مبانی برنامه نویسی
C

دانشکده برق و رباتیک

دانشگاه صنعتی شاهرود

حسین خسروی

HosseinKhosravi@gmail.com

سرفصل

- مقدمه ای بر زبان C
- نحوه نوشتن برنامه به زبان C
- اولین برنامه به زبان C

C چیست؟

- یک زبان برنامه نویسی سطح پایین برای رایانه
- اولین بار توسط Dennis Ritchie در آزمایشگاههای AT&T Bell در سال ۱۹۷۲ معرفی شد
- امروزه بسیار مورد استفاده است:
  - برنامه‌نویسی سیستم عاملها مثل Linux
  - برنامه‌نویسی میکروکنترلرها و پردازنده‌های DSP
  - و برنامه‌های کاربردی و سیستمی در تمام سیستم عاملها

ویژگیهای C

- کلمات کلیدی محدود «keywords»
- قابلیت تعریف داده‌های مرکب مثل ساختار و یونیون
- اشاره گرها
- کتابخانه‌های استاندارد
- تولید کد ماشین
- قابلیت تعریف پیش پردازنده‌ها

نسخه‌های C

- در سال ۱۹۷۲ زبان C ابداع شد
- در سال ۱۹۷۸ مشخصات اولین نسخه عمومی آن منتشر شد
- در سال ۱۹۸۹ استاندارد C89 توسعه داده شد
  - با عنوان ANSI C یا Standard C نیز شناخته می‌شود
- در سال ۱۹۹۰ توسط سازمان جهانی استاندارد ISO به عنوان یک استاندارد پذیرفته شد. C90
- در سال ۱۹۹۹ استاندارد C99 توسعه داده شد
  - سازگار با نسخه های قبل
- آخرین استاندارد C11
- مباحث این درس با استاندارد C99 و بعضا C11 خواهد بود.

C در مقابل زبانهای خویشاوند!

- مشتقات اخیر C عبارتند از: C++, Objective C , C#
- سایر زبانهایی که از C تاثیر پذیرفته اند: جاوا، پرل و پایتون
- زبان C فاقد موارد زیر است:
  - مدیریت استثنائات
  - کنترل محدوده‌ی حافظه‌ها
  - مدیریت و پاکسازی حافظه
  - شیء گرایی و چند ریختی
  - ...
- در مقابل، کد تولید شده به زبان C سرعت بیشتری نسبت به سایر زبانها دارد

