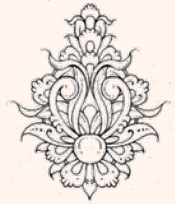
مبانی برنامه‌نویسی
(۱۳۹۹-۱۳-۱۱)
جلسه‌ی هجدهم

تابع ۳

دانشگاه شهید بهشتی
پاییز ۱۳۹۹
دانشکده‌ی مهندسی برق و کامپیوتر
احمد محمودی ازناوه

فهرست مطالب

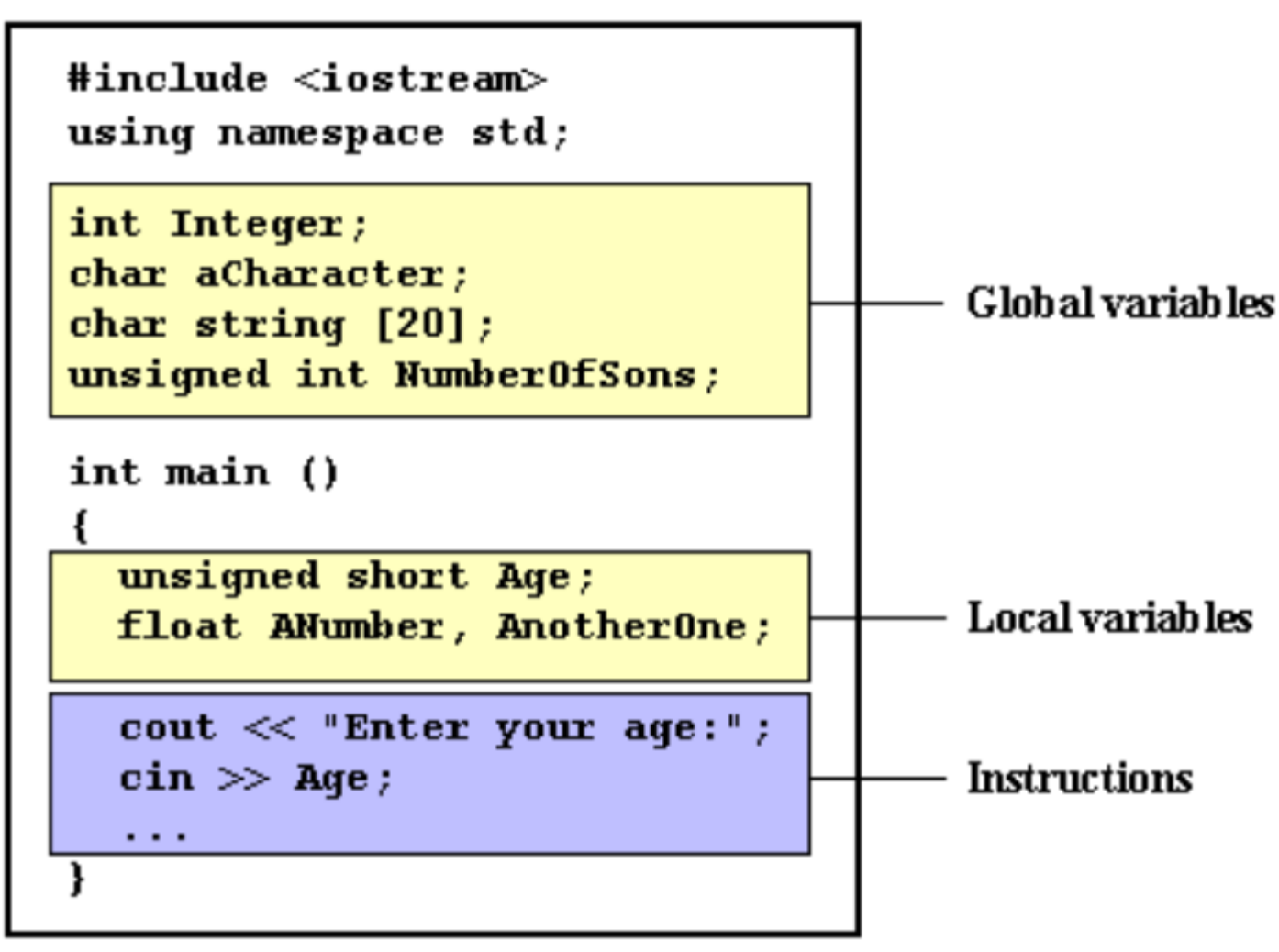
- محدوده‌ی اعتبار متغیرها
- متغیرهای محلی ایستا
- function overloading
- تابع inline
- ارسال از طریق مقدار و ارسال از طریق ارجاع
- ارجاع ثابت



