆ این اشاره‌گرها هنوز مقداردهی نشده‌اند و مقدار نامعتبر (garbage) دارند.

◆ 2. مقداردهی اشارهگر با آدرس یک متغیر

برای مقداردهی یک اشارهگر، باید آدرس یک متغیر را در آن ذخیره کنیم.
♦ برای دریافت آدرس یک متغیر از `&` (آمپرساند) استفاده می‌کنیم.

مثال: مقداردهی اشارهگر ✓

```
int x = 10; // متغیر معمولی
int *ptr = &x; // مقدارش برابر با آدرس ptr اشارهگر است
```

```
cout << "آدرس x: " << &x << endl;
cout << "آدرس ptr: " << ptr << endl;
cout << "از طریق اشارهگر x مقدار " << *ptr << endl;
```

Edit ⌂ Copy ☒

- ♦ آدرس متغیر `x` را برمی‌گرداند.
- ♦ آدرس `x` را ذخیره می‌کند.
- ♦ مقدار متغیری که اشارهگر به آن اشاره می‌کند را نشان می‌دهد.

خروجی (مثال فرضی) ✅

```
آدرس x: 0x61ff08
آدرس ptr: 0x61ff08
از طریق اشارهگر: 10 x مقدار
```

Edit ⌂ Copy ☒

◆ 3. تغییر مقدار متغیر از طریق اشارهگر

با استفاده از عملگر `*` (dereference) می‌توان مقدار یک متغیر را از طریق اشارهگر تغییر داد.
با استفاده از عملگر `*` (dereference) می‌توان مقدار یک متغیر را از طریق اشارهگر تغییر داد.

مثال: تغییر مقدار از طریق اشارهگر ✓

```
int x = 10;
int *ptr = &x; // ptr از x اشاره می‌کند
```

Edit ⌂ Copy ☒

```
*ptr = 20; // مقدار x را تغییر می‌دهیم ×
```

Edit ⌂ Copy ⌂

```
cout << " مقدار x: " << x << endl; // 20 خروجی:
```

مقدار x به ۲۰ تغییر می‌کند، زیرا ptr به x اشاره دارد.

4. مقداردهی nullptr به اشارهگر

اگر اشارهگری هنوز به جایی اشاره ندارد، مقدار آن را nullptr قرار دهید تا از ارجاع‌های نامعتبر جلوگیری شود.

```
int *ptr = nullptr; // اشارهگر اولیه به اشارهگر
```

Edit ⌂ Copy ⌂

5. اشارهگرهای چندگانه

یک اشارهگر می‌تواند آدرس یک اشارهگر دیگر را نیز نگه دارد (اشارهگر به اشارهگر).

```
int x = 10;  
int *ptr = &x;  
int **ptr2 = &ptr; // اشارهگر به اشارهگر
```

Edit ⌂ Copy ⌂

آدرس ptr را نگه می‌دارد که خودش به x اشاره دارد.

جمع‌بندی

- تعریف اشارهگر با *
- مقداردهی با & (آدرس متغیر)
- تغییر مقدار متغیر از طریق *
- مقداردهی اولیه با nullptr برای امنیت
- امکان استفاده از اشارهگر به اشارهگر

◆ عملیات روی اشارهگرها در C++

اشارهگرها در C++ علاوه بر نگهداری آدرس‌ها، قابلیت انجام عملیات ریاضی و مدیریت حافظه را نیز دارند. در اینجا با انواع عملیات روی اشارهگرها آشنا می‌شویم.

◆ 1. مقداردهی و دسترسی به مقدار اشاره شده

با استفاده از `&` (آدرس) و `*` (دسترسی به مقدار) می‌توان یک اشارهگر را مقداردهی و مقدار آن را تغییر داد.

Edit ⌂ Copy ⌂

```
int x = 10;
int *ptr = &x; // در x ذخیره‌ی آدرس ptr
cout << "نمایش آدرس x: " << ptr << endl; // x
cout << "از طریق اشاره‌گر x مقدار " << *ptr << endl; // مقدار x
```

آدرس `x` را دارد.

مقدار `x` را نمایش می‌دهد.

◆ 2. عملیات حسابی روی اشارهگرها

اشارهگرها را می‌توان در عملیات ریاضی مانند جمع و تفریق استفاده کرد.

افزایش و کاهش اشارهگر (`++` و `--`) ✓

Edit ⌂ Copy ⌂

```
int arr[] = {10, 20, 30};
int *ptr = arr; // اولین عنصر آرایه

cout << *ptr << endl; // 10
ptr++; // عنصر بعدی حرکت می‌کند
cout << *ptr << endl; // 20
ptr--;
cout << *ptr << endl; // 10 (برگشت به مقدار اول)
```

اشارهگر را به عنصر بعدی آرایه منتقل می‌کند.

اشارهگر را به عنصر قبلی برمی‌گرداند.

جمع و تفریق مقدار عددی با اشارهگر (+ و -)

Edit ⌛ Copy ⌂

```
int arr[] = {10, 20, 30, 40, 50};  
int *ptr = arr;  
  
cout << *(ptr + 2) << endl; // مقدار arr[2] یعنی 30  
cout << *(ptr + 4) << endl; // مقدار arr[4] یعنی 50
```

به دو خانه بعد در آرایه اشاره می‌کند.

