

Definizione di una funzione

Divide et impera

Per sviluppare e gestire un programma corposo il modo più efficiente è dividerlo in piccoli pezzi, moduli.

Questi moduli in C sono chiamati funzioni.

Caratteristiche delle function C ed esempi di function C

Argomenti trattati:

- ✓ definizione di una function C
- ✓ parametri e argomenti in C
- ✓ prototipo di una function C
- ✓ passaggio dei parametri per valore
- ✓ istruzione **return**
- ✓ esempi di function in C
- ✓ function C primitive
- ✓ header file C

function in C

```
float circon(float raggio)
{
    const float pi_greco = 3.1415926F;
    float risultato;
    risultato = 2.0F*pi_greco*raggio;
    return risultato;
}
```

function in C

```
float circon(float raggio)
{
    const float pi_greco = 3.1415926F;
    return 2.0F*pi_greco*raggio;
}
```

function in C

definizione di function

intestazione

tipo dell'output

parametro e tipo
del parametro

```
float circon (float raggio)
{
    const float pi_greco = 3.1415926F;
    float risultato;
    risultato = 2.0F*pi_greco*raggio;
    return risultato;
}
```

ritorno al chiamante
e passaggio del risultato

function in C

definizione di function

```
<intestazione>  
{  
  <corpo della function>  
}
```

(<tipo> <parametro>, <tipo> <parametro>, ..., <tipo> <parametro>)

```
<tipo output> <nome function> (<tipo> <parametro>)  
{  
  <corpo della function>  
}
```

function in C

definizione di function

```
<intestazione>  
{  
  <corpo della function>  
}
```

```
<intestazione>  
{  
  <parte dichiarativa>  
  <parte esecutiva>  
}
```

variabili **locali**
della function



function in C

attivazione (o chiamata)
di function

raggio è
l'argomento
della chiamata

```
#include <stdio.h>
float circon(float raggio);
void main ()
{
    float raggio, circonferenza;
    printf ("Inserire il raggio : ");
    scanf ("%f", &raggio);
    circonferenza = circon(raggio);
    printf ("circonferenza=%f\n", circonferenza);
}
```

function in C

chiamata di function

prototipo di function

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

```
float circon(float raggio);
```

```
void main ()
```

```
{
```

```
float raggio, circonferenza;
```

```
printf ("Inserire il raggio : ");
```

```
scanf ("%f", &raggio);
```

```
circonferenza = circon (raggio);
```

```
printf ("circonferenza=%f\n", circonferenza);
```

```
}
```

chiamata di function

il **prototipo** di una function C specifica il numero e il tipo dei **parametri**

```
float circon(float);
```

nel **prototipo** si possono omettere i nomi dei parametri

organizzazione di un programma C

unico file

```
#include <stdio.h>
float circon(float raggio) ;
void main ()
{
    float raggio, circonferenza;
    printf ("Inserire il raggio : ");
    scanf ("%f", &raggio) ;
    circonferenza = circon(raggio) ;
    printf ("circonferenza=%f\n", circonferenza) ;
}
float circon(float raggio)
{
    const float pi_greco = 3.1415926F;
    float risultato; ←
    risultato = 2.0F*pi_greco*raggio;
    return risultato;
}
```

variabile **locale**
della function

organizzazione di un programma C

più file

```
#include <stdio.h>
float circon(float raggio);
void main ()
{
    float raggio, circonferenza;
    printf ("Inserire il raggio : ");
    scanf ("%f", &raggio);
    circonferenza = circon(raggio);
    printf ("circonferenza=%f\n", circonferenza);
}
```

main.c

```
float circon(float raggio)
{
    const float pi_greco = 3.1415926F;
    float risultato;
    risultato = 2.0F*pi_greco*raggio;
    return risultato;
}
```

circon.c

Le definizioni di funzione

```
1 #include<stdio.h>
2
3 int quadrato(int);
4
5 int main (void){
6     int x;
7
8     for(x = 1; x<=10; x++)
9         printf("%d ", quadrato(x));
10    printf("/n");
11    return 0;
12 }
13
14
15 int quadrato(int q){
16     return q*q;
17 }
```

