Capitolo 3 - Functions

Outline

Capitolo 3 - Functions

Outline

3.1 Introduction

- Divide and conquer
  - Construct a program from smaller pieces or components
  - Each piece more manageable than the original program

3.2 Program Components in C++

- Programs written by
  - combining new functions with "prepackaged" functions in the C++ standard library.
  - The standard library provides a rich collection of functions.
- Functions are invoked by a function call
  - A function call specifies the function name and provides information (as arguments) that the called function needs
  - Boss to worker analogy:

    A boss (the calling function or caller) asks a worker (the called function) to perform a task and return (i.e., report back) the results when the task is done.

3.2 Program Components in C++

- Function definitions
  - Only written once
  - These statements are hidden from other functions.
  - Boss to worker analogy:

    The boss does not know how the worker gets the job done; he just wants it done

3.3 Math Library Functions

- Math library functions
  - Allow the programmer to perform common mathematical calculations
  - Are used by including the header file **`<cmath>`**
- Functions called by writing

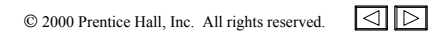
  functionName (*argument*)
- Example

```
cout << sqrt( 900.0 );
```

  - Calls the **`sqrt`** (square root) function. The preceding statement would print **`30`**
  - The **`sqrt`** function takes an argument of type **`double`** and returns a result of type **`double`**, as do all functions in the math library

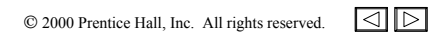
3.3 Math Library Functions

- Function arguments can be
  - Constants
    ```
    sqrt( 4 );
    ```
  - Variables
    ```
    sqrt( x );
    ```
  - Expressions
    ```
    sqrt( sqrt( x ) );
    sqrt( 3 - 6x );
    ```

3.4 Functions

- Functions
  - Allow the programmer to modularize a program
- Local variables
  - Known only in the function in which they are defined
  - All variables declared in function definitions are local variables
- Parameters
  - Local variables passed when the function is called that provide the function with outside information



3.5 Function Definitions

- Create customized functions to
  - Take in data
  - Perform operations
  - Return the result
- Format for function definition:

return-value-type function-name(*parameter-list*)
{
declarations and statements
}

- Example:

```
int square( int y)
{
  return y * y;
}
```


Outline

1. Function prototype
2. Loop
3. Function definition

```
1  // Fig. 3.3: fig03_03.cpp
2  // Creating and using a programmer-defined function
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  int square( int );   // function prototype
9
10 int main()
11 {
12    for ( int x = 1; x <= 10; x++ )
13       cout << square( x ) << "  ";
14
15    cout << endl;
16    return 0;
17 }
18
19 // Function definition
20 int square( int y )
21 {
22    return y * y;
23 }
```

Notice how parameters and return value are declared.

```
1  4  9  16  25  36  49  64  81  100
```

Program Output

```
1  // Fig. 3.4: fig03_04.cpp
2  // Finding the maximum of three integers
3  #include <iostream>
4
5  using std::cout;
6  using std::cin;
7  using std::endl;
8
9  int maximum( int, int, int );   // function prototype
10
11 int main()
12 {
13    int a, b, c;
14
15    cout << "Enter three integers: ";
16    cin >> a >> b >> c;
17
18    // a, b and c below are arguments to
19    // the maximum function call
20    cout << "Maximum is: " << maximum( a, b, c ) << endl;
```

Outline

1. Function prototype (3 parameters)

2. Input values

2.1 Call function

```
21
22    return 0;
23 }
24
25 // Function maximum definition
26 // x, y and z below are parameters to
27 // the maximum function definition
28 int maximum( int x, int y, int z )
29 {
30    int max = x;
31
32    if ( y > max )
33       max = y;
34
35    if ( z > max )
36       max = z;
37
38    return max;
39 }
```

```
Enter three integers: 22 85 17
Maximum is: 85

Enter three integers: 92 35 14
Maximum is: 92

Enter three integers: 45 19 98
Maximum is: 98
```

Outline

3. Function definition

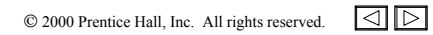
Program Output

3.6 Function Prototypes

- Function prototype
  - Function name
  - Parameters
    - Information the function takes in
  - Return type
    - Type of information the function passes back to caller (default **`int`**)
    - **`void`** signifies the function returns nothing
  - Only needed if function definition comes after the function call in the program
- Example:

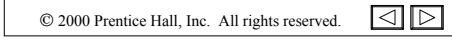
```
int maximum( int, int, int );
```

  - Takes in 3 **`int`**s
  - Returns an **`int`**



Convenzioni Notazionali

- Nomi di costante: interamente in maiuscolo
  - Es. UNO, SASSO, CARTA, GENNAIO, FEBBRAIO.
- Nomi di variabile: iniziale minuscola.
  - Es. mediaTotale, passivo, mese1, meseGennaio
- Nomi di funzione: iniziale minuscola
  - Es. fibonacci, fattoriale, f0, calcolaStipendio
- Nomi di tipo: iniziale maiuscola
  - Es. Mese, NumeroComplesso, Razionale



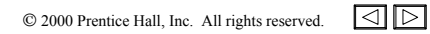
3.7 Header Files

- Header files
  - Contain function prototypes for library functions
  - **<cstdlib>** , **<cmath>**, etc.
  - Load with **#include <filename>**
    - Example:

```
#include <cmath>
```

- Custom header files
  - Defined by the programmer
  - Save as **filename.h**
  - Loaded into program using

```
#include "filename.h"
```

3.8 Random Number Generation

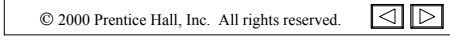
- **rand** function

```
i = rand();
```

  - Load **<cstdlib>**
  - Generates a pseudorandom number between **0** and **RAND_MAX** (usually 32767)
    - A pseudorandom number is a preset sequence of "random" numbers
    - The same sequence is generated upon every program execution
- **srand** function
  - Jumps to a seeded location in a "random" sequence

```
srand( seed );
srand( time( 0 ) );   //must include <ctime>
```

  - **time(0)**
    - The time at which the program was compiled
  - Changes the seed every time the program is compiled, thereby allowing **rand** to generate random numbers



3.8 Random Number Generation

- Scaling
  - Reduces random number to a certain range
  - Modulus (`%`) operator
    - Reduces number between 0 and **`RAND_MAX`** to a number between 0 and the scaling factor
  - Example

```
i = rand() % 6 + 1;
```

    - Generates a number between **`1`** and **`6`**

Outline

1. Define loop

2. Output random number

```
1  // Fig. 3.7: fig03_07.cpp
2  // Shifted, scaled integers produced by 1 + rand() % 6
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  #include <iomanip>
9
10 using std::setw;
11
12 #include <cstdlib>
13
14 int main()
15 {
16    for ( int i = 1; i <= 20; i++ ) {
17       cout << setw( 10 ) << ( 1 + rand() % 6 );
18
19       if ( i % 5 == 0 )
20          cout << endl;
21    }
22
23    return 0;
24 }
```

Notice **`rand() % 6.`** This returns a number between **`0`** and 5 (scaling)**.** Add 1 to get a number between **`1`** and **`6.`**

Executing the program again gives the same "random" dice rolls.

Program Output

```
    5     5     3     5     5
    2     4     2     5     5
    5     3     2     2     1
    5     1     4     6     4
```

```
1  // Fig. 3.9: fig03_09.cpp
2  // Randomizing die-rolling program
3  #include <iostream>
4
5  using std::cout;
6  using std::cin;
7  using std::endl;
8
9  #include <iomanip>
10
11 using std::setw;
12
13 #include <cstdlib>
14
15 int main()
16 {
17    unsigned seed;
18
19    cout << "Enter seed: ";
20    cin >> seed;
21    srand( seed );
22
23    for ( int i = 1; i <= 10; i++ ) {
24       cout << setw( 10 ) << 1 + rand() % 6;
25
26       if ( i % 5 == 0 )
27          cout << endl;
28    }
29
30    return 0;
31 }
```


Outline

1. Initialize seed

2. Input value for seed

2.1 Use `srand` to change random sequence

2.2 Define Loop

3. Generate and output random numbers

```
Enter seed: 67
         1         6         5         1         4
         5         6         3         1         2

Enter seed: 432
         4         2         6         4         3
         2         5         1         4         4

Enter seed: 67
         1         6         5         1         4
         5         6         3         1         2
```

Notice how the die rolls change with the seed.