3. مقایسه اشارهگرها (<, , >, , ==, , !=)

دو اشارهگر را می‌توان مقایسه کرد تا مشخص شود آیا به یک آدرس اشاره دارند یا خیر.

Edit ⌛ Copy ⌂

```
int a = 5, b = 10;  
int *ptr1 = &a, *ptr2 = &b;  
  
if (ptr1 == ptr2)  
    cout << ".هر دو اشارهگر به یک متغیر اشاره دارند"  
else  
    cout << ".اشارهگرها به متغیرهای متفاوتی اشاره دارند"
```

در اینجا `ptr1` و `ptr2` آدرس‌های متفاوتی دارند.

4. تفاضل دو اشارهگر

می‌توان دو اشارهگر را از هم کم کرد تا اختلاف آدرس‌ها را بدست آورد.

Edit ⌛ Copy ⌂

```
int arr[] = {10, 20, 30, 40, 50};  
int *p1 = &arr[1]; // اشاره به arr[1] یعنی 20  
int *p2 = &arr[4]; // اشاره به arr[4] یعنی 50  
  
cout << "تعداد خانه‌های بین p1 و p2: " << (p2 - p1) << endl;
```

این کد تعداد خانه‌های بین دو اشارهگر در آرایه را نشان می‌دهد.

◆ 5. اشارهگرهای ثابت (const Pointers)

گاهی نیاز داریم اشارهگر را فقط خواندنی کنیم یا مقدار اشاره شده را تغییر ندهیم.

اشارهگر به مقدار ثابت (مقدار قابل تغییر نیست) ✓

Edit ⌂ Copy ☉

```
const int x = 10;
const int *ptr = &x; // مقدار x را تغییر نمی‌تواند بکند
```

را نمی‌توان تغییر داد x خطا: مقدار X

اشارهگر ثابت (آدرس قابل تغییر نیست) ✓

Edit ⌂ Copy ☉

```
int x = 10, y = 20;
int *const ptr = &x; // ptr را همیشه باید به x اشاره کند
```

خطا: آدرس قابل تغییر نیست X

تغییر می‌کند x مجاز است، مقدار ✓

◆ 6. اشارهگرهای خالی (nullptr)

اگر اشارهگری مقداردهی نشده باشد، مقدار نامعتبر دارد و ممکن است باعث کرش برنامه شود.

راه حل: مقدار nullptr را مقداردهی اولیه کنید. ♦

Edit ⌂ Copy ☉

```
int *ptr = nullptr;
if (ptr == nullptr)
    cout << ".اشارهگر مقداردهی نشده است" ;
```

استفاده از `&` برای گرفتن آدرس متغیر 

استفاده از `*` برای دسترسی به مقدار 

عملیات ریاضی (`-`, `+`, `--`, `++`) روی اشاره‌گرها 

مقایسه‌ی اشاره‌گرها (`<`, `>`, `=!`, `==`) 

تفاضل اشاره‌گرها برای محاسبه فاصله 

استفاده از `nullptr` برای جلوگیری از ارجاع‌های نامعتبر 

استفاده از `const` برای محدود کردن تغییر مقدار یا آدرس 

◆ تخصیص و آزادسازی حافظه در C++

در C++, حافظه به دو دسته‌ی استاتیک (Static) و داینامیک (Dynamic) تقسیم می‌شود.

 حافظه‌ی استاتیک: در زمان کامپایل مشخص می‌شود. (مثلاً متغیرهای معمولی و آرایه‌های ثابت)

 حافظه‌ی داینامیک: در زمان اجرا (Run-Time) تخصیص داده شده و توسط برنامه‌نویس آزاد می‌شود. (با `new` و `(delete`

◆ چرا از حافظه‌ی داینامیک استفاده کنیم؟

♦ زمانی که اندازه‌ی متغیرها در زمان کامپایل مشخص نیست، باید از حافظه‌ی داینامیک استفاده کرد.

♦ برای مدیریت بهتر حافظه، مخصوصاً در ساختارهای داده‌ای مثل لیست‌های پیوندی، درخت‌ها و گراف‌ها.

◆ 1. تخصیص حافظه‌ی داینامیک برای یک متغیر (`new`)

برای تخصیص حافظه‌ی داینامیک از `new` استفاده می‌کنیم.

Edit ⌛ Copy ⌂

```
ایجاد یک عدد صحیح در حافظه‌ی داینامیک
int *ptr = new int; // ایجاد می‌کند و آدرس آن را به ptr می‌دهد.
*ptr = 10;           // متغیر ایجادشده
cout << *ptr << endl; // خروجی: 10
delete ptr;         // آزادسازی حافظه
```

♦ یک فضای جدید در Heap ایجاد می‌کند و آدرس آن را به `ptr` می‌دهد. 

♦ در پایان، همیشه باید حافظه را با `delete` آزاد کرد تا از نشت حافظه (Memory Leak) جلوگیری شود. 

