دوره برنامه‌نویسی C++ پیشرفته

دانشگاه ولی عصر رفسنجان

مهندس حسین بازماندگان

جلسه سوم

مقدمه ای بر کلاس ها و اشیاء

سرفصل مطالب

- یاد آوری
- قالب کلی ساخت کلاس

یاد آوری

تابع

- تابع یک قطعه برنامه کامل است که کار معینی را انجام می‌دهد و موجب جلوگیری از برنامه نویسی تکراری در بین برنامه‌ها می شود.
- توابع در C++ دو نوع هستند:
  - توابع از پیش نوشته شده
  - توابع نوشته شده توسط برنامه‌نویس

توابع از پیش نوشته شده

توابع کتابخانه ای ریاضی

- این توابع در فایل سرایند <cmath>تعریف شده‌اند.
- تمامی این توابع یک ورودی از نوع double دریافت و به عنوان خروجی مقدار double بر می‌گردانند.

`ceil(x)`	مقدار سقف x (گرد شده)	`ceil(3.141593)` مقدار 4.0 را برمی‌گرداند
`cos(x)`	کسینوس x (به رادیان)	`cos(2)` مقدار -0.416147 را برمی‌گرداند
`exp(x)`	تابع نمایی x (در پایه e)	`exp(2)` مقدار 7.38906 را برمی‌گرداند
`fabs(x)`	قدر مطلق x	`fabs(-2)` مقدار 2.0 را برمی‌گرداند
`floor(x)`	مقدار کف x (گرد شده)	`floor(3.141593)` مقدار 3.0 را برمی‌گرداند
`log(x)`	لگاریتم طبیعی x (در پایه e)	`log(2)` مقدار 0.693147 را برمی‌گرداند
`log10(x)`	لگاریتم عمومی x (در پایه 10)	`log10(2)` مقدار 0.30103 را برمی‌گرداند
`pow(x,p)`	x به توان p	`pow(2,3)` مقدار 8.0 را برمی‌گرداند
`sin(x)`	سینوس x (به رادیان)	`sin(2)` مقدار 0.909297 را برمی‌گرداند
`sqrt(x)`	جذر x	`sqrt(2)` مقدار 1.41421 را برمی‌گرداند
`tan(x)`	تانژانت x (به رادیان)	`tan(2)` مقدار -2.18504 را برمی‌گرداند

مثال: تابع جذر sqrt()

- برنامه زیر، تابع جذر را به کار می‌گیرد.
- به دستور #include <cmath> در خط اول توجه کنید. کامپایلر برای یافتن تعریف تابع sqrt () به این خط نیاز دارد.

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

int main() {
    cout << sqrt(25);
    return 0;
}
```

توابع نوشته شده توسط برنامه‌نویس

تعریف تابع

- شما برای تعریف تابع باید چهار چیز مشخص کنید.

1. **اسم تابع**: دلخواه است، اما در زمان تعریف اسمی با معنا بگذارید
2. **کار تابع**: تابع چه وظیفه ای دارند
3. **ورودی ها**: تابع برای انجام وظیف اش به چه چیزهایی نیاز داد
4. **خروجی**: تابع بعد از انجام وظیفه اش چه مقداری را به عنوان نتیجه به شما تحویل میدهد.

تعریف تابع

- تابع مانند یک کارگر است که اسمی دارد، و وظیفه خاصی را برای شما انجام میدهد و برای انجام وظیفه اش نیاز به مواد اولیه دارد و پس اتمام وظیفه اش دستاوردش را به شما میدهد.
- شما زمانی که بخواهید از این کارگر استفاده کنید نام آن را صدا میزنید و مواد مورد نیاز آن را به آن می دهید.

ساختار تابع

- فرض کنید می خواهید تابعی تعریف کنید که عمل به توان رسانده را انجام دهد، دو عدد از شما بگیرد و عدد اول را به توان عدد دوم برساند.

1. **اسم تابع**: pow
2. **وظیفه تابع**: به توان رساندن
3. **ورودی ها**: دو عدد از نوع صحیح
4. **خروجی**: حاصل به توان رساندن

مثال 1

- برنامه ای بنویسید که بدون استفاده از تابع حاصل عبارت $3^5 + 2^4$ محاسبه کند.

ساختار تابع pow

```
int pow (int x, int n) {

}
```

بدنه تابع pow

```
int pow (int x, int n) {
  int result = 1;
  for (int i = 1; i <= n; ++i) {
    result *= x;
  }
  return result;
}
```

- کلمه return مشخص می کند خروجی یا به عبارتی نتیجه کار تابع چیست و همان را به عنوان نتیجه برمیگرداند.

محل تعریف تابع

- تابع نباید در تابع دیگری تعریف شود. می توان تابع را بالای تابع main یا در پایین تابع main تعریف کرد، اما زمانی که تابع را در پایین تابع main تعریف میکند باید قبل از تابع main امضای تابع را بنوسید.

```
int pow (int x, int n);
int main() {
    return 0;
}

int pow (int x, int n) {
  //بدنه تابع
}
```

روش استفاده از یک تابع در برنامه اصلی

- برای استفاده از یک تابع باید اسم آن را بنویسم در یک پرانتز مقابل آن ورودی های آن را مشخص کنیم.