Le definizioni di funzione

```
1 #include<stdio.h>
2
3 int quadrato(int);
4
5 int main (void){
6     int x;
7
8     for(x = 1; x<=10; x++)
9         printf("%d ", quadrato(x));
10    printf("/n");
11    return 0;
12 }
13
```

```
int quadrato(int q){
    return q*q;
}
```



qualunque sia l'organizzazione del programma C:

una variabile dichiarata all'interno di una function è utilizzabile solo all'interno della function stessa;
essa risulta **indeterminata** (non dichiarata) al di fuori della function

regola di visibilità (*scope rule*):

gli identificatori sono accessibili solo all'interno del blocco nel quale sono dichiarati;
al di fuori del blocco essi risultano sconosciuti

programma C

insieme di comandi
al precompilatore

un **eventuale**
insieme di
dichiarazioni

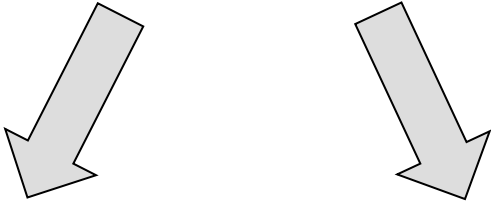
una function **main**

un **eventuale**
insieme di function

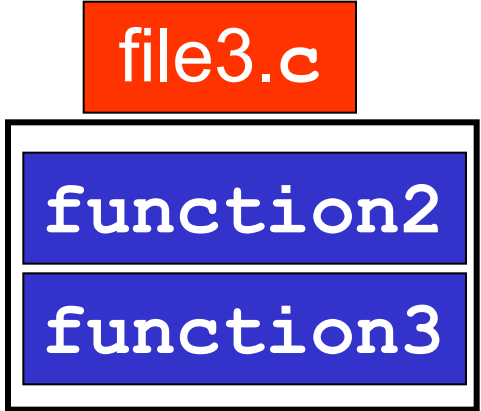
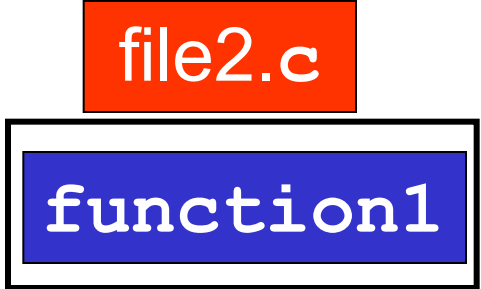
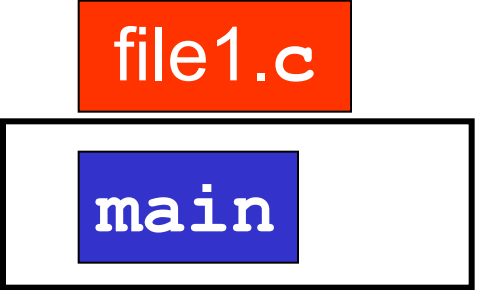
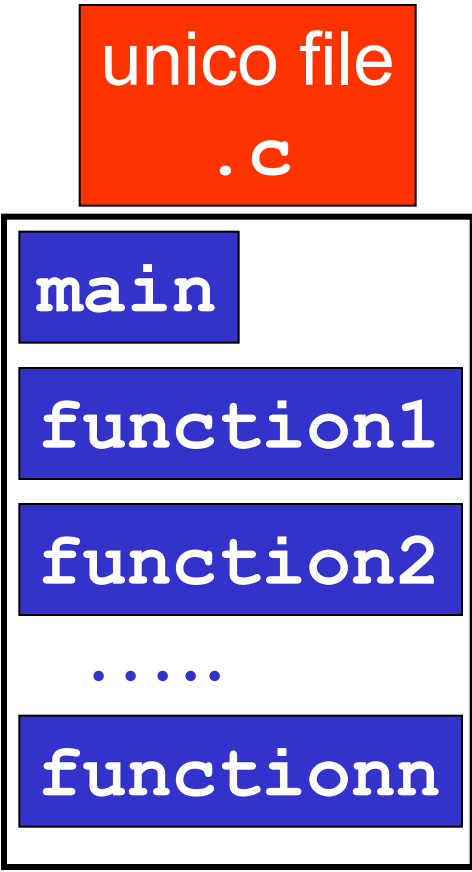
prototipi di function
variabili globali
tipi globali
....