◆ 2. تخصیص حافظه‌ی داینامیک برای آرایه (`[] new`)

می‌توان یک آرایه را در زمان اجرا ایجاد کرد.

Edit ⌛ Copy ⌂

```
int *arr = new int[5]; // تخصیص آرایه ۵ عنصری
for (int i = 0; i < 5; i++)
    arr[i] = (i + 1) * 10;

for (int i = 0; i < 5; i++)
    cout << arr[i] << " "; // 50 40 30 20 10
آزادسازی حافظه آرایه arr
```

یک آرایه با ۵ خانه ایجاد می‌کند.

برای آرایه‌ها باید از delete [] استفاده کنیم و گزنه فقط خانه‌ی اول آزاد می‌شود.

3. تخصیص حافظه برای آرایه‌ی دو بعدی

در C++, آرایه‌های دو بعدی به روش ماتریس پویا ساخته می‌شوند.

Edit ⌛ Copy ⌂

```
int rows = 3, cols = 4;
int **matrix = new int*[rows]; // تخصیص آرایه‌ای از اشاره‌گرهای
                                // مقداردهی
for (int i = 0; i < rows; i++)
    matrix[i] = new int[cols]; // تخصیص هر ردیف

// نمایش مقدارها
for (int i = 0; i < rows; i++) {
    for (int j = 0; j < cols; j++)
        cout << matrix[i][j] << " ";
    cout << endl;
}

// آزادسازی حافظه
for (int i = 0; i < rows; i++)
    delete[] matrix[i];
delete[] matrix;
```

- ☞ هر سطر جداگانه تخصیص داده شده و در نهایت باید آزاد شود!
- ☞ ابتدا هر سطر را با `[]delete` آزاد کرده و سپس کل ماتریس را حذف می‌کنیم.

4. آزادسازی حافظه (`[]delete` و `delete`) ◆

اگر حافظه‌ی داینامیک آزاد نشود، باعث نشت حافظه (Memory Leak) خواهد شد!
برای یک متغیر:

```
int *ptr = new int(50);
delete ptr; // آزادسازی
ptr = nullptr; // جلوگیری از اشاره‌گر معلق
```

Edit ⌗ Copy ⌓

برای آرایه‌ی دو بعدی:

```
for (int i = 0; i < rows; i++)
    delete[] matrix[i];
delete[] matrix;
```

Edit ⌗ Copy ⌓

☞ بعد از `delete` بهتر است اشاره‌گر را `nullptr` کنیم تا از دسترسی نامعتبر جلوگیری شود.

5. نکات مهم در تخصیص و آزادسازی حافظه ◆

- ☞ فراموش نکنید حافظه را آزاد کنید!
- ☞ برای آرایه‌ها باید از `[]delete` استفاده شود.
- ☞ اشاره‌گر را بعد از `delete` مقدار `nullptr` بدهید.
- ☞ نشت حافظه نداشته باشید! همیشه حافظه‌ای که با `new` گرفته‌اید، باید `delete` شود.

تعريف و استفاده از `struct` در C++

struct چیست؟

(ساختار) یک نوع داده‌ای ترکیبی در C++ است که به ما اجازه می‌دهد چندین مقدار مختلف را با انواع داده‌ای متفاوت در یک متغیر ذخیره کنیم.

تعريف `struct`

برای تعریف یک ساختار، از کلمه کلیدی `struct` استفاده می‌کنیم. درون آن، متغیرهای مختلفی را تعریف می‌کنیم که به آنها اعضاء (Members) می‌گوییم.

مثال: تعریف یک `struct` برای ذخیره اطلاعات یک دانشآموز

Edit  Copy 

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

struct Student {
    string name;
    int age;
    float gpa;
};

int main() {
    Student s1; // تعریف یک متغیر از نوع Student
    s1.name = "Ali";
    s1.age = 20;
    s1.gpa = 3.8;

    cout << "Name: " << s1.name << endl;
    cout << "Age: " << s1.age << endl;
    cout << "GPA: " << s1.gpa << endl;

    return 0;
}
```

ویژگی‌های این کد: 

1. یک ساختار جدید است که سه مقدار مختلف را ذخیره می‌کند.
2. برای دسترسی به اعضای `struct` از عملگر (dot operator)  استفاده می‌کنیم.

2 مقداردهی به `struct`

می‌توان مقداردهی را به روش‌های مختلف انجام داد:

روش مقداردهی مستقیم

```
Student s2 = {"Reza", 22, 3.6};
```

Edit  Copy 

3 مقداردهی با استفاده از لیست مقداردهی (C++11)

```
Student s3{"Sara", 19, 4.0};
```

Edit  Copy 

3 ارسال `struct` به تابع

می‌توان `struct` را به یک تابع ارسال کرد.

4 ارسال `struct` به صورت مقدار (کپی می‌شود)