سرفصل

- مقدمه ای بر زبان C
- **نحوه نوشتن برنامه به زبان C**
- اولین برنامه به زبان C

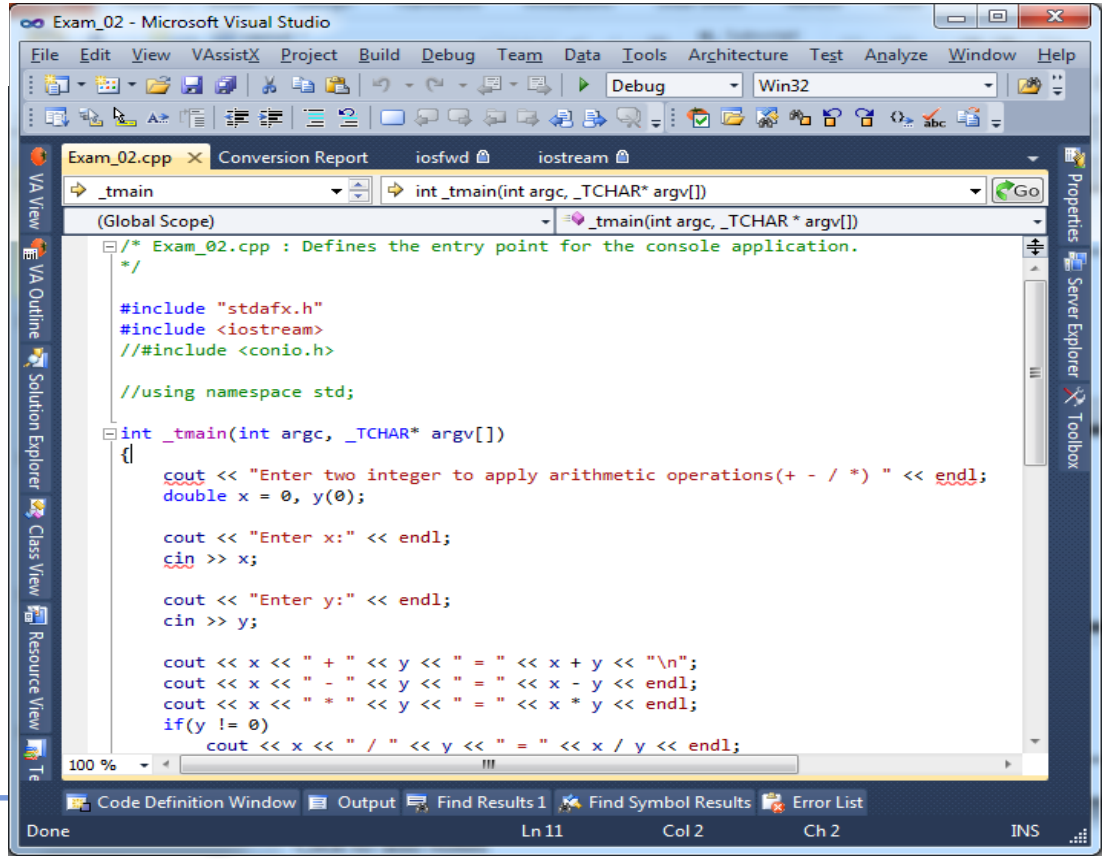
از نوشتن کد تا اجرا

- **هر زبان برنامه نویسی به سه ابزار نیازمند است:**
  - ویرایشگر: محیطی برای توسعه
  - کامپایلر: جهت بررسی صحت کد از نظر قواعد زبان و تولید کدهای ماشین برای هر فایل
  - لینکر: جهت ادغام کدهای ماشین فایلهای مختلف و تولید فایل اجرایی (exe)
- **محیط توسعه یکپارچه (IDE)**
  - هر سه مورد فوق (و بیش از آن) را یکجا فراهم می‌کند
  - مانند ویژوال استودیو، Eclipse، Dev-Cpp و Qt Creator