l'esecuzione del programma inizia sempre dal **main**

programma C



compilazione unica



compilazione separata

.....

passaggio di parametri in C

sempre per valore

alla function viene passata una **copia** del valore dell'**argomento**

argomento e **parametro**
non sono la **stessa variabile**

l'eventuale variazione del **valore** del **parametro**
(nella function)

non ha nessun effetto
sul **valore** dell'**argomento** (nel chiamante)

passaggio di parametri in C

per valore

```
#include <stdio.h>
float circon(float);
void main ()
{
    float raggio_main, circonferenza;
    printf ("Inserire il raggio : ");
    scanf ("%f", &raggio_main);
    circonferenza = circon(raggio_main);
    printf ("circonferenza=%f\n", circonferenza);
}
float circon(float raggio)
{
    const float pi_greco = 3.1415926F;
    float risultato;
    risultato = 2.0F*pi_greco*raggio;
    return risultato;
}
```

passaggio di parametri in C

per valore

```
#include <stdio.h>
float circon(float);
void main ()
{
    float raggio_main, circonferenza;
    raggio_main = 100.0F;
    circonferenza = circon(raggio_main);
    printf ("raggio=%f\n",raggio_main);
    printf ("circonferenza=%f\n",circonferenza);
}
float circon(float raggio)
{
    const float pi_greco = 3.1415926F;
    float risultato;
    risultato = 2.0F*pi_greco*raggio;
    raggio = 0.0F;
    return risultato;
}
```

raggio=100.000000

circonferenza= 628.318481

Esercizi sulle funzioni

Esercizio 1

Un garage addebita un importo minimo di 2\$ per un parcheggio fino a 3 ore.

Oltre le 3 ore il parcheggio costa 50 cent in più per ogni ora.

L'addebito massimo è 10\$ per 24 ore.

Non è possibile parcheggiare più di 24 ore per volta.

Scrivete un programma che calcoli e visualizza gli addebiti per ognuno dei tre clienti che hanno parcheggiato nel garage ieri.

Dovete immettere le ore di parcheggio per ogni cliente.

Il vostro programma dovrà visualizzare i risultati con il calcolo anche del totale.

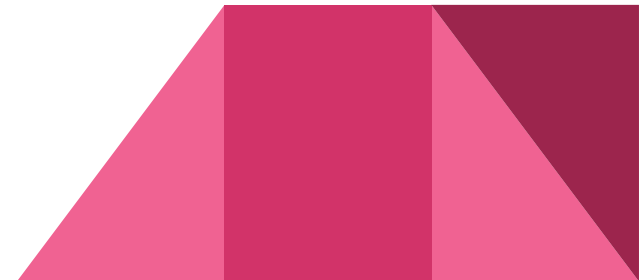
Il programma dovrà utilizzare la funzione **calcola_importo** per determinare l'addebito di ogni cliente. I risultati saranno più o meno così:



Esercizio 2

Modificare l'esercizio precedente in modo da avere n clienti.

n va richiesto all'inizio.



Esercizio 3

Scrivere un programma che dati in input ora e minuti restituisca il numero di secondi trascorsi dalla mezzanotte.



