First Session. C++ Study Group

Luis Barboza

PQFC

October 10th, 2012
1. Introduction: C Language
2. Basic Syntax
3. Control Flow
4. Functions
5. Pointers and Arrays
6. Structures
7. Compilation Process
Outline

1. Introduction: C Language
2. Basic Syntax
3. Control Flow
4. Functions
5. Pointers and Arrays
6. Structures
7. Compilation Process
Some History

- Developed by Dennis Ritchie between 1969 and 1973. (Bell Labs)
- It was designed to be used with UNIX operating system, but it was extended to other systems.
- Procedures is the basic structure of C language.
Advantages

- One of the most flexible languages.
- Most of potential employers demand Quants with C++ experience.
- Communication with statistical packages (R and Matlab).
How to access C/C++?

In these lectures we will use Linux operating system. You can do the same procedures with Microsoft Visual Studio (Windows)

In order to use Linux OS within Itap’s computers:

- We will use a SSH client:
  Standard Software > Telecommunications > SecureCRT

- We will use a SFTP client:
  Standard Software > Telecommunications > SecureFX
Servers

- Stat students: skew.stat.purdue.edu
- Or if you want direct access to Itap’s folders: expert.ics.purdue.edu

If you are familiar with other types of SSH/SFTP clients (PuTTY), you can use them.
Basic commands:

- `ls`: list files and directories
- `rm file1`: remove file1
- `mv file1`: move file1
- `cp file1 file2`: copy file1 to file2.
- `cd dir`: change directory
- `mkdir dir`: make directory.
- `man command`: getting help of command.

Text editor

- Notepad.
- Vi, Pico.
- Emacs

There are many emacs references on the web:

Outline

1. Introduction: C Language
2. Basic Syntax
3. Control Flow
4. Functions
5. Pointers and Arrays
6. Structures
7. Compilation Process
```c
#include <stdio.h>

void main()
{
    printf("First Program\n");
}

```
Data Types

- **char**: single byte (one character).
- **int**: integer.
- **float**: single-precision floating point.
- **double**: double-precision floating point.

We can extend the **int**, **float** and **double** types by using **long** or **short**:

```c
long double sh;
short int ii;
```

with their signed and unsigned versions.
All the variables must be declared before use.

```c
int lower, upper, step;
char c, line[1000];
```

The variables can be initialized (defined) in their declaration:

```c
char esc='\\';
int i=0;
```

The variables can have a fixed value by using the `const` qualifier:

```c
const double pi=3.14;
const char msg[]="warning";
```
Operators

- Arithmetic operators: +, -, *, %
- Relational operators: >, >=, <, <= and ==, !=.
- Logical operators: && and ||.
- Increment and Decrement operators:
  - i++ means: i = i + 1.
  - i-- means: i = i - 1.
Assignment operators

- Notation: $\text{expr}_1=(\text{expr}_1) \ op \ (\text{expr}_2)$ is equivalent to $\text{expr}_1 \ op=\text{expr}_2$.

- Examples:
  - $i=i+2$ is equiv. to $i+=2$.
  - $x=x*(y+1)$ is equiv. to $x*=y+1$
printf: converts, formats and prints its arguments on the standard output device. Example:

```c
#include <stdio.h>

int main()
{
    printf ("Characters: %c %c \n", 'a', 65);
    printf ("Decimals: %d %ld\n", 1977, 650000L);
    printf ("Preceding with blanks: %10d \n", 1977);
    printf ("Preceding with zeros: %010d \n", 1977);
    printf ("floats: %4.2f %+.0e \n", 3.1416, 3.1416);
    printf ("%s \n", "A string");
    return 0;
}
```

More details on:

**scanf**: input analog of printf.

```c
#include <stdio.h>

int main ()
{
    char str [80];
    int i;

    printf ("Enter your family name: ");
    scanf ("%s",str);
    printf ("Enter your age: ");
    scanf ("%d",&i);
    printf ("Mr. %s , %d years old.\n",str,i);
    printf ("Enter a hexadecimal number: ");
    scanf ("%x",&i);
    printf ("You have entered %#x (%d).\n",i,i);

    return 0;
}
```
Outline

1 Introduction: C Language
2 Basic Syntax
3 Control Flow
4 Functions
5 Pointers and Arrays
6 Structures
7 Compilation Process
To express decisions. Syntax:

```cpp
if (expression)
    statement 1;
else
    statement 2;
```

And it is possible to construct else-if structures:

```cpp
if (expression)
    statement 1;
else if (expression)
    statement 2;
else if (expression)
    statement 3;
else
    statement 4;
```
Example 1:

```c++
if(n > 0)
    average = sum / n;
else{
    printf("can’t compute average\n");
    average = 0;
}
```