محدوده‌ی اعتبار متغیرها

- متغیرهای محلی(Local) : در درون یک بلوک تعریف می‌شوند و حوزه اعتبار و طول عمرشان از ابتدا تا انتهای بلوکی است که در آن تعریف شده‌اند.
- متغیرهای سراسری (Global): متغیرهایی هستند که در خارج از بلوک و توابع تعریف می‌شوند و حوزه و طول عمر این دست متغیرها از هنگام تعریف تا انتهای برنامه است.
- محل تعریف «متغیر سراسری» قبل از تابع main است.
- به صورت کلی هنگامی از متغیرهای سراسری استفاده می‌شود که توابع نیاز به یک متغیر مشترک داشته باشند.

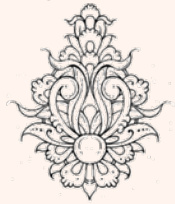
محدوده‌ی اعتبار متغیرها



نکات

- متغیرهای محلی داخل یک بلوک تعریف می‌شوند و در داخل بلوک قابل دسترسی‌اند.

- بدنه‌ی یک تابع یک بلوک محسوب می‌شود و متغیرهای تعریف شده در تابع، متغیرهای محلی آن تابع هستند.

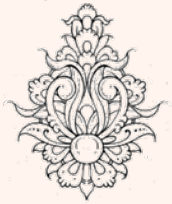


- پارامترهای تابع نیز برای تابع متغیر محلی خواهند بود.

مثال

```
void anotherFunction(); // Function prototype

int main()
 {
 int num = 1; // Local variable
 cout << "In main, num is " << num << endl;
 anotherFunction();
 cout << "Back in main, num is still " << num << endl;
 return 0;
 }


 void anotherFunction()
 {
 int num = 20; // Local variable
 cout << "In anotherFunction, num is " << num << endl;
 }
```

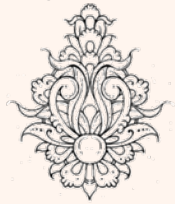
```
In main, num is 1
In anotherFunction, num is 20
Back in main, num is still 1
```



- به صورت پیش‌فرض متغیر سراسری با **صفر** مقداردهی می‌گردد.

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

int globalNum;

int main()
 {
 cout << "globalNum is " << globalNum << endl;
 return 0;
 }
```

```
globalNum is 0
```



متغیر سراسری و محلی هم‌نام

- اگر تابعی متغیر محلی و سراسری هم‌نام داشته باشد، تنها متغیر محلی توسط تابع دیده می‌شود.

```
// Function prototypes
 void A();
 void B();

 int num = 10; // Global variable

 int main()
 {
 cout << "\nIn main, the value of
 A();
 B();
 cout << "\nBack in main, the value of num is: " << num <<endl;
 return 0;
 }
```

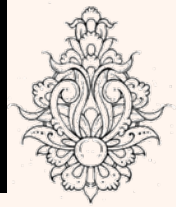
```
 void A()
 { int num = 100; // Local variable
 cout << "\nThere value of num is: " << num;
 }

void B()
 { int num = 200; // Local variable
 cout << "\nThere value of num is: " << num;
 }
```