```
void printStudent(Student s) {  
    cout << s.name << " - " << s.age << " - " << s.gpa << endl;  
}  
  
int main() {  
    Student s1{"Ali", 20, 3.8};  
    printStudent(s1);  
}
```

مشکل این روش: اگر `struct` حجم زیادی از داده‌ها داشته باشد، ارسال آن باعث ایجاد یک کپی غیرضروری و افزایش مصرف حافظه می‌شود.

4 ارسال `struct` با استفاده از مرجع (بهینه‌تر)

برای جلوگیری از کپی غیرضروری، می‌توانیم `struct` را با مرجع (`&`) ارسال کنیم:

```
void printStudent(const Student &s) { // مرجع استفاده شده
    cout << s.name << " - " << s.age << " - " << s.gpa << endl;
}
```

مزیت این روش:

- از ایجاد کپی اضافی جلوگیری می‌شود.
- باعث می‌شود که داده‌های `struct` در تابع تغییر نکنند.

5 آرایه‌ای از `struct`

می‌توان چندین `struct` را درون یک آرایه ذخیره کرد.

مثال: ذخیره اطلاعات چند دانشآموز

```
Student students[3] = { {"Ali", 20, 3.8}, {"Sara", 19, 4.0}, {"Reza", 22, 3.6} };

for (int i = 0; i < 3; i++) {
    cout << students[i].name << " - " << students[i].age << " - " << students[i].gpa << endl;
}
```

6 تو در تو `struct`

می‌تواند شامل `struct` دیگری باشد.

مثال: ذخیره اطلاعات یک فرد و آدرس او

```
struct Address {  
    string city;  
    string street;  
};  
  
struct Person {  
    string name;  
    int age;  
    Address addr;  
};  
  
int main() {  
    Person p = {"Ali", 25, {"Tehran", "Enghelab St."}};  
    cout << p.name << " lives in " << p.addr.city << ", " << p.addr.street << endl;  
}
```

اشارهگر به struct و استفاده از - 7

می‌توان یک اشارهگر به struct تعریف کرد و با عملگر - به اعضای آن دسترسی داشت.

مثال: استفاده از اشارهگر برای دسترسی به struct

```
Student s = {"Ali", 20, 3.8};  
Student *ptr = &s;  
  
cout << ptr->name << " - " << ptr->age << " - " << ptr->gpa << endl;
```

.name.(ptr*) معادل ptr->name است.

تفاوت ++C در class و struct 8

در C++, struct مشابه class است، اما:

- اعضای public به صورت پیشفرض struct هستند.
- اعضای private به صورت پیشفرض class هستند.

مثال: تفاوت `class` و `struct`

```
struct S {  
    int x; // پیشفرض: public  
};  
  
class C {  
    int x; // پیشفرض: private  
};
```

Edit ⌂ Copy ⌂

تفاوت `++C` در `class` و `struct`

در `++C`, هر دو `class` و `struct` برای تعریف انواع داده‌ای سفارشی استفاده می‌شوند. اما تفاوت‌هایی بین آنها وجود دارد که در ادامه توضیح داده می‌شود.

۱ تفاوت اصلی: سطح دسترسی پیشفرض

: مهمترین تفاوت `class` و `struct` در `++C` این است که

- در `struct`, اعضا به صورت پیشفرض `public` هستند.
- در `class`, اعضا به صورت پیشفرض `private` هستند.

struct vs class

```
#include <iostream>  
using namespace std;  
  
struct S {  
    int x; // پیشفرض: public  
};  
  
class C {  
    int x; // پیشفرض: private  
};
```

Edit ⌂ Copy ⌂

```

int main() {
    S s;
    s.x = 10; // ✓ مشکلی ندارد (است public چون)

    C c;
    // c.x = 10; // ✗ خطأ ميدهد (است private چون)

    return 0;
}

```

استفاده از `struct` برای داده‌ها و `class` برای رفتار 2

قاعده کلی: ✓

معمولًا برای ساختارهای ساده‌ی داده‌ای که فقط شامل داده‌ها هستند، استفاده می‌شود. •

معمولًا برای شیءگرایی کردن کد و ایجاد رفتار و متدها استفاده می‌شود. •

مثال: `struct` برای داده‌ها

```

struct Point {
    int x, y;
};

```

Edit ⌂ Copy ⌂

مثال: `class` برای شیءگرایی

```

class Point {
private:
    int x, y;
public:
    void set(int a, int b) { x = a; y = b; }
    void print() { cout << x << ", " << y << endl; }
};

```

Edit ⌂ Copy ⌂

3 سطح دسترسی در `class` و `struct`

هم در `struct` و هم در `class` میتوان از `protected` و `public`, `private` استفاده کرد. اما در `struct` پیشفرض `public` و در `class`, پیشفرض `private` است.

مثال: تغییر سطح دسترسی در `struct`

```
struct Test {  
private:  
    int a;  
public:  
    void setA(int val) { a = val; }  
    int getA() { return a; }  
};
```

Edit 🔍 Copy ⌂

مثال: تغییر سطح دسترسی در `class`

```
class Test {  
    int a; // پیشفرض: private  
public:  
    void setA(int val) { a = val; }  
    int getA() { return a; }  
};
```

Edit 🔍 Copy ⌂

4 ارثبری (Inheritance)

تفاوت دیگر بین `class` و `struct` در ارثبری (Inheritance) است:

- در `struct`, ارثبری پیشفرض `public` است.
- در `class`, ارثبری پیشفرض `private` است.

struct مثال: ارث بری در

Edit  Copy 