Example 2:

```c++
if(grade >= 90)
    printf("A");
else if(grade >= 80)
    printf("B");
else if(grade >= 70)
    printf("C");
else if(grade >= 60)
    printf("D");
else printf("F");
```
Loops: While and For

- Syntax:
  ```
  while(expression)
  statement
  ```

- Syntax:
  ```
  for(expr1; expr2; expr3)
  statement
  ```
Examples

- While statement:
  ```c
  int x = 2;
  
  while(x < 1000) {
    printf("%d\n", x);
    x = x * 2;
  }
  ```

- For statement:
  ```c
  for(i=0; i<n; i++) {
    k=i+3
    printf("%d\n", k);
  }
  ```
Useful statements within loops

- **Break**: provides an early exit from a loop. Example:

```cpp
while(t<=100){
    if(t%4==0)
        break;
    else
        t*=4;
}
```

- **Continue**: it causes the next iteration of a loop to begin. Example:

```cpp
for(i=0; i < n; i++){
    if(a[i]<0)
        continue;
    else
        sum+=a[i]
}
```
Functions

Some details:

- In C there are no procedures, just functions.
- The function called: “main” is where the program execution begins.
- The “return” statement is the mechanism for returning a value from the called function to the user.
- Up to now, the return values are of type: void (no value) and int. We can extend this to any type of variable.
return-type function-name(argument declarations)
{
    declarations and statements
}

Example:

int multbytwo(int x)
{
    return x * 2;
}
In C all the functions should be declared. There are two ways to declare a function:

- By using an `#include` statement. (Header files)
- By calling the function directly (Functions defined within the .c file or located within the current directory)

In order to understand these facts we need to study the scope rules further.
The functions and external variables that make up a C program need not all be compiled at the same time. You can keep several files and previously compiled routines can be loaded using libraries.

The scope of an internal variable is the function in which the name is defined. (the same is true for the parameters of a function)

The scope of an external variable lasts from the point which it is defined to the end of the file being compiled.

Important: any function is a variable!
extern int sp;

void main(){
    double val[10];
    sp=0;
}

void compute1(double f){...}

double compute2(int k,double g){...}
Scope Rules

- Important vocabulary:
  - “To define“ a variable: to announce the properties of a variable and to store that variable in memory.
  - ”To declare“ a variable: to announce the properties of a variable.

- If an external variable is to be referred before it is defined or if it is defined in a different .c file then you have to use `extern` command.

- You can have multiple declarations but only one definition.
Example

file 1:

double sp;
double val[1];
void compute1(double);

void main(){
    sp=1.20;
    val[0]=compute1(sp);
    ...
}

void compute1(double f){...}
Example

file2:

.extern int sp;
.extern double val[];
.extern void compute1(double);
.double compute2(int,double)

double compute2(double k,double g){
    double temp;
    temp=(abs(val[0])+k)/g;
    compute1(temp)
    ...
}
The problem with this structure is that it can become extremely complicated.

In order to solve this we can create Header functions (.h) containing only variable declarations.

We can call it by using #include "name.h".
Example

file 1:
#include "headerfile.h"
void main(){...}

file2:
int sp=0;
double val[10];
void compute1(double f){...}
double compute2(int k,double g){...}

header file:
const int sp;
const double val[];
double compute2(int,double)
void compute1(double)
Exercise

Simplify the structure of these functions using a header file:

```c
#include <stdio.h>

int multbytwo(int x)
{
    return x * 2;
}

int main()
{
    int i, j;
    i = 3;
    j = multbytwo(i);
    printf("%d\n", j);
    return 0;
}
```
Exercise

- Implement a function that computes $n!$.
- Using the previous function implement the binomial coefficient of $n$ and $k$.
- Implement a procedure that takes a positive number $p < 1$ and an integer $n$ and computes the CDF of a Binomial($n,p$).
Some references

- http://www.cplusplus.com/
- http://www.cprogramming.com/tutorial.html#ctutorial
Outline

1. Introduction: C Language
2. Basic Syntax
3. Control Flow
4. Functions
5. Pointers and Arrays
6. Structures
7. Compilation Process
- Pointer: variable that contains the address of a variable.
- Pointers and arrays are closely related.
- The operator & gives the address of an object. For example:
  \[ p = \&c \]
  assigns the address of c to the variable p. (p is pointing to c).
- The operator * is the “indirection” operator. When applied to a pointer it accesses the object the pointer points to.
- Pointer declaration:
  \[ \text{type } *\text{variable}_\text{name} \]
int x=3, y=1;
int *ip;

ip=&x;
y=*ip;
*ip=0;