```
In main, the value of num is: 10
There value of num is: 100
There value of num is: 200
Back in main, the value of num is: 10
```

Scope های تودرتو و موازی

```
void f();
void g();
int x = 11;
int main()
{
    int x = 22;
    {
        int x = 33;
        cout << "In block inside main(): x = " << x << endl;
    }
        cout << "In main(): x = " << x << endl;
        cout << "In main(): ::x = " << ::x << endl;
        f();
        g();
} // end scope of main()
void f()
{
    int x = 44;
    cout << "In f(): x = " << x << endl;
} // end scope of f()
void g()
{
    cout << "In g(): x = " << x << endl;
}
```

```
In block inside main(): x = 33
In main(): x = 22
In main(): ::x = 11
In f(): x = 44
In g(): x = 11
```

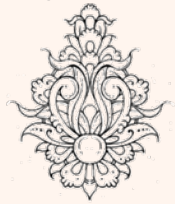


Scope های تودرتو و موازی (ادامه...)

- هر متغیر تنها در محدوده‌ی خویش معتبر است.
- مقدار هر متغیر سراسری با تعریف در محدوده‌ی جدید override می‌گردد.

استفاده از اپراتور :: باعث می‌شود متغیر global نشان داده شود .

Scope resolution operator



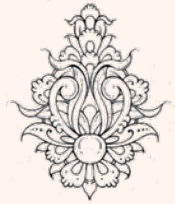
متغیرهای محلی ایستا

Static Local Variables

- اگر یک تابع در جریان فراخوانی بیش از یک‌بار فراخوانی شود، متغیرهای محلی استفاده شده در هر بار فراخوانی ایجاد و از میان می‌روند.

```
// Function prototype
 void showLocal();
 int main()
 {
 showLocal();
 showLocal();
 return 0;
 }
 void showLocal()
 {
 int localNum = 5; // Local variable

 cout << "localNum is " << localNum << endl;
 localNum = 99;
 }
```

```
localNum is 5
localNum is 5
```



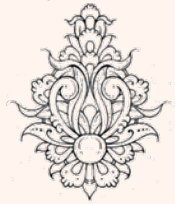
متغیرهای محلی ایستا

- اگر بخواهیم در جریان فراخوانی تابع مقدار متغیر محلی از آخرین فراخوانی به خاطر سپرده شود و متغیر مذکور از میان نرود از متغیر ایستا استفاده می‌کنیم.
- هنگامی که لازم باشد یک متغیر محلی مقدار قبلی خود را حفظ کرده، در فراخوانی بعدی بتوان از آن استفاده کرد، متغیر را ایستا تعریف می کنیم.
- طول عمر متغیر ایستا تا انتهای برنامه است.
- به صورت پیش‌فرض مقدار متغیر ایستا با صفر مقداردهی اولیه می‌شود.
- کاربر می‌تواند متغیر ایستا را با مقداری غیرصفر مقداردهی اولیه نماید.

متغیر ایستا (مثال)


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

// Function prototype
void showStatic();
int main()
 {
 for (int count = 0; count < 5; count++)
 showStatic();
 return 0;
 }

 void showStatic()
 {
 static int statNum; // Static local variable

 cout << "statNum is " << statNum << endl;
 statNum++;
 }
```

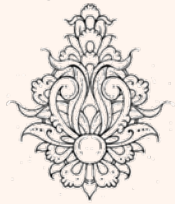
```
statNum is 0
statNum is 1
statNum is 2
statNum is 3
statNum is 4
```



متغیر ایستا (مثال)

```
statNum is 5
statNum is 6
statNum is 7
statNum is 8
statNum is 9
```

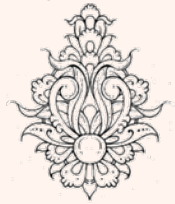
```
// Function prototype
void showStatic();
int main()
 {
 for (int count = 0; count < 5; count++)
 showStatic();
 return 0;
 }

 void showStatic()
 {
 static int statNum=5; // Static local variable

 cout << "statNum is " << statNum << endl;
 statNum++;
 }
```



آرگومان‌های پیش‌فرض

- چنان‌چه برای یک تابع آرگومان‌های پیش‌فرض تعریف شود، در صورت عدم ارسال آرگومان، آرگومان‌های پیش‌فرض مورد استفاده قرار می‌گیرند.