Outline

Program Output

Monty Hall

Descrizione

Il gioco *Monty Hall* è così chiamato dal nome del conduttore di un gioco televisivo in cui un concorrente può vincere un'auto indovinando la porta dietro la quale si trova l'auto.

Il giocatore sceglie una porta, allora il conduttore ne apre un'altra dietro la quale non si trova l'auto.

A questo punto del gioco, il giocatore può scegliere se cambiare la sua scelta con la terza porta.

Progettazione

Rappresentiamo le porte con i numeri da 1 a 3. Per fare in modo che il gioco non sia sempre uguale, utilizziamo la funzione rand() per scegliere in modo casuale la porta (e quindi il numero) dietro la quale nascondere l'auto.

L'algoritmo dovrebbe eseguire le seguenti operazioni

•Stampa la spiegazione del gioco;

•Inizializza il seme per la generazione dei numeri casuali;

• Nascondi l'auto dietro una porta scelta in base ad un numero casuale estratto;

• Chiedi al giocatore di scegliere una porta;

• Determina la porta da aprire ed il cambio da proporre in base alla porta scelta ed a quella vincente;

• Proponi il cambio al giocatore ed acquisisci la sua scelta;

•Determina il risultato finale e comunicalo al giocatore.

```
#include<iostream>
#include<cstdlib> //#include<stdlib.h>
#include<time.h>

using std::cout;
using std::cin;
using std::endl;

void stampaIntroduzione();
void inizializzaSeme();
int estraiPortaACaso();
void determinaCambio(int, int, int &, int &)
int scegli();
int vuoiCambiare(int, int);
void risultatoFinale(int, int);

int main()
{
      int portaVincente, portaScelta, portaDaAprire, cambio;
      stampaIntroduzione();
      inizializzaSeme();
      portaVincente = estraiPortaACaso();
      portaScelta= scegli();
      determinaCambio(portaVincente, portaScelta, portaDaAprire, cambio);
```



```
  if (vuoiCambiare(portaDaAprire,cambio))
     portaScelta = cambio;
  risultatoFinale(portaScelta,portaVincente);
  return 0;
}

void stampaIntroduzione()
{
  cout<<" Questo è il gioco Monty Hall"<<endl;
  cout<<"Davanti a voi ci sono tre porte, una di esse ha dietro un'auto nuova.” <<endl
      <<"Scegliete una di queste porte; "<<endl
      <<"Ne vedrete aperta un'altra dietro la quale non si trova l'auto "
      <<“ed avrete la possibilità di cambiare la vostra scelta "<<endl;
}

void inizializzaSeme()
{
srand(time(0));
}

int estraiPortaACaso()
{
  int porta = 1+rand()%3;
  return porta;
}
```

```
int scegli()
{
    int scelta;
    do
    {
        cout<<"quale porta scegliete? (1|2|3)"<<endl;
        cin>>scelta;
    }
    while(scelta !=1 && scelta !=2 && scelta != 3);
    return scelta;
}

void determinaCambio(int portaV, int portaS, int& portaA, int & c)
{
    if (portaV == 1 && portaS == 1){ portaA = 3; c = 2;}
    if (portaV == 1 && portaS == 2){ portaA = 3; c = 1;}
    if (portaV == 1 && portaS == 3){ portaA = 2; c = 1;}
    if (portaV == 2 && portaS == 1){ portaA = 3; c = 2;}
    if (portaV == 2 && portaS == 2){ portaA = 1; c = 3;}
    if (portaV == 2 && portaS == 3){ portaA = 1; c = 2;}
    if (portaV == 3 && portaS == 1){ portaA = 2; c = 3;}
    if (portaV == 3 && portaS == 2){ portaA = 1; c = 3;}
    if (portaV == 3 && portaS == 3){ portaA = 2; c = 1;}
}
```



```
int vuoiCambiare(int portaA, int change)
{
    char risposta;
    cout<<"Viene aperta la porta "<<portaA<< " e si mostra che dietro non c'e' l'auto "<<endl;
    do
    {
    cout<<" Volete cambiare la vostra scelta con la porta "<<change<<" (s|S,n|N) ?"<<endl;
    cin>>risposta;
    }
    while(risposta != 's' && risposta != 'S' && risposta != 'n' && risposta != 'N');
    if (risposta == 's' || risposta == 'S')
        return 1;
    return 0;
}

void risultatoFinale(int portaS, int portaV)
{
    cout<<"l'auto è dietro la porta "<<portaV<<endl;
    cout<<"Poichè la vostra scelta è stata " <<portaS<<endl;
    if (portaS==portaV)
        cout<<"Avete vinto l'auto! "<<endl;
    else
        cout<<"non avete vinto " <<endl;
}
```

