Introduction to C programming

Xiaoqiang Wang

Start with the first program

- First, we need to write a file, with ‘source code’, and save it as a ‘*.c’ file.

```
/* you generally want to
 * include stdio.h and
 * stdlib.h
 * */
#include <stdio.h>
#include <stdlib.h>

int main (void)
{
   printf(“Hello World\n”);
   exit(0);
}
```

comments

Include files

Main function,
the start point

Print function

End of main function

From source code to executable file

- The computer needs the help of the so called ‘compiler’ to compile the source code, and link the complied code together to generate an executable file.
- In Linux, we use command line
  g++ try.c –o try
- In windows, we use visual c++. We embed the code into a project, and build the project.

Run the code

- In Linux, we type the command:
  ./try
- In windows, we can double click the executable file to run it. Or use menu/toolbar in the Visual c++.

Variables

- Each variable has a ‘type’:
  int, float, char, double, bool, …
  ```
  int x, y;
  x = 1;
  y = x*2;
  ```
- Each variable has a life scope:
  ```
  int x;
  x = 14;
  {
      int x;
      …
  }
  int y = x*2;
  ```

operations

- Operations are for same type variables
- int type: +, -, *, /, %, ++, --

  Note that 5/2 = 2, not 2.5
- Float or double type: +, -, *, /, ++, --
- Bool type:

  && ---- and

  || ---- or

  ! ---- not
- Comparison:

  ==, >, <, >=, <=, …

  Note:

  ```
  if (x = 1) {
          ……
  }
  ```
- Type conversion: automatically or forced

  ```
  double a = 1;
  a = (double) 1;
  ```

Loops

- For loop:
  for (variable initialization; condition; variable update)
  { Code to execute while the condition is true }
```
      for( i = 0; I < 10; I ++) {
                ……
      }
```
- Do-while:
```
      do { … } while ( condition );
      while(condition) { … };
```
- Infinite loop:
```
      for(;;) {…}
      while(true) { … }
```

Control

-
  ```
  If (bool) {
  }
  else if (bool) {
  }
  else {
  }
  ```
- Break from loop: break, jump out of the loop.
- Continue inside the loop: continue, jump to the loop control statement.
- Customize your loop using break and continue.
  ```
  for(;;) {
          if (I < 10) continue;
          else if (I < 20) {…}
          else break;
  }
  ```

Output

- ‘printf’ function, write output to the screen.
- The printf function is just a useful function from the standard library of functions that are accessible by C programs.
- int printf(const char *format, ...);

```
char name[20] = "Bob";
int age = 21;
printf( "Hello %s, you are %d years old\n", name, age );

double a = 20.5;
int i = 1;
printf(“i = %d; a = %f\n”, i, a);
```

Static Array

- You need to give it a size for a static array when you declare it.

```
int a[20];
double x[1000];
float y[100][100];
```

- You can not do:

```
int n = 10;
int a[n];
```

- It is dangerous to access the array out of bound.

```
int a[10];
int n = 20;
a[n] = 15;
```

- Array index starts from 0.

Head file

- System provided some head files with the definition of some functions. To use those functions, include those head files.

  ```
  #include <stdlib.h>
  #include <math.h>
  ```

- You can create your own head file. You can include your own head file by:

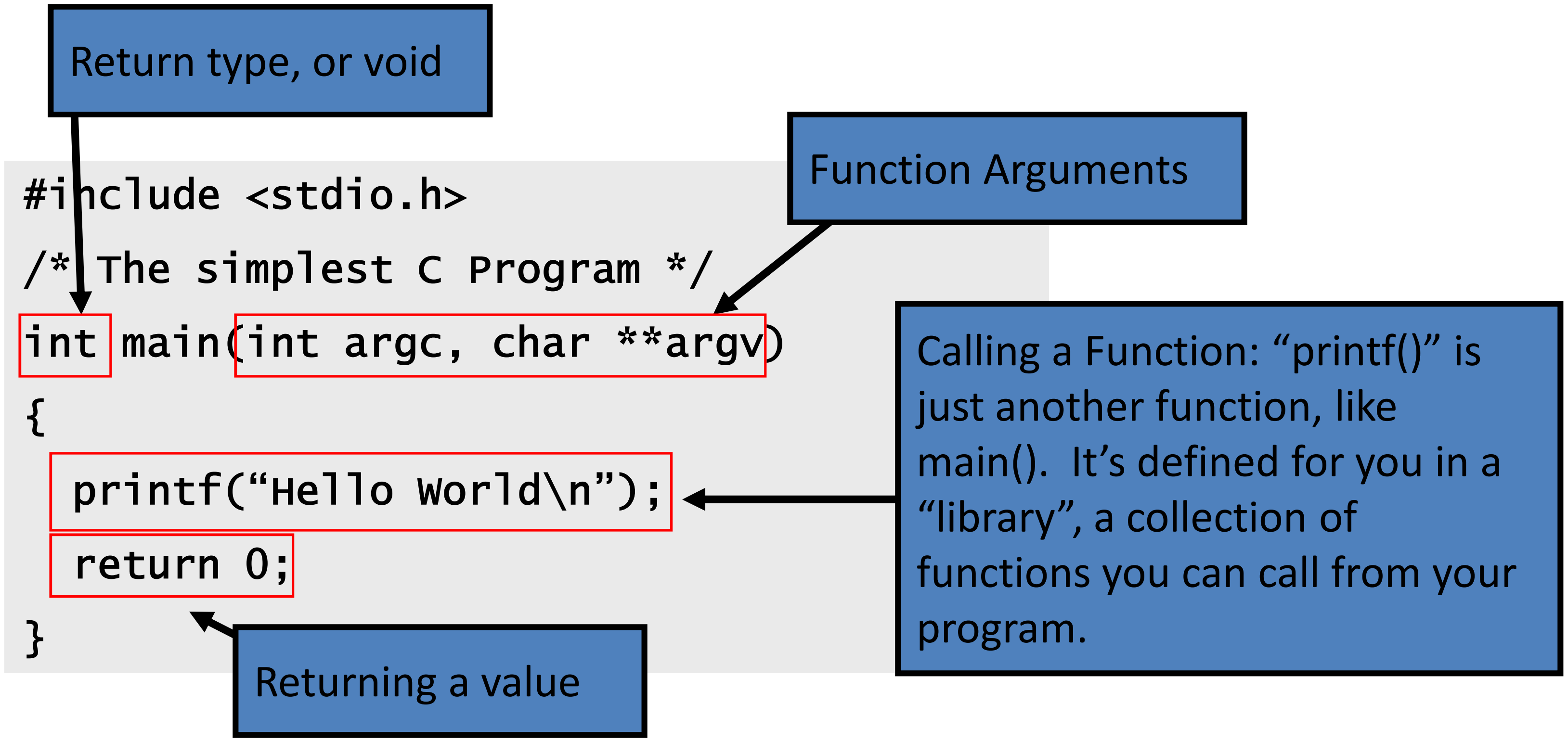
  ```
  #include “tool.h”
  ```

- math.h defines:

  sin, cos, fabs, abs, pow, sqrt, tan, exp, …

function

- A Function is a series of instructions to run. You pass Arguments to a function and it returns a Value.
- "main()" is a Function. It's only special because it always gets called first when you run your program.



```
#include <stdio.h>
/* The simplest C Program */
int main(int argc, char **argv)
{
  printf("Hello World\n");
  return 0;
}
```

Return type, or void
Function Arguments
Calling a Function: "printf()" is just another function, like main(). It's defined for you in a "library", a collection of functions you can call from your program.
Returning a value