Esercizio:

Versione 1: `main` e `function` in **un unico** file `.c`

Versione 2: `main` e `function` in **due** file `.c` nello stesso **project**

scrivere una function `area_cerchio` che calcoli l'area di un cerchio, dato come parametro il raggio.

Scrivere un `main` che calcoli l'area di un cerchio (letto con `scanf` il suo raggio) richiamando la function `area_cerchio`

```
#include <stdio.h>
float area_cerchio(float);
void main ()
{
    float raggio_main, area;
    printf ("Inserire il raggio : ");
    scanf ("%f", &raggio_main);
    area = area_cerchio(raggio_main);
    printf ("l'area del cerchio di raggio %f e' :%f\n",
        raggio_main, area);
}
float area_cerchio(float raggio)
{
    const float pi_greco = 3.1415926F;
    return pi_greco*raggio*raggio;
}
```

Esercizio:

Versione 1: `main` e `area_cerchio` e `area_corona` in **un unico** file `.c`

scrivere una function `area_corona` che calcoli l'area di una corona circolare, dati come parametri il raggio minore e il raggio maggiore, richiamando la function `area_cerchio`.

Scrivere un `main` che calcoli l'area di una corona circolare (letti con `scanf` il raggio maggiore e il raggio minore) richiamando la function `area_corona`

```
#include <stdio.h>
float area_cerchio(float);
float area_corona(float, float);
void main ()
{
    float raggio_mag, raggio_min, area;
    printf ("Inserire il raggio maggiore : ");
    scanf ("%f", &raggio_mag);
    printf ("Inserire il raggio minore : ");
    scanf ("%f", &raggio_min);
    area = area_corona(raggio_min, raggio_mag);
    printf ("l'area della corona circolare di raggio minore %f e
raggio maggiore %f e' :%f\n", raggio_min, raggio_mag, area);
}
.....
```

Esercizio:

Versione 1: `main` e `area_cerchio` e `area_corona` in **un unico** file `.c`

scrivere una function `area_corona` che calcoli l'area di una corona circolare, dati come parametri il raggio minore e il raggio maggiore, richiamando la function `area_cerchio`.

Scrivere un `main` che calcoli l'area di una corona circolare (letti con `scanf` il raggio maggiore e il raggio minore) richiamando la function `area_corona`

```
.....  
float area_cerchio(float raggio)  
{  
    const float pi_greco = 3.1415926F;  
    return pi_greco*raggio*raggio;  
}  
float area_corona(float r_min, float r_mag)  
{  
    return area_cerchio(r_mag) - area_cerchio(r_min);  
}
```

Esercizio:

Versione 2: `main` in **un** file, `area_cerchio` e `area_corona` in **un** altro file `.c` nello stesso **project**

scrivere una function `area_corona` che calcoli l'area di una corona circolare, dati come parametri il raggio minore e il raggio maggiore, richiamando la function `area_cerchio`.

Scrivere un `main` che calcoli l'area di una corona circolare (letti con `scanf` il raggio maggiore e il raggio minore) richiamando la function `area_corona`

```
#include <stdio.h>
float area_corona(float, float);
void main ()
{
    float raggio_mag,raggio_min,area;
    printf ("Inserire il raggio maggiore : ");
    scanf ("%f",&raggio_mag);
    printf ("Inserire il raggio minore : ");
    scanf ("%f",&raggio_min);
    area = area_corona(raggio_min,raggio_mag);
    printf ("l'area della corona circolare di
           raggio minore %f eraggio maggiore
           %fe' :%f\n",raggio_min,raggio_mag,
           area);
}
```

file `main.c`

```
float area_cerchio(float);
float area_corona(float r_min,float
                 r_mag)
{
    return area_cerchio(r_mag) -
           area_cerchio(r_min);
}
float area_cerchio(float raggio)
{
    const float pi_greco = 3.1415926F;
    return pi_greco*raggio*raggio;
}
```

file `fun.c`

Esercizio:

scrivere una function `valore_assolutoF` che calcoli il valore assoluto di un numero `float`, dato come parametro

```
#include <stdio.h>
float valore_assolutoF(float);
void main ()
{
    float numero, val_ass_numero;
    printf ("Inserire un numero (float) : ");
    scanf ("%f", &numero);
    val_ass_numero = valore_assolutoF(numero);
    printf ("il valore assoluto di %f e' :%f\n", numero, val_ass_numero);
}
float valore_assolutoF(float x)
{
    if (x >= 0)
        return x;
    else
        return -x;
}
```