Output del programma

Questo è il gioco Monty Hall
Davanti a voi ci sono tre porte una di esse ha dietro un'auto nuova.
Scegliete una di queste porte;
Ne vedrete aperta un'altra dietro la quale non si trova l'auto ed avrete la possibilità
di cambiare la vostra scelta
quale porta scegliete? (1|2|3)
3
Viene aperta la porta 1 e si mostra che dietro non c'e' l'auto
Volete cambiare la vostra scelta con la porta 2?
n
L'auto è dietro la porta 2
Poichè la vostra scelta è stata 3 non avete vinto

Introducing `enum`

- Enumeration - set of integers with identifiers
  `enum` *typeName* {*constant1*, *constant2*...}**`;`**
  - Constants start at **`0`** (default), incremented by **`1`**
  - Unique constant names
  - Example:
    ```
    enum Status {CONTINUE, WON, LOST};
    ```
- Create an enumeration variable of type *typeName*
  - Variable is constant, its value may not be reassigned
    ```
    Status enumVar;  // create variable
    enumVar = WON;   // set equal to WON
    enumVar = 1;     // ERROR
    ```
- Enumeration constants can have values pre-set
  ```
  enum Months { JAN = 1, FEB, MAR, APR, MAY, JUN,
    JUL, AUG, SEP, OCT, NOV, DEC};
  ```
  - Starts at **`1`**, increments by **`1`**

29

Sasso, carta e forbici

Descrizione

Sasso, carta e forbici è un famoso gioco in cui due giocatori, simultaneamente, devono scegliere sasso, carta oppure forbici, indicando con la mano un segno che li rappresenta.

Il vincitore è il giocatore la cui scelta domina quella dell'altro in base alle seguenti regole:

- la carta domina il sasso
- le forbici tagliano la carta
- il sasso rompe le forbici

30

Progettazione

Utilizziamo due tipi enumerazione per rappresentare le scelte e il risultato:

enum Scelta {SASSO, CARTA, FORBICI}

enum Risultato{VINCE1, VINCE2, PARI}

Uno schema per l'algoritmo è il seguente:

1. stampa spiegazione
2. acquisisci le scelte dei giocatori
3. confronta le scelte
4. comunica chi è il vincitore

```
#include<iostream>

using std::cout;
using std::cin;
using std::endl;

enum Scelta {SASSO,CARTA,FORBICI};
enum Risultato {VINCE1, VINCE2, PARI};

void stampaDescrizione();
Scelta acquisisciScelta();
Risultato confrontaScelte(Scelta, Scelta);
void comunicaVincitore(Risultato);

int main()
{
      Scelta scelta1,scelta2;
      Risultato risultato;
      stampaDescrizione();
      cout<< "giocatore 1: "<<endl;
      scelta1 = acquisisciScelta();
      cout<< "giocatore 2: "<<endl;
      scelta2 = acquisisciScelta();
      risultato = confrontaScelte(scelta1, scelta2);
      comunicaVincitore(risultato);
      return 0;
}
```



```
void stampaDescrizione()
{
     cout<<"Questo è il famoso gioco Sasso, Carta o Forbici"<<endl;
     cout<<" Si gioca in due; i due giocatori devono scegliere simultaneamente"
          <<" una di queste tre cose: sasso, carta o forbici." << endl
          <<" Le seguenti regole stabiliscono chi è il vincitore "<<endl
          <<" la carta vince sul sasso "<<endl
          <<" il sasso rompe le forbici "<<endl
          <<" le forbici tagliano la carta"<<endl;
}
```


```
Scelta acquisisciScelta() {
    Scelta sceltaCorrente;
    int n;
    bool sceltaEffettuata = false;
    do
    {
        cout<< "scegli sasso(0), carta(1), o forbici(2): ";
        cin>>n;
        switch (n)
        {
            case 0:
                sceltaCorrente = SASSO;
                sceltaEffettuata = true;  break;
            case 1:
                sceltaCorrente = CARTA;
                sceltaEffettuata = true;  break;
            case 2:
                sceltaCorrente = FORBICI;
                sceltaEffettuata = true;  break;
            default:
                cout<<"Errore "<<endl;
        }
    }
    while (!sceltaEffettuata);
    return sceltaCorrente;
}
```



```
Risultato confrontaScelte( Scelta scelta1, Scelta scelta2)    //  versione 1
{
    Risultato risultato;
    if (scelta1 == scelta2)
        risultato = PARI;
    else if (scelta1 == SASSO)
        if (scelta2 == CARTA)
            risultato = VINCE2;
        else
            risultato = VINCE1;
    else if (scelta1 == CARTA)
        if (scelta2 == SASSO)
            risultato = VINCE1;
        else
            risultato = VINCE2;
    else //scelta1 == FORBICI
        if (scelta2 == SASSO)
            risultato = VINCE2;
        else
            risultato = VINCE1;
    return risultato;
}
```


```
Risultato confrontaScelte( Scelta scelta1, Scelta scelta2)  //  versione 2
{
        Risultato risultato;
        switch (scelta1)
        {
                case SASSO:
                        switch (scelta2)
                        {
                                case SASSO:
                                        risultato = PARI;
                                        break;
                                case FORBICI:
                                        risultato = VINCE1;
                                        break;
                                case CARTA:
                                        risultato = VINCE2;
                                        break;
                        }
                        break;
                case FORBICI:
                        switch (scelta2)
                        {
                                case SASSO:
                                        risultato = VINCE2;
                                        break;
```



```
                        case FORBICI:
                             risultato = PARI;
                             break;
                        case CARTA:
                           risultato = VINCE1;
                           break;
                        }
                        break;
                case CARTA:
                        switch (scelta2)
                        {
                        case SASSO:
                             risultato = VINCE1;
                             break;
                        case FORBICI:
                             risultato = VINCE2;
                             break;
                        case CARTA:
                             risultato = PARI;
                             break;
                        }
                        break;
                }
        return risultato;
}
```

```
void comunicaVincitore(Risultato risultato)
{
        if (risultato == PARI)
                cout<< " Avete pareggiato. "<<endl;
        else if (risultato == VINCE1)
                cout<<"Vince il giocatore1 "<<endl;
        else
                cout<<"Vince il giocatore1 "<<endl;
}
```

Output del programma

```
Questo è il famoso gioco Sasso, Carta o Forbici.
Si gioca in due; i due giocatori devono scegliere simultaneamente una di queste tre cose:
 sasso, carta o forbici.
Le seguenti regole stabiliscono chi è il vincitore
la carta vince sul sasso
il sasso rompe le forbici
le forbici tagliano la carta
Giocatore1:
Scegli sasso (0), carta (1),  forbici(2):  1
Giocatore2:
Scegli sasso (0), carta (1),  forbici(2):1
Avete pareggiato.
```


Il gioco dei dadi CRAPS

Descrizione

Il gioco dei dadi CRAPS viene giocato con due dadi. Ogni volta che i dadi vengono gettati, ne vengono sommati i due numeri ottenuti; la somma sar_ un intero compreso tra 2 e 12.

Il giocatore vince immediatamente se ottiene come punteggio un 7 oppure un 11; perde immediatamente se ottiene 2, 3 oppure 12.

Se il giocatore ottiene 4, 5, 6, 8, 9, 10 deve ricordare questo punteggio (sia P) e ripetere il lancio dei dadi finch_ vince ottenendo ancora come punteggio P oppure perde ottenendo un 7.

Progettazione

Simuliamo il lancio dei dadi, utilizzando la funzione rand().

I passi che l'algoritmo dovrebbe eseguire sono i seguenti:

- Stampare la descrizione del gioco;
- Simulare il lancio dei dadi;
- Decidere in base al punteggio ottenuto se il giocatore ha vinto, perso o deve ritentare.
  - Nel caso in cui sia necessario ritentare, rilanciare ripetutamente i dadi finch_ non si ottiene un 7 oppure non si ottiene nuovamente il punteggio precedentemente.