```
int main() {
    cout << pow(2,4);
    return 0;
}
```

مثال 2

- برنامه ای بنویسید که با استفاده از تابع حاصل عبارت $2^4 + 3^5$ محاسبه کند.

تمرین 1

- تابعی به اسم rec_area بنویسید که طول و عرض یک مستطیل را گرفته و مساحت مستطیل را محاسبه کند.

تمرین 2

- تابعی به اسم c_area بنویسید که شعاع یک دایره را گرفته و مساحت دایره را محاسبه کند.

تابع بدون خروجی یا بدون ورودی

- ممکن است تابع هیچ نتیجه ای را به عنوان خروجی نداشته باشد یا برای انجام وظیفه اش نیاز به هیچ ورودی نداشته باشد مانند تابع زیر که وظیفه آن تنها چاپ کردن یک عبارت است. برای نشان دادن اینکه تابع هیچ خروجی ندارد از **void** استفاده می کنیم.

```
void say_hello() {
    cout << "Hello!" << endl;
}
```

متغیر های عمومی و محلی

- متغیر هایی که داخل از توابع تعریف شوند به عنوان **متغیر های محلی** شناخته می شوند. متغیر هایی که خارج از توابع تعریف شوند به عنوان **متغیر های عمومی** شناخته می شوند.
- متغیرهای محلی فقط برای همان تابع شناخته شده و خارج از تابع داخل توابع دیگر قابل دسترسی و تغییر نمی‌باشند اما متغیرهای سراسری در هر بخش از برنامه قابل دسترسی و تغییر می‌باشند.
- اگر داخل توابع متغیر محلی همنام با متغیر سراسری تعریف شود داخل آن تابع متغیر سراسری همنام با متغیر محلی دیگر قابل دسترسی و استفاده نخواهد بود بلکه از متغیر محلی استفاده می‌شود.

متغیر های عمومی و محلی

```
int a; متغیر عمومی
void f1() {
    int b; متغیر محلی
}
void f2() {
    int c; متغیر محلی
}
```

نحوه ارسال متغیر به تابع

کلمه کلیدی const

آرگومان پیشفرض

سربارگذاری توابع

تعریف کلاس

اسم کلاس

```
class Student {

};
```

- فعلا کلاس ها را بالای تابع main تعریف میکنم

تعریف ویژگی ها

```
class Student {
    public:
    string name;
    int age;
    double gpa;
};
```

- فعلا public: را به صورت قراردادی می نویسم

ساخت شیء

```
int main() {
    Student s;
    return 0;
}
```

اسم کلاس اسم شیء;

دسترسی به ویژگی های کلاس

```
int main() {
    Student s1;
    s1.name = "Hossein";
    s1.age = 24;
    s1.gpa = 19.5;
    return 0;
}
```

- با استفاده از نقطه (.) می توانیم به ویژگی های کلاس دسترسی پیدا کنیم.

مقادیر نامعلوم

```
Student s1;
cout << s1.name << endl;
cout << s1.age << endl;
cout << s1.gpa << endl;
```

- چون ویژگی ها را مقدار دهی نکردیم، هر مقداری ممکن است در آنها باشد.

تمرین 1

برنامه‌ای بنویسید که اطلاعات دو دانشجو را شامل موارد زیر دریافت کند:

- نام (رشته)
- سن (عدد صحیح)
- معدل (عدد اعشاری)

سپس، برنامه باید معدل‌های دو دانشجو را مقایسه کند و اطلاعات دانشجویی که معدل بالاتری دارد (نام، سن، و معدل) را چاپ کند.

متد

```
class Student {
public:
    string name;
    int age;
    double gpt;

    void printInfo() {
        cout << name << endl;
        cout << age << endl;
        cout << gpt << endl;
    }
};
```

استفاده از متد

```
int main() {
    Student s;
    s.name = "Hossein";
    s.age = 20;
    s.gpt = 19.5;

    s.printInfo();
    return 0;
}
```

سطوح دسترسی

سطوح دسترسی public و private

```
class Student {
private:

public:

};
```

متد های setter و getter

متد setAge

```
void setAge(int a) {
    if (age <= 0) {
        age = 20;
    }else {
        age = a;
    }
}
```

This کلمه

استفاده از This

```
void setAge(int age) {
    if (age <= 0) {
        this->age = 20;
    }else {
        this->age = age;
    }
}
```

متد setAge

```
void setAge(const int &age) {
    if (age <= 0) {
        this->age = 20;
    }else {
        this->age = age;
    }
}
```

متد getAge

```
int getAge() {
    return age;
}
```

توابع ثابت

```
int getAge() const {
    return age;
}
```

قوائد نامگذاری

سازنده

سازنده روش اول

```
Student(string n, int a, double g) {
    name = n;
    age = a;
    gpt = g;
}
```

استفاده از سازنده

```
int main() {
    Student s("Hossein", 20, 19.5);
    return 0;
}
```

سازنده روش دوم

```
Student(string name, int age, double gpt) {
    this->name = name;
    this->age = age;
    this->gpt = gpt;
}
```

سازنده روش سوم

```
Student(string name, int age, double gpt):
    name(name), age(age), gpt(gpt) {
}
```

سازنده پیشفرض

```
Student() {


}
```

سازنده پیشفرض

```
Student() {
    name = "Student Name";
    age = 20;
    gpt = 15.5;
}
```

سربار گذاری سازنده
(داشتن همزمان چند سازنده)

چرخه حیات یک شیء

اجرا شدن سازنده

```
Student(string name, int age, double gpt) {
    this->name = name;
    this->age = age;
    this->gpt = gpt;
    cout << "Student " << this->name
         << " is born." << endl;
}
```

مخرب ☺

مخرب پیشفرض

```
~Student() {

}
```

مخرب

```
~Student() {
    cout << "Student " << this->name
         << " is dead :)" << endl;
}
```

مرور مباحث

- یادآوری مباحث تابع
- تعریف کلاس
- تعریف ویژگی
- تعریف متد
- سطوح دسترسی
- تعریف متدهای getter و setter
- سازنده و مخرب