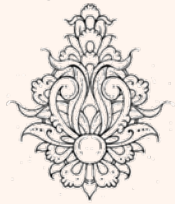
C++



آرگومان‌های پیش‌فرض

```
// default values in functions
#include <iostream>
using namespace std;

int divide (int a, int b=2)
{
  int r;
  r=a/b;
  return (r);
}

int main ()
{
  cout << divide (12);
  cout << endl;
  cout << divide (20,4);
  return 0;
}
```

```
6
5
```



مثال

```
// Function prototype with default arguments
 void displayStars(int = 10, int = 1);

int main()
 {
 displayStars();
 cout << endl;
 displayStars(5);
 cout << endl;
 displayStars(7, 3);
 return 0;
 }
 void displayStars(int cols, int rows)
 {
  for (int down = 0; down < rows; down++)
   {
      for (int across = 0; across < cols; across++)
       cout << '*';
     cout << endl;
   }
 }
```

Uses default values of 10 and 1 for cols and rows

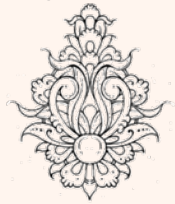
Uses 5 for cols and default value of 1 for rows

Uses 7 for cols and 3 for rows (no default values)

C++

overloading

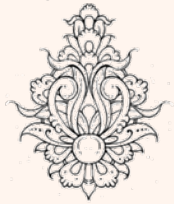
- دو یا تعداد بیشتری تابع هم‌نام که به لحاظ آرگومان‌های ورودی (تعداد و نوع) متفاوت باشند را overload شده می‌نامند.
- در جریان فراخوانی، هرزمان پارامترهای ورودی تابع با آرگومان‌های یکی از توابع همسان باشد، تابع متناظر فراخوانی می‌شود.



مثال

```
int main()
{
int userInt;
double userReal;

cout << "Enter an integer and a floating-point value: ";
cin >> userInt >> userReal;
cout << "Here are their squares: ";
cout << square(userInt) << " and " << square(userReal) << endl;
return 0;
}

int square(int number)
{
return number * number;
}

double square(double number)
{
return number * number;
}
```

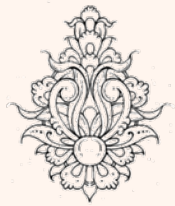
```
Enter an integer and a floating-point value: 7 5.6
Here are their squares: 49 and 31.36
```



مثال

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

int operate (int a, int b)
{
  return (a*b);
}

float operate (float a, float b)
{
  return (a/b);
}

int main ()
{
  int x=5,y=2;
  float n=5.0,m=2.0;
  cout << operate (x,y);
  cout << "\n";
  cout << operate (n,m);
  cout << "\n";
  return 0;
}
```

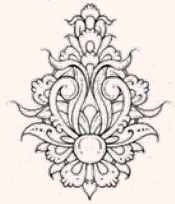
```
10
2.5
```



نکات

- براساس پارامترهای ورودی overloading صورت می‌پذیرد.
- کامپایلر بر اساس نوع آرگومان بازگشتی تفاوتی مابین توابع قایل نیست.

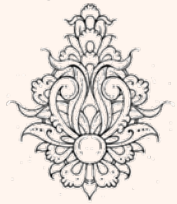
```
int square(int)
double square(int)
```



C++ C99

تابع inline

- در فراخوانی تابع همواره مراحلی باید انجام شود که هم زمان‌بر است و هم به حافظه‌ی بیشتری جهت فراخوانی نیاز دارد.
- مراحلی چون ارسال آرگومان‌های ورودی، در نظر گرفتن فضا برای متغیرهای محلی و... وجود دارد.
- در برخی موارد بهتر است برای به دست آوردن بازده بیشتر از انجام چنین مراحلی جلوگیری شود.
- در این گونه موارد با قرار دادن کلمه‌ی کلیدی inline، کامپایلر بدنه‌ی تابع را به جای فراخوانی، در برنامه‌ی اصلی قرار می‌دهد.



تابع inline