39

```
#include <iostream>
#include <cstdlib>
#include <time.h>

using std::cout;
using std::endl;

enum Esito {VINCI, PERDI, RITENTA};

void inizializzaSeme();
int lancialDadi();
Esito determinaEsitoDelGioco( int punteggio );
void haiVinto();
void haiPerso();
void ritenta(int);
```

40

```
int main()
{
    inizializzaSeme();
    int punteggio = lancialDadi();
    cout<< "Hai ottenuto "<<punteggio<<endl;
    Esito esito = determinaEsitoDelGioco(punteggio);
    if ( esito == VINCI)
        haiVinto();
    else if (esito == PERDI)
        haiPerso();
    else ritenta(punteggio);
    return 0;
}

void inizializzaSeme()
{
    srand(time(0));
}
```

41

```
int lancialDadi()
{
    int dado1 =1 + rand()%6;
    int dado2 =1 + rand()%6;
    int punti = dado1+dado2;
    return punti;
}

Esito determinaEsitoDelGioco(int punteggio)
{
    Esito esito;
    if( punteggio == 7 || punteggio == 11)
        esito = VINCI;
    else if ( punteggio == 2 || punteggio ==3  || punteggio == 12)
        esito = PERDI;
    else  esito = RITENTA;
    return esito;
}

void haiVinto()
{
    cout << " Hai Vinto "<<endl;
}
```

42

```
void haiPerso()
{
    cout << " Hai Perso "<<endl;
}

void ritenta( int punteggio)
{
    int punt =0;
    while ( punt != 7 && punt != punteggio )
    {
        cout <<"Devi rilanciare i dadi "<<endl;
        punt = lancialDadi();
        cout <<"Hai ottenuto " <<punt<<endl;
    }
    if (punt == punteggio)
        haiVinto();
    else
        haiPerso();
}
```


3.10 Storage Classes

- Storage class specifiers
  - Storage class
    - Where object exists in memory
  - Scope
    - Where object is referenced in program
  - Linkage
    - Where an identifier is known
- Automatic storage
  - Object created and destroyed within its block
  - **`auto`**
    - Default for local variables.
    - Example:

```
auto float x, y;
```

  - **`register`**
    - Tries to put variables into high-speed registers
  - Can only be used with local variables and parameters

3.10 Storage Classes

- Static storage
  - Variables exist for entire program execution
  - **`static`**
    - Local variables defined in functions
    - Keep value after function ends
    - Only known in their own function
  - **`Extern`**
    - Default for global variables and functions.
    - Known in any function

3.11 Identifier Scope Rules

- File scope
  - Defined outside a function, known in all functions
  - Examples include, global variables, function definitions and functions prototypes
- Function scope
  - Can only be referenced inside a function body
  - Only labels (**`start:`**, **`case:`**, etc.)
- Block scope
  - Declared inside a block. Begins at declaration, ends at }
  - Variables, function parameters (local variables of function)
  - Outer blocks "hidden" from inner blocks if same variable name
- Function prototype scope
  - Identifiers in parameter list
  - Names in function prototype optional, and can be used anywhere

```
1  // Fig. 3.12: fig03 12.cpp
2  // A scoping example
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  void a( void );   // function prototype
9  void b( void );   // function prototype
10 void c( void );   // function prototype
11
12 int x = 1;        // global variable
13
14 int main()
15 {
16    int x = 5;     // local variable to main
17
18    cout << "local x in outer scope of main is " << x << endl;
19
20    {              // start new scope
21       int x = 7;
22
23       cout << "local x in inner scope of main is " << x << endl;
24    }              // end new scope
25
26    cout << "local x in outer scope of main is " << x << endl;
27
28    a();           // a has automatic local x
29    b();           // b has static local x
30    c();           // c uses global x
31    a();           // a reinitializes automatic local x
32    b();           // static local x retains its previous value
33    c();           // global x also retains its value
34
```

x is different inside and outside the block.

```
local x in outer scope of main is 5
local x in inner scope of main is 7
local x in outer scope of main is 5
```