Esercizio:

scrivere una function `errore_relativoF` che calcoli l'errore relativo di tra due dati `float`, dati come parametri

$$\text{Errore relativo} = \frac{|x - y|}{|x|}$$

errore relativo di y rispetto a x

scrivere un `main` che richiami la function `errore_relativoF` e utilizzarlo per calcolare l'errore relativo per le coppie di numeri:

$$x=7.2, y=6.4$$

$$x=8.221222, y=8.221233$$

$$x=-3.21, y=-3.25$$

Esercizio:

scrivere una function `errore_relativoF` che calcoli l'errore relativo di tra due dati `float`, dati come parametri

```
#include <stdio.h>
float valore_assolutoF(float);
float errore_relativoF(float, float);
void main ()
{
    float x,y,errore_rel;
    printf ("Inserire valore di riferimento (float) :
");
    scanf ("%f", &x);
    printf ("Inserire approssimazione (float) : ");
    scanf ("%f", &y);
    errore_rel = errore_relativoF(x,y);
    printf ("l'errore relativo di %f rispetto a %f
        e' :%f\n", y,x,errore_rel);
}
.....
```

Esercizio:

scrivere una function `errore_relativoF` che calcoli l'errore relativo di tra due dati `float`, dati come parametri

```
.....  
float errore_relativoF(float x, float y)  
{  
    return valore_assolutoF(x-y)/valore_assolutoF(x) ;  
}  
float valore_assolutoF(float x)  
{  
    if (x >= 0)  
        return x;  
    else  
        return -x;  
}
```


Esercizio:

scrivere una function `distanza_origF` che calcoli la distanza dall'origine di un punto del piano, date come parametri le sue due coordinate

$$d_{origine} = \sqrt{x^2 + y^2}$$

scrivere un `main` che richiami la function `distanza_origF` e utilizzarlo per calcolare la distanza dall'origine dei punti con le seguenti coordinate:

$$x=2.1, y=4.2$$

$$x=-1.1, y=3$$

$$x=0, y=-100$$

Esercizio:

$$d_{origine} = \sqrt{x^2 + y^2}$$

scrivere una function `distanza_origF` che calcoli la distanza dall'origine di un punto del piano, date come parametri le sue due coordinate

```
#include <stdio.h>
#include <math.h>
float distanza_origF (float, float);
void main ()
{
    float x,y,distanza_origine;
    printf ("Inserire le coordinate del punto (float) : ");
    scanf ("%f%f", &x, &y);
    distanza_origine = distanza_origF(x,y);
    printf ("la distanza dall'origine del punto di
coordinate (%f,%f) e' :%f\n",x,y,distanza_origine);
}
float distanza_origF (float ascissa, float ordinata)
{
    return sqrt(ascissa*ascissa+ordinata*ordinata);
}
```

Esercizio:

$$d_{origine} = \sqrt{x^2 + y^2}$$

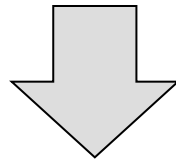
scrivere una function `distanza_origF` che calcoli la distanza dall'origine di un punto del piano, date come parametri le sue due coordinate

```
#include <stdio.h>
#include <math.h>
float distanza_origF (float, float);
void main ()
{
    float x,y,distanza_origine;
    printf ("Inserire le coordinate del punto (float) : ");
    scanf ("%f%f", &x, &y);
    distanza_origine = distanza_origF(x,y);
    printf ("la distanza dall'origine del punto di
coordinate (%f,%f) e' :%f\n",x,y,distanza_origine);
}
float distanza_origF (float ascissa, float ordinata)
{
    return sqrt(pow(ascissa,2)+pow(ordinata,2));
}
```

function C primitive

in C vi sono function predefinite per svolgere compiti specifici di utilità in programmazione