```
inline int cube(int x)
{ // returns cube of x:
  return x*x*x;
}
```

```
int main()
{ // tests the cube() function:
  cout << cube(4) << endl;
  int x, y;
  cin >> x;
  y = cube(2*x-3);
}
```

```
int main()
{ // tests the cube() function:
  cout << (4)*(4)*(4) << endl;
  int x, y;
  cin >> x;
  y = (2*X-3)*(2*X-3)*(2*X-3);
}
```

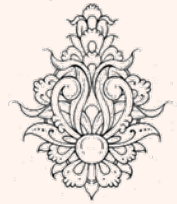
توسط کامپایلر این گونه در نظر گرفته می‌شود

نکات

- چه تابعی را بهتر است inline در نظرگرفت؟

تابعی که کوچک و محاسباتی در حد چند خط باشد را بهتر است inline در نظر بگیرید، زیرا overhead فراخوانی تابع برای این دست توابع کوچک که میزان فراخوانی زیادی نیز در برنامه دارند بالاست.

لزوماً استفاده از inline برای هر تابعی بازده را افزایش نخواهد داد، می‌تواند سبب کاهش بازده نیز گردد.

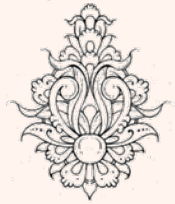


call by value

ارسال از طریق مقدار

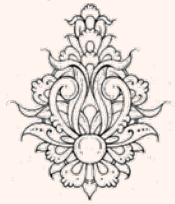
- به صورت معمول، هنگامی‌که تابعی فراخوانی می‌شود، آرگومان‌های ورودی به صورت ارسال از طریق مقدار در نظر گرفته می‌شوند.
- در این حالت یک کپی از مقادیر گرفته شده، فرآیندها به روی کپی مورد نظر اعمال می‌گردند.
- در بسیاری موارد هنگامی‌که مقادیر ارسالی دارای حجم بالا باشند، کپی به حافظه‌ی زیادی نیاز دارد و مقرون به صرفه نیست.

راه حل ارسال از طریق ارجاع



ارسال با ارجاع

- متغیر ارجاعی نام دیگری برای متغیر اصلی است.
- به همین علت هر تغییری که به روی متغیر ارجاعی اعمال گردد، متغیر اصلی نیز تغییر خواهد کرد.
- در مواردی که نیاز باشد تغییرات به روی متغیر اصلی نیز صورت پذیرد آرگومان‌های تابع را به صورت «ارجاع» در نظر می‌گیریم.
  - مثال: تابعی که دو عدد را جابه‌جا می‌کند.



ارسال با ارجاع (ادامه...)

Ampersand

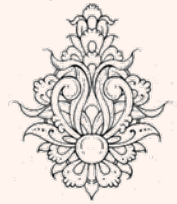
- متغیر ارجاعی همانند متغیر معمولی تعریف می‌شود و تنها یک «&» برای نشان دادن نوع ارجاع در نظر گرفته می‌شود.