```
struct A {  
    int x;  
};  
  
struct B : A { // ارث بری پیش فرض public  
    int y;  
};  
  
int main() {  
    B obj;  
    obj.x = 10; //  مشکلی ندارد  
    obj.y = 20;  
}
```

class مثال: ارث بری در

Edit  Copy 

```
class A {  
    int x; // private  
};  
  
class B : A { // ارث بری پیش فرض private  
    int y;  
};  
  
int main() {  
    B obj;  
    // obj.x = 10; //  خطای می دهد (چون private است)  
    obj.y = 20;  
}
```

نتیجه‌گیری: کی `class` و کی `struct` استفاده کنیم؟ ✓

<code>class</code> (کلاس)	<code>struct</code> (ساختار)	ویژگی
<code>private</code>	<code>public</code>	سطح دسترسی پیش‌فرض
برنامه‌نویسی شیء‌گرا	داده‌های ساده	مناسب برای
معمول‌آ دارد	معمول‌آ ندارد	(<code>methods</code>) رفتار
<code>private</code>	<code>public</code>	ارث‌بری پیش‌فرض

اگر فقط داده ذخیره می‌کنید (مانند `Point`, `Student`...), از `struct` استفاده کنید.

اگر رفتار (متدها) و مفهوم شیء‌گرایی نیاز دارید، از `class` استفاده کنید.

یونین (union) در C++ و کاربرد آن 📌

1 یونین (union) چیست؟

یونین (union) یک نوع داده در C++ است که مشابه `struct` عمل می‌کند، اما همه اعضای آن از یک فضای حافظه مشترک استفاده می‌کنند.

در `union`، تمام اعضا در یک آدرس حافظه ذخیره می‌شوند و فقط یکی از آنها در هر لحظه مقدار معتبر دارد.

2 تفاوت `struct` و `union`

<code>union</code>	<code>struct</code>	ویژگی
همه اعضا یک فضای مشترک دارند	هر عضو فضای جداگانه دارد	نحوه ذخیره داده
اندازه‌ی بزرگ‌ترین عضو	مجموع اندازه‌ی تمام اعضا	حجم حافظه مصرفی
فقط یک عضو در هر لحظه معتبر است	همه اعضا همزمان مقدار دارند	دسترسی به اعضا

3 تعریف `union` در C++

: `union` ساختار کلی تعریف ✓

```
union UnionName {
    DataType member1;
    DataType member2;
    // ...
};
```

Edit ⌐ Copy ⌐

مثال ساده: 

Edit ⌛ Copy ⌂

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

union Data {
    int i;
    float f;
    char ch;
};

int main() {
    Data d;

    d.i = 10;
    cout << "Integer: " << d.i << endl;

    d.f = 5.5;
    cout << "Float: " << d.f << endl; // دیگر معتبر نیست i مقدار

    d.ch = 'A';
    cout << "Char: " << d.ch << endl; // دیگر معتبر نیست f مقدار

    return 0;
}
```

خروجی: 

Edit ⌛ Copy ⌂

```
Integer: 10
Float: 5.5
Char: A
```

! نکته: مقدار `i` و `f` بعد از مقداردهی به `ch` نامعتبر می‌شود، چون همه اعضای `union` یک حافظه مشترک دارند.

در `union` ، حافظه‌ای برابر با بزرگ‌ترین عضو اختصاص داده می‌شود.

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

union Test {
    int i;
    double d;
    char ch;
};

int main() {
    cout << "Size of union: " << sizeof(Test) << " bytes" << endl;
    return 0;
}
```

Edit ⌂ Copy ☉

اگر `int` برابر 4 بایت، `double` برابر 8 بایت و `char` برابر 1 بایت باشد، خروجی:

```
Size of union: 8 bytes
```

Edit ⌂ Copy ☉

چرا؟ چون `double` بزرگ‌ترین عضو است (8 بایت).

کاربردهای `union` 5

در مواردی استفاده می‌شود که نیاز به ذخیره یک مقدار در هر لحظه داشته باشیم.
کاربردهای متداول:

1. صرفه‌جویی در حافظه (در سیستم‌های محدود مانند میکروکنترلرهای)
 2. داده‌های چندحالتی (مثلًاً یک متغیر که بسته به نیاز می‌تواند `int` یا `float` باشد)
 3. کار با حافظه در سطح پایین (مانند خواندن و نوشتن بایت‌های خام)
- مثال: استفاده از `union` برای ذخیره مقدار در چندین فرمت

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

Edit ⌂ Copy ☉