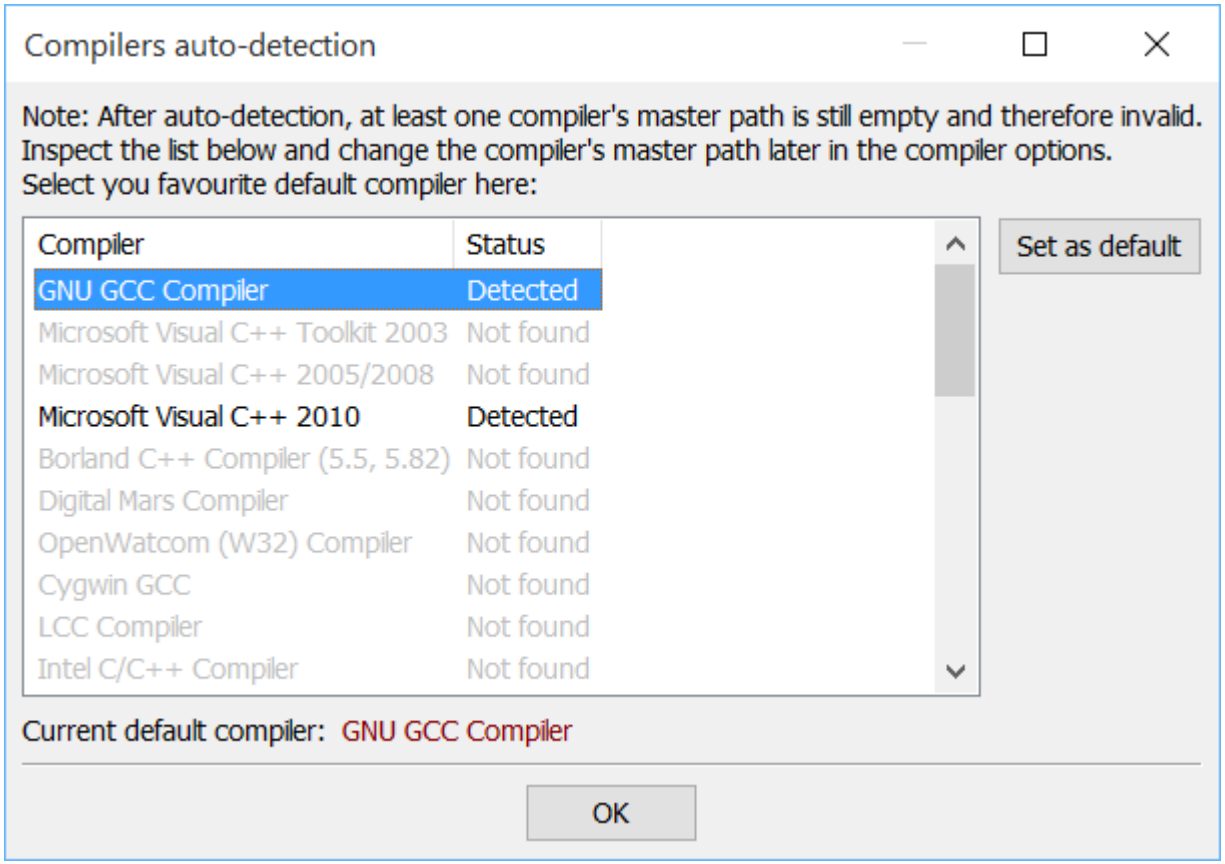
محیطهای برنامه نویسی C

- محیطهای مورد استفاده ما در این درس Code::Blocks و ویژوال استودیو ۲۰۱۰ خواهد بود