```
void doubleNum(int &refVar)
{
    refVar *= 2;
}
```

متغیر **refVar** یک رفرنس به **int** خواهد بود

prototype

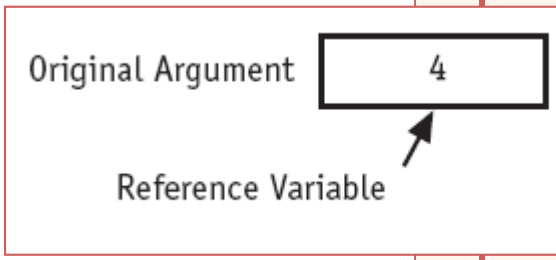
```
void doubleNum(int &);

void doubleNum(int&);
```



ارسال با ارجاع (مثال)

```
// Function prototype. The parameter is a reference variable.
 void doubleNum(int&);
 int main()
 {
 int value = 4;

 cout << "In main, value is " << value << endl;
 cout << "Now calling doubleNum..." << endl;
 doubleNum(value);

 cout << "Now back in main, value is " << value << endl;
 return 0;
 }
```

هنگامی که با متغیر ارجاعی کار می کنیم فرآیند به روی متغیری که با آن ارجاع می‌شود صورت می‌گیرد




```
void doubleNum (int& refVar)
{
refVar *= 2;
}
```

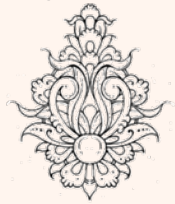
```
In main, value is 4
Now calling doubleNum...
Now back in main, value is 8
```

ارسال با ارجاع (مثال)

```
void doubleNum(int&);
void getNum(int&);
 int main()
 {
 int value=0;
 getNum(value);
 doubleNum(value);
 cout << "That value doubled is " << value << endl;
 return 0;
 }
```

```
Enter a number: 45
That value doubled is 90
```

```
void getNum(int& userNum)
{
cout << "Enter a number: ";
cin >> userNum;
}
void doubleNum (int& refVar)
{
refVar *= 2;
}
```



ارسال با ارجاع (نکات)

- تنها «متغیرها» از طریق ارجاع می‌توانند ارسال گردند.
- ارسال عبارات و ثابت‌ها از طریق ارجاع برنامه را با خطا مواجه می‌کند.

```
doubleNum(5);                 // Error
doubleNum(value + 10);        // Error
```

- به دلیل تغییر متغیر خارج از تابع هنگام ارسال از طریق ارجاع، این پتانسیل به برنامه داده می‌شود که به صورت سهوی مقادیر تغییر یابد.

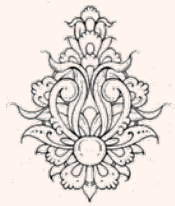
تنها در مواردی که نیاز است از ارسال به طریق ارجاع استفاده کنید

ارسال با ارجاع (نکات)

- چه زمان ارسال از طریق ارجاع داشته باشیم چه زمان از طریق مقدار؟
  - هنگامی که آرگومان ورودی ثابت است، ارسال از طریق مقدار صورت می‌گیرد (تنها متغیرها از طریق ارجاع قابلیت ارسال دارند)

ارسال از طریق مقدار

  - هنگامی‌که مقدار یک متغیر نمی‌باید در جریان فراخوانی تابع تغییر کند ارسال از طریق مقدار است.
  - هنگامی‌که خروجی تنها یک متغیر است از طریق return بازگشت داده می‌شود و ارسال از طریق مقدار است.



ارسال با ارجاع (نکات – ادامه...)

- چه زمان ارسال از طریق ارجاع داشته باشیم چه زمان از طریق مقدار؟
  - هنگامی‌که بیش از یک مقدار بناست تغییر داده شود (چون تنها یک مقدار بازگشتی می‌توان داشت)، ارسال از طریق ارجاع صورت می‌گیرد.
  - هنگامی‌که داده‌ی اصلی مقدار بزرگی است، کپی آن فرآیندی منطقی نیست، ارسال به شیوه‌ی ارجاع صورت می‌گیرد.
  - هنگامی‌که ماهیت تابع تغییر آرگومان‌های ورودی باشد.

تابع Swap