```

union Value {
    int i;
    float f;
    char str[4];
};

int main() {
    Value v;

    v.i = 65;
    cout << "Integer: " << v.i << endl;

    v.f = 4.5;
    cout << "Float: " << v.f << endl;

    v.str[0] = 'A';
    v.str[1] = 'B';
    v.str[2] = 'C';
    v.str[3] = '\0';
    cout << "String: " << v.str << endl;

    return 0;
}

```

خروجی:

Edit ⚙ Copy ☉

```

Integer: 65
Float: 4.5
String: ABC

```

نکته: مقدار `i` و `f` بعد از مقداردهی به `str` نامعتبر می‌شود.

۶ تفاوت `struct` و `union` در استفاده واقعی

اگر می‌خواهید هم‌زمان چندین مقدار ذخیره کنید، از `struct` استفاده کنید.

اگر فقط یک مقدار در هر لحظه لازم دارید و می‌خواهید حافظه را بهینه مصرف کنید، از `union` استفاده کنید.

union و struct مقایسه

Edit ⚙ Copy ☉

```

#include <iostream>
using namespace std;

```

```

struct S {
    int i;
    float f;
};

union U {
    int i;
    float f;
};

int main() {
    cout << "Size of struct: " << sizeof(S) << endl; // i و f مجموع اندازه
    cout << "Size of union: " << sizeof(U) << endl; // فقط بزرگترین عضو

    return 0;
}

```

خروجی (اگر `float = 4` بایت باشد):

Edit ⌛ Copy ⌂

```

Size of struct: 8
Size of union: 4

```

يعني `union` حافظه‌ی کمتری نسبت به `struct` مصرف می‌کند.

نتیجه‌گیری ✓

union	struct	ویژگی
همه اعضای یک فضای مشترک دارند	هر عضو فضای جداگانه دارد	نحوه ذخیره داده
اندازه‌ی بزرگترین عضو	مجموع اندازه‌ی تمام اعضای	حجم حافظه مصرفی
فقط یک عضو در هر لحظه معتبر است	همه اعضای هم‌زمان مقدار دارند	دسترسی به اعضای
ذخیره یک مقدار متغیر با صرفه‌جویی در حافظه	ذخیره چند مقدار هم‌زمان	مناسب برای

اگر بخواهید چند مقدار را هم‌زمان ذخیره کنید، `struct` انتخاب بهتری است.

اگر فقط یکی از اعضای در هر لحظه مقدار داشته باشد و می‌خواهید حافظه را بهینه کنید، `union` استفاده کنید.

خواندن و نوشتن در فایل‌ها در C++

در C++, برای کار با فایل‌ها از فایل‌استریم‌ها (File Streams) استفاده می‌شود که در هدر <fstream> قرار دارند.

1 انواع کلاس‌های کار با فایل‌ها

کاربرد	کلاس
خواندن از فایل (Input File Stream)	ifstream
نوشتن در فایل (Output File Stream)	ofstream
هم خواندن و هم نوشتن در فایل (File Stream)	fstream

برای استفاده از این کلاس‌ها باید هدر <fstream> را در برنامه قرار دهید:

Edit ⌐ Copy ☒

```
#include <fstream>
```

2 نوشتن در فایل (ofstream)

برای نوشتن در فایل:

1. از کلاس ofstream استفاده کنید.

2. فایل را باز کنید (با open() یا مستقیم در سازنده).

3. اطلاعات را در فایل بنویسید.

4. فایل را ببندید (close()).

♦ مثال: ایجاد و نوشتن در فایل

Edit ⌐ Copy ☒

```
#include <iostream>
#include <fstream>
using namespace std;
```

```

int main() {
    ofstream file("example.txt"); // ایجاد یا باز کردن فایل برای نوشتن

    if (!file) {
        cout << "خطا در باز کردن فایل" << endl;
        return 1;
    }

    file << "Hello, World!\n"; // نوشتن در فایل
    file << "This is a test file.\n";

    file.close(); // بستن فایل

    cout << ".اطلاعات در فایل ذخیره شد" << endl;
    return 0;
}

```

نتیجه: اگر `example.txt` وجود نداشته باشد، ایجاد شده و متن در آن ذخیره می‌شود.

۳ خواندن از فایل (ifstream)

برای خواندن از فایل:

۱. از کلاس `ifstream` استفاده کنید.

۲. فایل را باز کنید (`open()`) یا مستقیم در سازنده).

۳. اطلاعات را بخوانید (`<<` یا `getline()`).

۴. فایل را ببندید (`close()`).

♦ مثال: خواندن کل محتوای فایل

```

#include <iostream>
#include <fstream>
using namespace std;

int main() {
    ifstream file("example.txt"); // باز کردن فایل برای خواندن

```

Edit ⌛ Copy ⌂

```

if (!file) {
    cout << "خطا در باز کردن فایل" << endl;
    return 1;
}

string line;
while (getline(file, line)) { // خواندن خط به خط
    cout << line << endl;
}

file.close(); // بستن فایل
return 0;
}

```

نتیجه: محتوای example.txt در خروجی نمایش داده می‌شود.

۴ خواندن و نوشتن همزمان (fstream)

برای خواندن و نوشتن همزمان، از `fstream` استفاده کنید.

♦ مثال: باز کردن فایل در حالت خواندن و نوشتن