Microsoft Visual Studio 2010

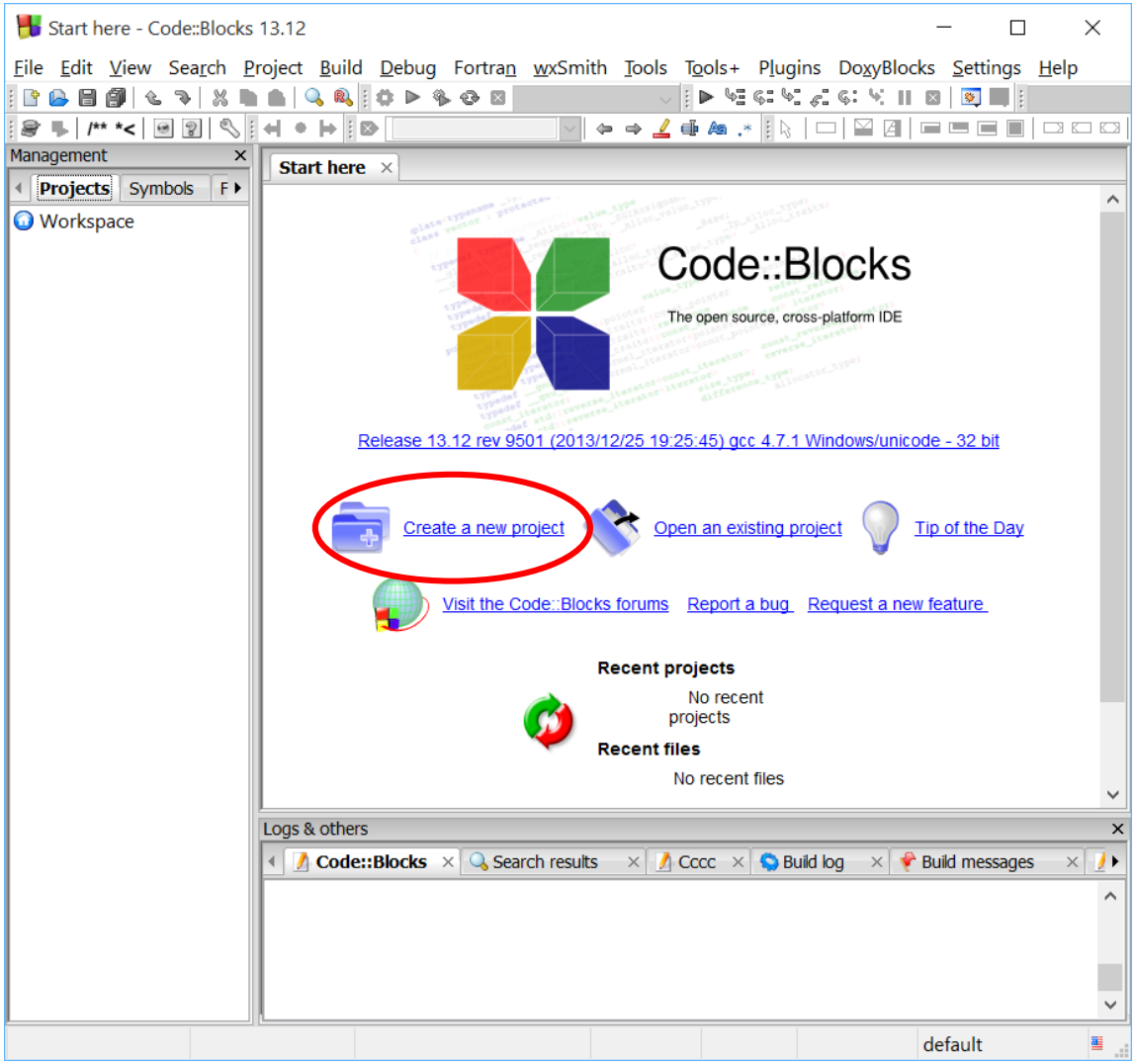


- پس از نصب، اولین گام انتخاب کامپایلر است

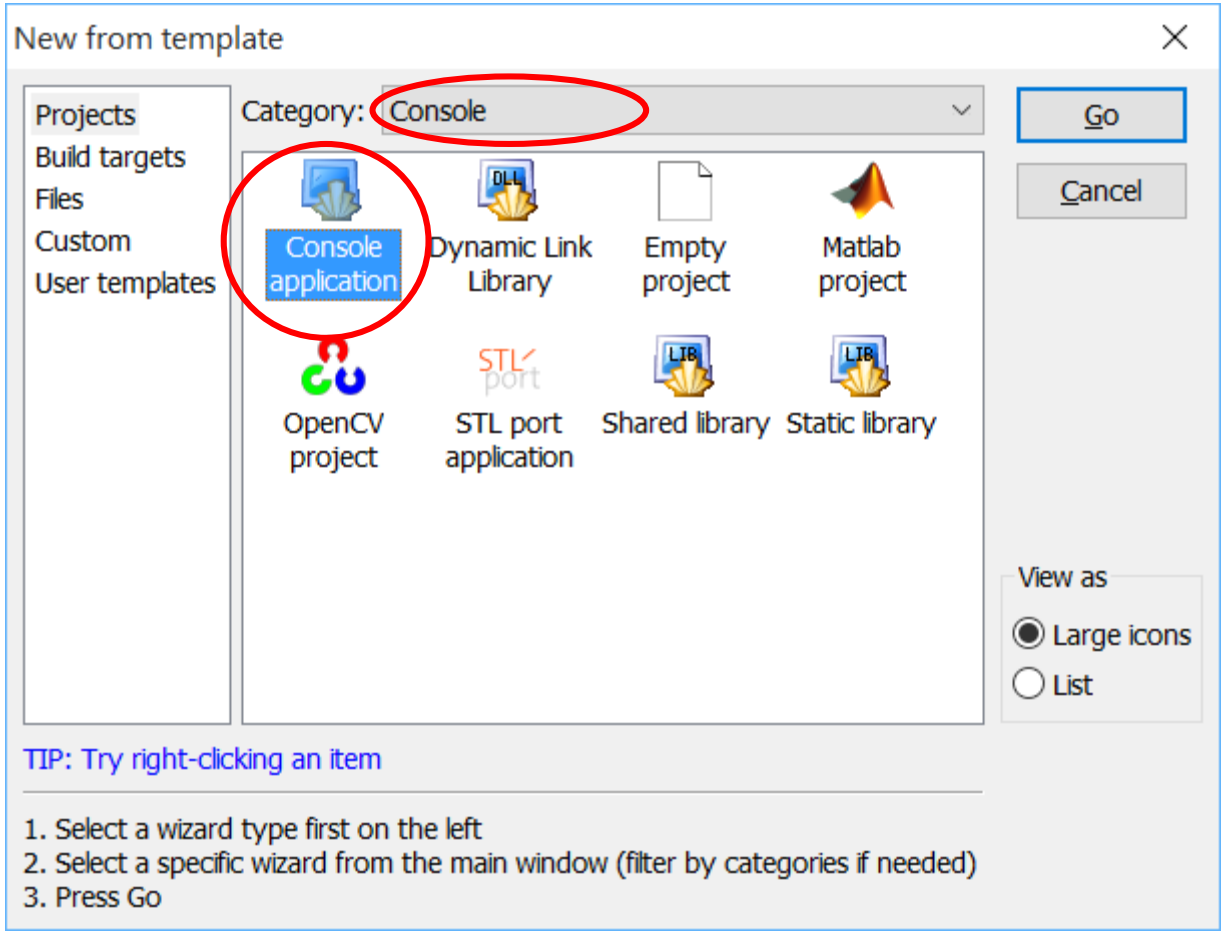


شروع کار با کدبلاکس

Create a new project

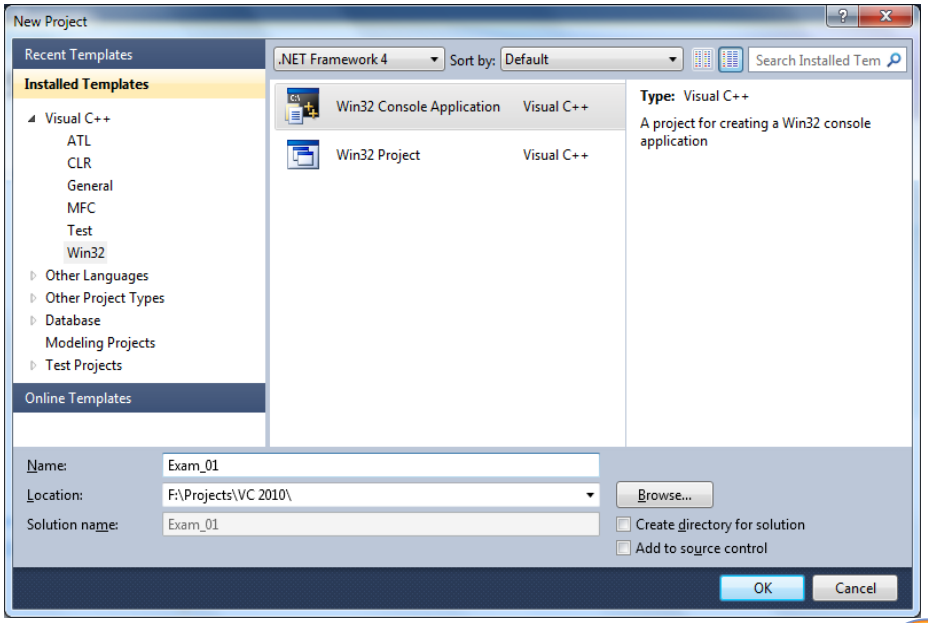


Console Application



نحوه استفاده از ویژوال سی

- برای برنامه نویسی در محیط VC باید یک پروژه کنسول ایجاد کنیم:
- File -> New Project ->Visual C++ -> Win32 -> Win32 Console Application
- با انتخاب این گزینه، یک نام مناسب برای پروژه انتخاب می کنیم، مثلا Calculator یا HW_01
- پس از تایید، فایلی با پسوند cpp جهت ورود کد تولید می‌شود.



نحوه کامپایل و اجرای برنامه

- کامپایل برنامه و دیدن خطاهای احتمالی
  - Build -> Compile (Ctrl+F7)
  - Build -> Build Solution (F7)
  - Build -> Build {Project-Name}
- کامپایل، اجرا و دیباگ کردن برنامه
  - Debug -> Start Debugging (F5)
- توضیح بیشتر بعدا

سرفصل

- مقدمه ای بر زبان C