`printf`
`scanf`
`getchar`
`putchar`
`sqrt`
`pow`
.....



sono organizzate in **librerie**

una **libreria** C è un insieme di file;
ogni file contiene il codice oggetto di una
function C (insieme di file `.obj`, cioè
ottenuti compilando function C)

function C primitive

librerie

in C vi sono function predefinite per svolgere compiti specifici di utilità in programmazione

libreria standard di I/O	<code>stdio</code>
libreria di funzioni matematiche	<code>math</code>
libreria standard	<code>stdlib</code>
libreria per la misurazione dei tempi	<code>time</code>
.....	...

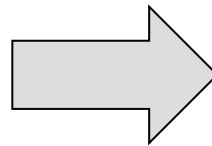
a ogni libreria è associato un **file di intestazioni** (*header file*)
che è un file testo contenente i **prototipi** delle function della libreria
(più eventuali altre informazioni necessarie per l'esecuzione delle function della libreria)

function C primitive

file di intestazioni

Se in un programma C (in un file `.c`) si utilizza una function primitiva di una libreria, è necessario includere il corrispondente file di intestazioni della libreria

```
printf  
scanf  
getchar  
putchar  
sqrt  
pow  
.....
```



```
<stdio.h>  
<stdio.h>  
<stdio.h>  
<stdio.h>  
<math.h>  
<math.h>  
.....
```

< >
per gli
header file
di librerie C

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

`#include` produce l'inserimento nel testo del programma, al posto del comando stesso, di una copia del file di intestazioni

function C

file di intestazioni

in un programma C (in un file `.c`) è possibile definire e includere file di intestazioni

```
#include "mioheader.h"
void main ()
{
    float raggio, circonferenza;
    printf ("Inserire il raggio : ");
    scanf ("%f", &raggio);
    circonferenza = circon(raggio);
    printf ("circonferenza=%f\n", circonferenza);
}
float circon(float raggio)
{
    const float pi_greco = 3.1415926F;
    float risultato;
    risultato = 2.0F*pi_greco*raggio;
    return risultato;
}
```

“ ”

per gli
header file
di utente

```
#include <stdio.h>
float circon(float raggio);
```