Final values of x and y?
By default C passes arguments to functions by value.
The only way to change a variable in the calling function is by using pointers.
What is the problem with the following code?

```c
void swap(int x, int y){
    int temp;
    temp=x;
    x=y;
    y=temp;
}
```
We can fix the swap function using pointers.

```c
#include<stdio.h>

void swap(int *x, int *y){
    int temp;

    temp= *x;
    *x=*y;
    *y=temp;
}
```
- Syntax: type name[size]. Example:
  ```
  int a[10];
  ```
- This notation defines an array with elements \(a[0], a[1], \ldots, a[9]\).
- Suppose that \(pa\) is a pointer to \(a[0]\). Then we have to state:
  ```
  int *pa;
  pa = &a[0];
  ```
- Clearly, if \(i = 1, \ldots, 9\), then:
  - \(pa + i\) is the address of the component \(a[i]\) and therefore \(pa + i = &a[i]\).
  - \(a[i]\) has the same value than \(*(pa + i)\).
- Then there is a 1-1 relationship between pointers and arrays.
Then the relationship $\text{pa} = \& \text{a}[0]$ can be written as $\text{pa} = \text{a}$.

Then when array is passed to a function, what is passed is the location of the first element.

Example:

```c
int strlen(char *s){
    int n;
    for(n=0; *s != '\0'; s++)
        n++;
    return n;
}
```
As long as two pointers point to the same array:

- When you subtract them the result is the number of elements between those pointers.
- One pointer is greater than the other if it points beyond where the other one points.
- You can compare them by equality and inequality.
- Example:

```cpp
define array1[10], array2[10];
define *ip1, *ip2 = &array2[0];
define *ep = &array1[10];

for(ip1 = &array1[0]; ip1 < ep; ip1++)
    *ip2++ = *ip1;
```
Pointers to functions

- Syntax:
  
  ```
  type (*name)(arguments)
  ```

- Examples:
  
  ```
  double (*func)(double x);
  double (*myFunc)(double x);
  double (*AnyFuncName)(double x);
  ```
Outline

1. Introduction: C Language
2. Basic Syntax
3. Control Flow
4. Functions
5. Pointers and Arrays
6. Structures
7. Compilation Process
Collection of one or more variables. They are grouped under a single name.

Example:
```
struct point {
    int x;
    int y;
}
```

You can declare your structure as usual:
```
struct point pt={2,1};
```

And you can call its members:
```
printf("%d,%d", pt.x, pt.y);
```
We can do several operations on structures:

- Copy.
- Assign.
- Take its address.
- Access its members.
Pointers to structures

- We can declare pointers to structures, in the usual way:
  ```
  struct point *pp;
  ```
- Note that *pp is the structure and (*pp).x, (*pp).y are the members.
- Pointers to structures admit the following notation:
  ```
  (*pp).x = pp->x ;
  (*pp).y = pp->y ;
  ```
- Using structures we can introduce nice data schemes, like trees and linked lists. If you are interested there are a lot of web resources, for example:
  ```
  http://publications.gbdirect.co.uk/c_book/chapter6/structures.html
  ```
Outline

1. Introduction: C Language
2. Basic Syntax
3. Control Flow
4. Functions
5. Pointers and Arrays
6. Structures
7. Compilation Process
Compilation Process

- Source Files
- Source Files
- Source Files
  - C Preprocessor
  - C/C++ Compiler
  - Object Files
  - Object Files
  - Library Compiler
  - Library Files
  - Linker
  - Final Executable / Binary
1 Implement the Normal CDF using the well-known approximation of Abramowitz and Stegun (1964): For $x > 0$:

$$\Phi(x) \approx 1 - \phi(x)(b_1 t + b_2 t^2 + b_3 t^3 + b_4 t^4 + b_5 t^5)$$

where $t = \frac{1}{1+b_0 x}$, $\phi(x)$ is the standard normal pdf and:

<table>
<thead>
<tr>
<th>$b_0$</th>
<th>0.2316419</th>
</tr>
</thead>
<tbody>
<tr>
<td>$b_1$</td>
<td>0.319381530</td>
</tr>
<tr>
<td>$b_2$</td>
<td>-0.356563782</td>
</tr>
<tr>
<td>$b_3$</td>
<td>1.781477937</td>
</tr>
<tr>
<td>$b_4$</td>
<td>-1.821255978</td>
</tr>
<tr>
<td>$b_5$</td>
<td>1.330274429</td>
</tr>
</tbody>
</table>
Example 1

2 Create a function that computes the Black-Scholes price of an European Call option.

3 Compute the greeks of an European Call option, by implementing a C function. This function should include: Delta, Gamma, Theta, Vega and Rho.
Example 2

1. Implement the bisection method and the Newton-Raphson algorithm using pointers to functions.

2. Test your code with the function \( f(\sigma) = c_0 - c(S, K, \sigma, T) \) in order to compute the implied volatility.

3. Test your final code with the following data:
   - Stock Price=50
   - Strike price=50
   - Interest rate=0.10
   - Time=0.5
   - Observed option price=2.5