- نحوه نوشتن برنامه به زبان C
- **اولین برنامه به زبان C**

ساختار یک فایل C

```
/* Begin with comments about file contents */
```

- Insert `#include` statements and preprocessor definitions Function prototypes and variable declarations
- Define `main()` function

```
{
    Function body
}
```

- Define other functions

```
{
    Function body
}
...
```

توضیحات

- توضیح تک خطی با استفاده از //

```
//  this  is  a simple comment
```

- توضیح چند خطی با استفاده از /* */

```
/* Hello C Program
Created by H. Khosravi
1391/06/01 */
```

- توضیحات، توسط کامپایلر نادیده گرفته می‌شود.
- هر جایی از برنامه می‌تواند ظاهر شود.

ماکروی #include

فایلهای سرآیه (header)

- الگوی توابع، ثوابت و سایر اعلانات در فایلهای سرآیه هستند
- وقتی می‌خواهیم از اطلاعات یک فایل یا کتابخانه در برنامه‌ی خودمان استفاده کنیم، فایل سرآیه‌ی آن کتابخانه را در برنامه خودمان به صورت `#include <targetfile.h>` قرار می‌دهیم

وجه تسمیه

- از آنجا که معمولا فایلهای سرآیه را در ابتدای برنامه، یا سر برنامه، قرار می‌دهیم به آنها فایل سرآیه گوییم.

فایلهای سرآیه

- مثال: برای استفاده از توابع استاندارد ورودی و خروجی باید از کتابخانه stdio.h استفاده کنیم؛ این کتابخانه بخشی از کتابخانه‌ی استاندارد C است: `#include <stdio.h>`
- سایر فایلهای سرآیه‌ی مهم: ctype.h, math.h, stdlib.h, string.h, time.h
- فایلهای سرآیه باید در مسیرهای include پروژه قرار داشته باشند
- تعیین مسیرهای اضافه بر مسیرهای پیش فرض با استفاده از:
  - Property sheet
- برای فایلهای سرآیه‌ی موجود در غیر مسیر پیش‌فرض، از ” “ به جای <> استفاده می‌شود: `#include “myLib.h”`

Declaring variables

- Must declare variables before use
- Variable declaration:

```
int  n;
float  phi;
```

- int - integer data type
- float - floating-point data type
- Many other types (more next lecture. . .)