```

#include <iostream>
#include <fstream>
using namespace std;

int main() {
    fstream file("example.txt", ios::in | ios::out | ios::app);

    if (!file) {
        cout << "خطا در باز کردن فایل" << endl;
        return 1;
    }

    file << "Adding a new line.\n"; // نوشتن در فایل

    file.seekg(0); // رفتن به ابتدای فایل برای خواندن
}

```

Edit ⌛ Copy ⌂

```

string line;
while (getline(file, line)) {
    cout << line << endl;
}

file.close();
return 0;
}

```

نتیجه: یک خط جدید به انتهای example.txt اضافه می‌شود و سپس کل محتوای فایل خوانده و نمایش داده می‌شود.

حالات‌های باز کردن فایل (ios modes) 5

توضیح	حالت
باز کردن برای خواندن	ios::in
باز کردن برای نوشتن (اگر فایل موجود باشد، محتوا حذف می‌شود)	ios::out
باز کردن برای افزودن به انتهای فایل	ios::app
حذف محتوای فایل هنگام باز کردن	ios::trunc
باز کردن فایل در حالت باینری	ios::binary

مثال: باز کردن فایل برای نوشتن بدون حذف محتوا

Edit ⌛ Copy ⌂

```
fstream file("example.txt", ios::app);
```

مثال: باز کردن فایل برای خواندن و نوشتن

Edit ⌛ Copy ⌂

```
fstream file("example.txt", ios::in | ios::out);
```

بررسی باز شدن فایل 6

همیشه بررسی کنید که آیا فایل به درستی باز شده است یا خیر:

Edit ⌛ Copy ⌂

```
if (!file) {  
    cout << "خطا در باز کردن فایل" << endl;  
}
```

۷ حذف فایل و تغییر نام (remove() و rename())

◆ حذف فایل

Edit ⌛ Copy ⌂

```
#include <iostream>  
#include <cstdio> // برای remove()  
using namespace std;  
  
int main() {  
    if (remove("example.txt") == 0)  
        cout << "فایل حذف شد." << endl;  
    else  
        cout << "خطا در حذف فایل" << endl;  
  
    return 0;  
}
```

◆ تغییر نام فایل

Edit ⌛ Copy ⌂

```
#include <iostream>  
#include <cstdio> // برای rename()  
using namespace std;  
  
int main() {  
    if (rename("example.txt", "newfile.txt") == 0)  
        cout << ".فایل تغییر نام یافت" << endl;  
    else  
        cout << "خطا در تغییر نام فایل" << endl;  
  
    return 0;  
}
```

نتیجه‌گیری ✓

تابع	کلاس مناسب	عملیات
<code>() open() , << , write() , close</code>	<code>ofstream</code>	نوشتن در فایل
<code>() open() , >> , getline() , read() , close</code>	<code>ifstream</code>	خواندن از فایل
<code>() open() , << , >> , getline() , close</code>	<code>fstream</code>	خواندن و نوشتن همزمان
<code>remove("filename")</code>	<code>cstdio</code>	حذف فایل
<code>rename("old.txt", "new.txt")</code>	<code>cstdio</code>	تغییر نام فایل

دستکاری داده‌های فایل در C++ 🔪

برای دستکاری داده‌های فایل، باید بتوانید محتوای آن را خوانده، تغییر داده و مجددآ ذخیره کنید. این کار شامل جایگزینی، حذف، درج یا ویرایش داده‌ها در یک فایل است.

۱ جایگزینی کل محتوای فایل

اگر بخواهید کل محتوای فایل را جایگزین کنید، می‌توانید فایل را با `ios::trunc` باز کنید تا محتوای قبلی پاک شده و محتوای جدید جایگزین شود.

♦ مثال: جایگزینی کل محتوا

```
#include <iostream>
#include <fstream>
using namespace std;

int main() {
    ofstream file("data.txt", ios::trunc); // باز کردن فایل در حالت پاک کردن محتوا
    if (!file) {
        cout << "خطا در باز کردن فایل" << endl;
        return 1;
    }

    file << "Hello, this is new data!\n"; // نوشتن محتوای جدید
    file.close();

    cout << ".محتوای فایل جایگزین شد" << endl;
    return 0;
}
```

نتیجه: محتوای جدید در فایل جایگزین داده‌های قبلی می‌شود.

اضافه کردن داده به انتهای فایل 2

برای افزودن داده بدون حذف محتوای قبلی، فایل را در حالت `ios::app` باز کنید.

مثال: افزودن داده به انتهای فایل

```
#include <iostream>
#include <fstream>
using namespace std;

int main() {
    ofstream file("data.txt", ios::app); // باز کردن در حالت افزودن
    if (!file) {
        cout << "خطا در باز کردن فایل" << endl;
        return 1;
    }

    file << "Appending new data.\n"; // افزودن داده جدید
    file.close();

    cout << ".داده به انتهای فایل اضافه شد" << endl;
    return 0;
}
```

Edit ⌛ Copy 🖉

نتیجه: متن جدید در انتهای فایل اضافه می‌شود.

جایگزینی یک خط خاص در فایل 3

اگر بخواهید یک خط خاص را در فایل تغییر دهید، باید:

1. محتوای فایل را بخوانید.
2. داده‌های جدید را جایگزین کنید.
3. کل محتوا را مجدداً در فایل بنویسید.

♦ مثال: جایگزینی یک خط خاص

Edit ⌚ Copy ☉