Outline

1. Function prototypes

1.1 Initialize global variable

1.2 Initialize local variable

1.3 Initialize local variable in block

2. Call functions

3. Output results

```
35    cout << "local x in main is " << x << endl;
36
37    return 0;
38 }
39
40 void a( void )
41 {
42    int x = 25;  // initialized each time a is called
43
44    cout << endl << "local x in a is " << x
45         << " after entering a" << endl;
46    ++x;
47    cout << "local x in a is " << x
48         << " before exiting a" << endl;
49 }
50
51 void b( void )
52 {
53    static int x = 50;  // Static initialization only
54                        // first time b is called.
55    cout << endl << "local static x is " << x
56         << " on entering b" << endl;
57    ++x;
58    cout << "local static x is " << x
59         << " on exiting b" << endl;
60 }
61
62 void c( void )
63 {
64    cout << endl << "global x is " << x
65         << " on entering c" << endl;
66    x *= 10;
67    cout << "global x is " << x << " on exiting c" << endl;
68 }
```

Local automatic variables are created and destroyed each time **a** is called.

```
local x in a is 25 after entering a
local x in a is 26 before exiting a
```

Local static variables are not destroyed when the function ends.

```
local static x is 50 on entering b
local static x is 51 on exiting b
```

Global variables are always accessible. Function **c** references the global **x.**

```
global x is 1 on entering c
global x is 10 on exiting c
```


Outline

1 Define Functions

Outline

Program Output

```
local x in outer scope of main is 5
local x in inner scope of main is 7
local x in outer scope of main is 5

local x in a is 25 after entering a
local x in a is 26 before exiting a

local static x is 50 on entering b
local static x is 51 on exiting b

global x is 1 on entering c
global x is 10 on exiting c

local x in a is 25 after entering a
local x in a is 26 before exiting a

local static x is 51 on entering b
local static x is 52 on exiting b

global x is 10 on entering c
global x is 100 on exiting c
local x in main is 5
```


3.15 Functions with Empty Parameter Lists

- Empty parameter lists
  - Either writing **`void`** or leaving a parameter list empty indicates that the function takes no arguments

    ```
    void print();
    ```

    or

    ```
    void print( void );
    ```

  - Function **`print`** takes no arguments and returns no value

```
1  // Fig. 3.18: fig03_18.cpp
2  // Functions that take no arguments
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  void function1();
9  void function2( void );
10
11 int main()
12 {
13    function1();
14    function2();
15
16    return 0;
17 }
18
19 void function1()
20 {
21    cout << "function1 takes no arguments" << endl;
22 }
23
24 void function2( void )
25 {
26    cout << "function2 also takes no arguments" << endl;
27 }
```

Notice the two ways of declaring no arguments.

Outline

1. Function prototypes (take no arguments)
2. Call the functions
3. Function definitions

Program Output

```
function1 takes no arguments
function2 also takes no arguments
```


3.17 References and Reference Parameters

- Call by value
  - Copy of data passed to function
  - Changes to copy do not change original
  - Used to prevent unwanted side effects
- Call by reference
  - Function can directly access data
  - Changes affect original
- Reference parameter alias for argument
  - **&** is used to signify a reference

    ```
    void change( int &variable )
       { variable += 3; }
    ```

  - Adds 3 to the variable inputted

    int y = &x.
  - A change to **y** will now affect **x** as well

```
1  // Fig. 3.20: fig03_20.cpp
2  // Comparing call-by-value and call-by-reference
3  // with references.
4  #include <iostream>
5
6  using std::cout;
7  using std::endl;
8
9  int squareByValue( int );
10 void squareByReference( int & );
11
12 int main()
13 {
14    int x = 2, z = 4;
15
16    cout << "x = " << x << " before squareByValue\n"
17         << "Value returned by squareByValue: "
18         << squareByValue( x ) << endl
19         << "x = " << x << " after squareByValue\n" << endl;
20
21    cout << "z = " << z << " before squareByReference" << endl;
22    squareByReference( z );
23    cout << "z = " << z << " after squareByReference" << endl;
24
25    return 0;
26 }
27
28 int squareByValue( int a )
29 {
30    return a *= a;   // caller's argument not modified
31 }
```

Notice the use of the **&** operator

Outline

1. Function prototypes

1.1 Initialize variables

2. Print `x`

2.1 Call function and print `x`

2.2 Print `z`

2.3 Call function and print `z`

3. Function Definition of `squareByValue`

Outline

3.1 Function Definition of `squareByReference`

```
32
33 void squareByReference( int &cRef )
34 {
35    cRef *= cRef;    // caller's argument modified
36 }
```

Program Output

```
x = 2 before squareByValue
Value returned by squareByValue: 4
x = 2 after squareByValue

z = 4 before squareByReference
z = 16 after squareByReference
```