Initializing variables

- Uninitialized, variable assumes a default value
- Variables initialized via assignment operator:

  n = 3;
- Can also initialize at declaration:

  float phi = 1.6180339887;
- Can declare/initialize multiple variables at once:

  int a, b, c = 0, d = 4;

Arithmetic expressions

Suppose x and y are variables

- x+y, x-y, x*y, x/y, x%y: binary arithmetic
- A simple statement:

  ```
  y = x+3*x/(y-4);
  ```

- Numeric literals like 3 or 4 valid in expressions
- Semicolon ends statement (not newline)
- `x += y; x -= y; x *= y; x /= y; x %= y;` arithmetic and assignment

Order of operations

- Order of operations:

Operator	Evaluation direction
+,- (sign)	Left-to-right
*,/,%	left-to-right
+,-	left-to-right
=,+=,-=,*=,/=,%=	right-to-left

- Use parentheses to override order of evaluation

Order of operations

Assume x = 2.0 and y = 6.0. Evaluate the statement

```
float z = x+3*x/(y-4);
```

1. Evaluate expression in parentheses
   `float z = x+3*x/(y-4);` → `float z = x+3*x/2.0;`

Order of operations

Assume x = 2.0 and y = 6.0. Evaluate the statement

float z = x+3∗x/(y−4);

1. Evaluate expression in parentheses

   float z = x+3∗x/(y−4); → **float** z = x+3∗x/2.0;

2. Evaluate multiplies and divides, from left-to-right

   float z = x+3∗x/2.0; → **float** z = x+6.0/2.0; → **float** z = x+3.0;

Order of operations

Assume x = 2.0 and y = 6.0. Evaluate the statement

float z = x+3∗x/(y−4);

1. Evaluate expression in parentheses

   float z = x+3∗x/(y−4); → **float** z = x+3∗x/2.0;

2. Evaluate multiplies and divides, from left-to-right

   float z = x+3∗x/2.0; → **float** z = x+6.0/2.0; → **float** z = x+3.0;

3. Evaluate addition

   float z = x+3.0; → **float** z = 5.0;

Order of operations

Assume x = 2.0 and y = 6.0. Evaluate the statement

float z = x+3∗x/(y−4);

1. Evaluate expression in parentheses

   float z = x+3∗x/(y−4); → **float** z = x+3∗x/2.0;

2. Evaluate multiplies and divides, from left-to-right

   float z = x+3∗x/2.0; → **float** z = x+6.0/2.0; → **float** z = x+3.0;

3. Evaluate addition

   float z = x+3.0; → **float** z = 5.0;

4. Perform initialization with assignment

   Now, z = 5.0.

Order of operations

Assume x = 2.0 and y = 6.0. Evaluate the statement

float z = x+3∗x/(y−4);

1. Evaluate expression in parentheses

   float z = x+3∗x/(y−4); → **float** z = x+3∗x/2.0;

2. Evaluate multiplies and divides, from left-to-right

   float z = x+3∗x/2.0; → **float** z = x+6.0/2.0; → **float** z = x+3.0;

3. Evaluate addition

   float z = x+3.0; → **float** z = 5.0;

4. Perform initialization with assignment

   Now, z = 5.0.

How do I insert parentheses to get z = 4.0?

Order of operations

Assume x = 2.0 and y = 6.0. Evaluate the statement

float z = x+3∗x/(y−4);

1. Evaluate expression in parentheses

   float z = x+3∗x/(y−4); → **float** z = x+3∗x/2.0;

2. Evaluate multiplies and divides, from left-to-right

   float z = x+3∗x/2.0; → **float** z = x+6.0/2.0; → **float** z = x+3.0;

3. Evaluate addition

   float z = x+3.0; → **float** z = 5.0;

4. Perform initialization with assignment

   Now, z = 5.0.

How do I insert parentheses to get z = 4.0?

float z = (x+3∗x)/(y−4);

Function prototypes

- Functions also must be declared before use
- Declaration called function prototype
- Function prototypes:

  `int factorial ( int );` or `int factorial ( int n);`
- Prototypes for many common functions in header files for C Standard Library

Function prototypes

- General form:
  return_type function_name(arg1,arg2,...);
- Arguments: local variables, values passed from caller
- Return value: single value returned to caller when function exits
- void – signifies no return value/arguments

```
int  rand(void);
```

The main() function

- main(): entry point for C program
- Simplest version: no inputs, outputs 0 when successful, and nonzero to signal some error

```
int  main(void);
```

- Two-argument form of main(): access command-line arguments

```
int  main(int argc,  char **argv);
```

- More on the char **argv notation later. . .