```
#include <iostream>
#include <fstream>
#include <vector>
using namespace std;

int main() {
    ifstream file("data.txt");
    if (!file) {
        cout << "خطا در باز کردن فایل" << endl;
        return 1;
    }
```

Edit ⌚ Copy ☉

```
vector<string> lines;
string line;
int lineToReplace = 2; // میخواهیم جایگزین کنیم
int currentLine = 1;
```

```
while (getline(file, line)) {
    if (currentLine == lineToReplace) {
        lines.push_back("This is the new content for line 2.");
    } else {
        lines.push_back(line);
    }
    currentLine++;
}
```

```
file.close();
```

```
ofstream outFile("data.txt", ios::trunc);
for (const string& l : lines) {
    outFile << l << endl;
}
outFile.close();
```

```
cout << "خط مورد نظر جایگزین شد" << endl;
```

```
return 0;
}
```

نتیجه: خط دوم در فایل با محتوای جدید جایگزین می‌شود.

۴ حذف یک خط خاص از فایل

مثال: حذف یک خط از فایل

```
#include <iostream>
#include <fstream>
#include <vector>
using namespace std;

int main() {
    ifstream file("data.txt");
    if (!file) {
        cout << "خطا در باز کردن فایل" << endl;
        return 1;
    }

    vector<string> lines;
    string line;
    int lineToDelete = 2; // شماره خطی که می‌خواهیم حذف کیم
    int currentLine = 1;

    while (getline(file, line)) {
        if (currentLine != lineToDelete) {
            lines.push_back(line);
        }
        currentLine++;
    }
    file.close();

    ofstream outFile("data.txt", ios::trunc);
    for (const string& l : lines) {
        outFile << l << endl;
    }
}
```

Edit ⚙ Copy ☉

Edit ⚙ Copy ☉

4

```

outfile.close();

cout << "خط مورد نظر حذف شد" << endl;
return 0;
}

```

نتیجه: خط دوم از فایل حذف می‌شود و بقیه خطوط دست‌نویسه باقی می‌مانند.

درج یک خط در مکان مشخص 5

برای درج یک خط در یک مکان مشخص در فایل:

1. کل محتوا را بخوانید.
2. خط جدید را در مکان مشخص اضافه کنید.
3. فایل را دوباره بنویسید.

مثال: درج یک خط در مکان مشخص ◆

```
#include <iostream>
#include <fstream>
#include <vector>
using namespace std;
```

```
int main() {
    ifstream file("data.txt");
    if (!file) {
        cout << "خطا در باز کردن فایل" << endl;
        return 1;
}
```

Edit ⌂ Copy ☉

```
vector<string> lines;
string line;
int insertAt = 2; // آنجا اضافه کنیم
int currentLine = 1;
```

Edit ⌂ Copy ☉

```
while (getline(file, line)) {
    if (currentLine == insertAt) {
        lines.push_back("This is the inserted line.");
    }
    lines.push_back(line);
    currentLine++;
}
```

```

}

file.close();

ofstream outFile("data.txt", ios::trunc);
for (const string& l : lines) {
    outFile << l << endl;
}
outFile.close();

cout << "خط جدید اضافه شد." << endl;
return 0;
}

```

نتیجه: یک خط جدید در خط دوم فایل اضافه می‌شود. 

6 جستجوی یک مقدار خاص در فایل

♦ مثال: جستجوی یک کلمه در فایل

```

#include <iostream>
#include <fstream>
#include <string>
using namespace std;

int main() {
    ifstream file("data.txt");
    if (!file) {
        cout << "خطا در باز کردن فایل" << endl;
        return 1;
    }

    string wordToFind = "test";
    string line;
    int lineNumber = 1;
    bool found = false;
}

```

Edit ⌛ Copy ⌂

```

while (getline(file, line)) {
    if (line.find(wordToFind) != string::npos) {
        cout << " کلمه " << wordToFind << " در خط " << lineNumber << " پیدا شد: " << line << endl;
        found = true;
    }
    lineNumber++;
}
file.close();

if (!found) {
    cout << "کلمه مورد نظر یافت نشد" << endl;
}

return 0;
}

```

نتیجه: اگر کلمه "test" در فایل باشد، شماره خط آن نمایش داده می‌شود. 

نتیجه‌گیری

روش مناسب	عملیات
باز کردن فایل با <code>ios::trunc</code> و نوشتمن مجدد	جایگزینی کل محتوا
باز کردن فایل با <code>ios::app</code>	اضافه کردن به انتهای فایل
خواندن کل فایل، تغییر خط، و نوشتمن مجدد	ویرایش یک خط خاص
خواندن کل فایل، حذف خط، و نوشتمن مجدد	حذف یک خط خاص
خواندن کل فایل، درج خط، و نوشتمن مجدد	درج یک خط در مکان مشخص
خواندن و استفاده از <code>find()</code>	جستجو در فایل

حالا می‌توانی داده‌های فایل‌ها را به هر روشنی که بخواهی دستکاری کنی! 