3.16 Inline Functions

- **`inline`** functions
  - Reduce function-call overhead
  - Asks the compiler to copy code into program instead of using a function call
  - Compiler can ignore **`inline`**
  - Should be used with small, often-used functions
- Example:

```
inline double cube( const double s )
   { return s * s * s; }
```


3.18 Default Arguments

- If function parameter omitted, gets default value
  - Can be constants, global variables, or function calls
  - If not enough parameters specified, rightmost go to their defaults
- Set defaults in function prototype

```
int defaultFunction( int x = 1,
   int y = 2, int z = 3 );
```


```
1  // Fig. 3.23: fig03_23.cpp
2  // Using default arguments
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  int boxVolume( int length = 1, int width = 1, int height = 1 );
9
10 int main()
11 {
12    cout << "The default box volume is: " << boxVolume()
13         << "\n\nThe volume of a box with length 10,\n"
14         << "width 1 and height 1 is: " << boxVolume( 10 )
15         << "\n\nThe volume of a box with length 10,\n"
16         << "width 5 and height 1 is: " << boxVolume( 10, 5 )
17         << "\n\nThe volume of a box with length 10,\n"
18         << "width 5 and height 2 is: " << boxVolume( 10, 5, 2 )
19         << endl;
20
21    return 0;
22 }
23
24 // Calculate the volume of a box
25 int boxVolume( int length, int width, int height )
26 {
27    return length * width * height;
28 }
```


Outline

1. Function prototype
2. Print default volume
   2.1 Print volume with one parameter
   2.2 Print with 2 parameters
   2.3 Print with all parameters.
3. Function definition

Outline

Program Output

```
The default box volume is: 1

The volume of a box with length 10,
width 1 and height 1 is: 10

The volume of a box with length 10,
width 5 and height 1 is: 50

The volume of a box with length 10,
width 5 and height 2 is: 100
```

Notice how the rightmost values are defaulted.

3.19 Unary Scope Resolution Operator

- Unary scope resolution operator (`::`)
  - Access global variables if a local variable has same name
  - not needed if names are different
  - instead of **`variable`** use **`::variable`**

Outline

1. Define variables

2. Print variables

```
1  // Fig. 3.24: fig03 24.cpp
2  // Using the unary scope resolution operator
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  #include <iomanip>
9
10 using std::setprecision;
11
12 const double PI = 3.14159265358979;
13
14 int main()
15 {
16    const float PI = static cast< float >( ::PI );
17
18    cout << setprecision( 20 )
19         << "  Local float value of PI = " << PI
20         << "\nGlobal double value of PI = " << ::PI << endl;
21
22    return 0;
23 }
```

Notice the use of `::`

Program Output

```
Local float value of PI = 3.141592741012573242
Global double value of PI = 3.141592653589790007
```

3.20 Function Overloading

- Function overloading
  - Having functions with same name and different parameters
  - Should perform similar tasks (i.e., a function to square **int**s, and function to square **float**s).

```
int square( int x) {return x * x;}
float square(float x) { return x * x; }
```

  - Program chooses function by signature
    - signature determined by function name and parameter types
  - Can have the same return types

Outline

```
1  // Fig. 3.25: fig03_25.cpp
2  // Using overloaded functions
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  int square( int x ) { return x * x; }
9
10 double square( double y ) { return y * y; }
11
12 int main()
13 {
14    cout << "The square of integer 7 is " << square( 7 )
15         << "\nThe square of double 7.5 is " << square( 7.5 )
16         << endl;
17
18    return 0;
19 }
```

Functions have same name but different parameters

1. Define overloaded function

2. Call function

```
The square of integer 7 is 49
The square of double 7.5 is 56.25
```

Program Output

3.21 Function Templates

- Function templates
  - Compact way to make overloaded functions
  - Keyword **`template`**
  - Keyword **`class`** or **`typename`** before every formal type parameter (built in or user defined)

```
template < class T >
   // or template< typename T >
T square( T value1 )
{
   return value1 * value1;
}
```

  - **`T`** replaced by type parameter in function call.

```
int x;
int y = square(x);
```

    - If **`int`**, all **`T`**'s become **`int`**s
    - Can use **`float`**, **`double`**, **`long`**...