اعلان تابع

```
function declaration
{
        declare variables;
        program statements;
}
```

- اعلان تابع باید با امضای تابع (در صورت وجود) یکسان باشد
  - نام متغیرها لازم نیست یکسان باشد
  - انتهای اعلان تابع، نقطه ویرگول ندارد
- **آکلادها، بلوک های کد را مشخص می کنند**
  - متغیرهایی که در یک بلوک تعریف می شوند، تنها در همان بلوک دیده می شوند.

تابع main

```
//The main() function
int _tmain(int argc, _TCHAR* argv[])
{
   /* Write message to console */
   puts("Hello, C students");

   return 0; //Exit with success code (0)
}
```

- **تابع puts پارامتر متن خودش را روی کنسول (stdout) نمایش می دهد**
  - این تابع و توابع بسیار دیگری مثل printf، fopen و scanf در کتابخانه <stdio.h> قرار دارند.

Alternative main() function

- Alternatively, store the string in a variable first:

```
int main(void) /* entry point */
{
   const char msg[] = "hello, C students";

   /* write message to console */
   puts(msg);
```

- const keyword: qualifies variable as constant
- char: data type representing a single character; written in quotes: 'a', '3', 'n'
- const char msg[]: a constant array of characters

More about strings

- Strings stored as character array
- Null-terminated (last character in array is '\0' null)
  - Not written explicitly in string literals
- Special characters specified using \ (escape character):
  - \\ ➔ backslash \' ➔ apostrophe
  - \" ➔ quotation mark \b ➔ backspace
  - \t ➔ tab \r ➔ carriage return
  - \n ➔ linefeed
  - *0nn* ➔ octal ASCII \x*nn* ➔ hexadecimal ASCII
  - *e.g.* \x41 – 'A', \060 – '0'

Console I/O

- stdout, stdin: console output and input streams
- puts(string): print string to stdout
- putchar(char): print character to stdout
- char = getchar(): return character from stdin
- string = gets(string): read line from stdin into string
- Many others - later ...

Preprocessor macros

- Preprocessor macros begin with # character

```
#include <stdio.h>
```

- `#define msg "hello, C students"`
  - defines msg as “hello, C students” throughout source file

- Many constants specified this way

Defining expression macros

- **#define** can take arguments and be treated like a function
  - **#define** add3(x,y,z) ((x)+(y)+(z))

- Parentheses ensure order of operations

- compiler performs inline replacement; not suitable for recursion

Conditional preprocessor macros

- **#if**, **#ifdef**, **#ifndef**, **#else**, **#elif** , **#endif** conditional preprocessor macros, can control which lines are compiled
- Evaluated before code itself is compiled, so conditions must be preprocessor defines or literals
- Example:

```
#ifdef TRIAL_VERSION
puts("This is a demo version, please buy full version");
#endif
```

- Other useful directives:
  - **#pragma, #error, #warning, #undef**

Conditional preprocessor macros

- **#pragma**
- preprocessor directive
- **#error**, **#warning** trigger a custom compiler error/warning
- **#undef** msg remove the definition of msg at compile time

تمرین سری اول

- برنامه ماشین حساب ساده (۴ عمل اصلی)
- برنامه تبدیل رشته به حروف بزرگ و کوچک
- برنامه معکوس کردن یک عدد ۳ رقمی
- برنامه محاسبه ب.م.م. و ک.م.م.

Run Sample Program

- How to compile
- How to find compile errors
- How to Run
- How to Debug
  - Watch window
  - Call Stack

Summary

- Topics covered:
  - How to edit, compile, and debug C programs
  - C programming fundamentals: comments
  - preprocessor macros, including #include
  - the main() function
  - declaring and initializing variables, scope
  - using puts() – calling a function and passing an argument
  - returning from a function