Ready for coding – prime numbers

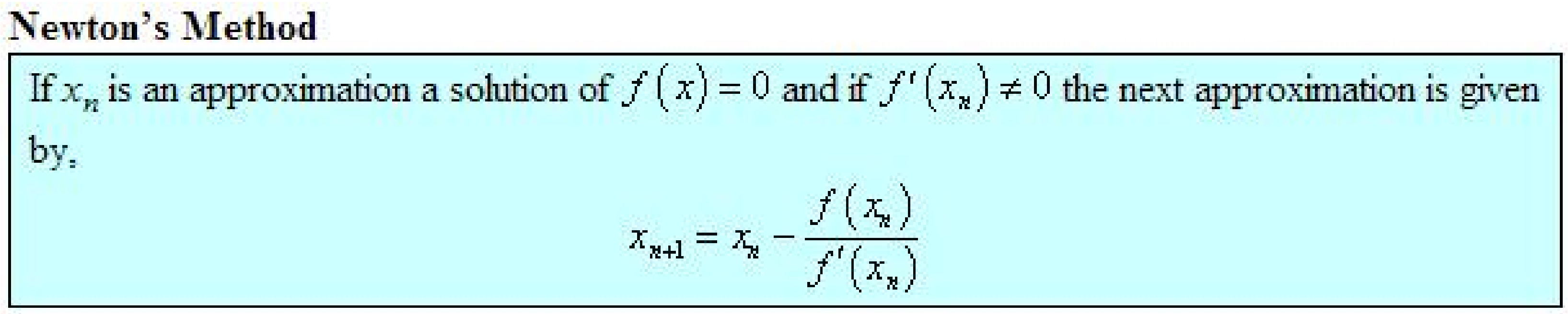
```c
//Header files
#include <stdio.h>
#include <stdlib.h>

int main()
{
   //Program variables
   int cn,cn1,temp,num=200,p_flag=0;

    for(cn=2;cn<num;cn++) //Looping statement
   {
      p_flag = 1;
      for(cn1=2;cn1<cn;cn1++)
      {
         if(cn%cn1==0) //Conditional statement
         {
            p_flag=0;
            break;
         }
      }
      if(p_flag==1)
      printf("%d\n",cn);
   }
}
```

Coding – Newton's method

- Find the solution of $f(x)=0$ by Newton's method.



Newton's Method

If x_n is an approximation a solution of $f(x)=0$ and if $f'(x_n)\neq 0$ the next approximation is given by,

$$x_{n+1}=x_n-\frac{f(x_n)}{f'(x_n)}$$

- First, we need to define f and f'.

```
double f(double x) { return x*x -1; }
double f_d(double x) {return 2*x; }
```

Coding – Newton’s method

- Then we can code for Newton’s method as a function:

```
double Newton(double x0) {
    for(;;) {
        double x1 = x0 – f(x0)/f_d(x0);
        if (fabs(x1 – x0) < 0.00001) break;
        x0 = x1;
    }
    return x0;
}
```

Coding – Newton's method

- Finally, call Newton function in the main function:

```
int main()
{
	double sol = Newton(0.0);
	printf("The soluction of f(x) = 0 is x = %f\n", sol);
	return 0;
}
```

- Compile and run the code.

Coding – Matrix multiplication

```
int main()
{
        double A[10][10];
        double B[10][10];
        double x[10];
        //initialize A, B, and x
        int i, j, k;
        for (i = 0; i < 10; i++) {
                for (j = 0; j < 10; j ++) {
                        A[i][j] = 1.0/(i + j + 1);
                        B[i][j] = 1.0/(i*j+1);
                }
                x[i] = i;
        }
        //matrix multiplication
        double C[10][10];
        for (i = 0; i < 10; i++) {
                for (j = 0; j < 10; j ++) {
                        c[i][j] = 0.0;
                        for {k = 0; k < 10; k++) {
                                c[i][j] += A[i][k]*B[k][j];
                        }
                }
        }

}
```

Home work

- Write a program to do numerical integration. For example,

$$\int_0^\pi \sin x \, dx = 2$$