file di intestazioni
`mioheader.h`

```
# include <math.h>
double pow(a,b);
```

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

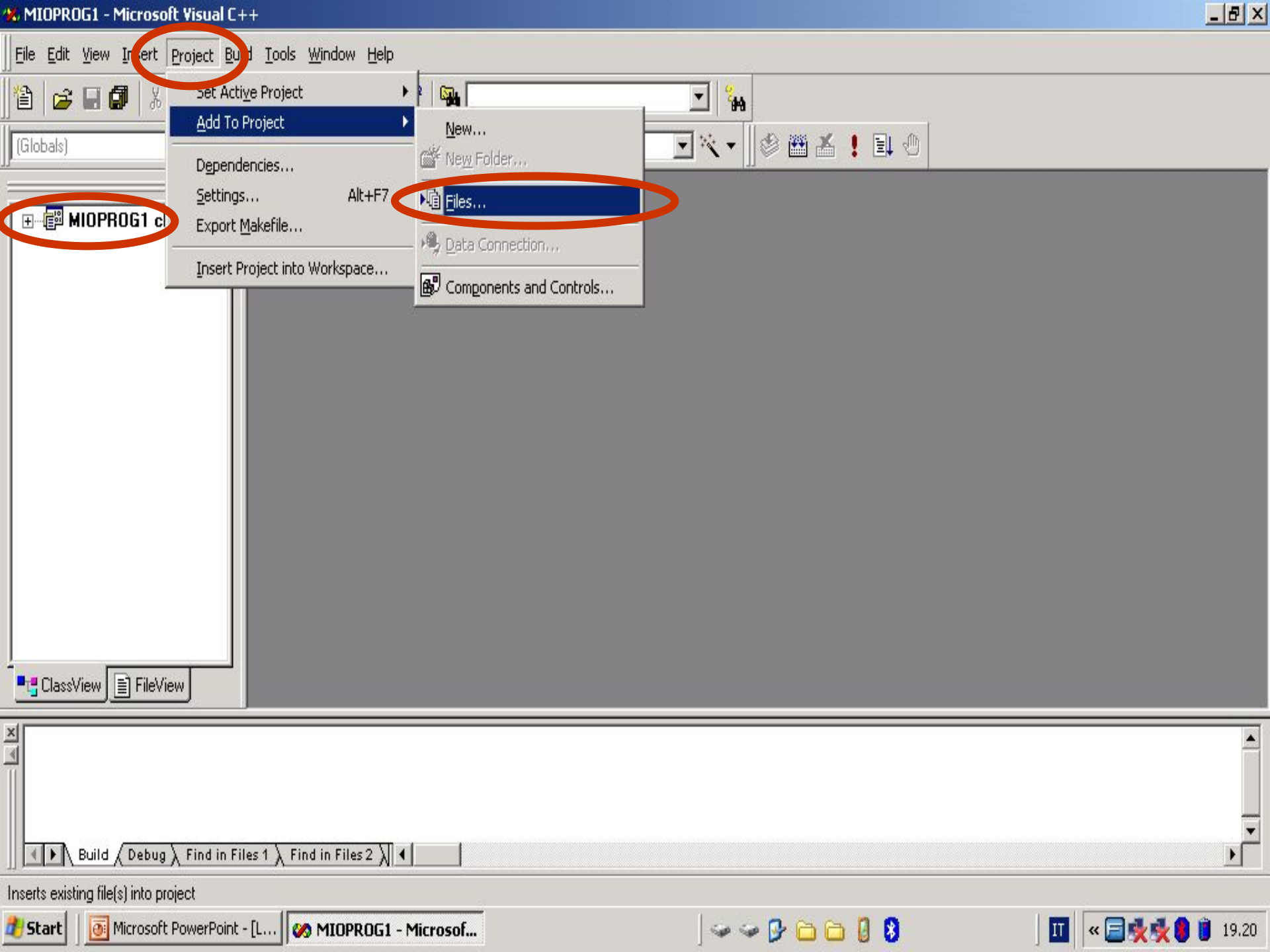
```
int main(void) {
    double numero;
    numero = pow(10, 2); // Calcola 10^2
    printf("%.2f\n", numero); // Stampa il risultato come numero decimale con
    due cifre decimali
    return 0;
}
```



```
# include <math.h>
double sqrt(x);
```

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

int main(void) {
    double numero;
    numero = sqrt(100); // Calcola radice quadrata di 100
    printf("%.2f\n", numero); // Stampa il risultato come numero decimale con
    // due cifre decimali
    return 0;
}
```



- Set Active Project
- Add To Project**
 - New...
 - New Folder...
 - Files...**
 - Data Connection...
 - Components and Controls...
- Dependencies...
- Settings... Alt+F7
- Export Makefile...
- Insert Project into Workspace...

MIOPROG1 c...

ClassView FileView

Build Debug Find in Files 1 Find in Files 2

Inserts existing file(s) into project

MIOPROG1 classes

ClassView | FileView

Insert Files into Project

Look in: MIOPROG1

Debug

File name: **miheader.h**

Files of type: C++ Files (.c;.cpp;.cxx;.tli;.h;.tlh;.inl;.rc)

Insert into: MIOPROG1

OK Cancel

estensione **.h**

