User Id : weizhe.goh@digipen.edu | Started : 2020.06.12 01:57:57 | Score : 75%

DigiPen

Functions

Practice

.

Review

- In C/C++, a function is a block of code which only runs when it is called by using the function call operator ().
- Functions are used to perform certain actions, and they are the most common way of reusing code.
- You can pass data, known as parameters, into a function.
  - Data you pass into a function is always passed as a **copy**
- A function can return a single value using return statement.

Example 1

Run

```
#include <iostream>

char abc(char ch1, char ch2) {
   char ch3;
   ch3 = ch2-ch1;
   return ch3;
}

int main()
{
   char c = abc('a', 'x');
   std::cout << c;
   return 0;
}
```

Default Parameters

- In C++, you can give a default value for a parameter in a function.
- if you leave out this argument in the function call, it takes the default value.
- To do so, follow the parameter declaration with the = symbol followed by the default value:

```
char abc(char ch1, char ch2='x');
```

Example 2

Run

```
#include <iostream>

int & inc(int & value, int amount = 1)
{
  value += amount;
  return value;
}

int main(void)
{
  int i = 10;
  std::cout << inc(i) << std::endl;
  std::cout << inc(i, 2) << std::endl;
  return 0;
}
```

```
11
13
```

★ Multiple default parameters 1 ☑

- A function can have multiple default parameters. However, all default parameters must be at the end of the parameter list.

Check all correct declarations of functions.

- [ ] `void foo(int a = 0, int b, int c);`
- [ ] `void foo(int a, int b = 3, int c = b);`
- [x] `void foo(int a, int b = 3, int c = 6);`
- [ ] `void foo(int a, int b = 3, int c);`
- [x] `void foo(int a, int b, int c);`

Overloaded Functions

Consider the following code. Do you think the output is correct?

Run

```
#include <iostream>

int cube(int n)
{
  return n * n * n;
}

int main()
{
  int i = 8;
  long l = 50L;
  float f = 2.5F;
  double d = 3.14;
  std::cout << cube(i) << std::endl
    << cube(l) << std::endl << cube(f)
    << std::endl << cube(d);
  return 0;
}
```

```
512
125000
8
27
```

Solution 1 (C)

```
int cube_int(int n)
{
  return n * n * n;
}
long cube_long(long n)
{
  return n * n * n;
}
float cube_float(float n)
{
  return n * n * n;
}
double cube_double(double n)
{
  return n * n * n;
}
```

- What do you think about this solution? Pros and cons?

Solution 2 (C++)

```
int cube(int n)
{
  return n * n * n;
}
long cube(long n)
{
  return n * n * n;
}
float cube(float n)
{
  return n * n * n;
}
double cube(double n)
{
  return n * n * n;
}
```

- This is the C++ solution using **overloaded functions** cube.

Problem 1 Solution

Consider the following code. What is the output? Do you think the output is correct?

Run

```
#include <iostream>

int cube(int n)
{
  return n * n * n;
}
long cube(long n)
{
  return n * n * n;
}
float cube(float n)
{
  return n * n * n;
}
double cube(double n)
{
  return n * n * n;
}

int main()
{
  int i = 8;
  long l = 50L;
  float f = 2.5F;
  double d = 3.14;
  std::cout << cube(i) << std::endl
    << cube(l) << std::endl << cube(f)
    << std::endl << cube(d);
  return 0;
}
```

```
512
125000
15.625
30.9591
```

Overloaded Functions Facts

- Overloaded functions have the same name but different parameters.
- There are 3 attributes of parameters: type, number and order.
- These are all valid overloads:

```
void foo(double);
void foo(int);
void foo(int, double);
void foo(double, int);
void foo(void);
```

★ Overloading And Ambiguity ☐

- **Ambiguity** in function overloading is the situation when the compiler can not choose a function among two or more overloaded functions.

```
int foo(int);
double foo(int); // Ambiguous
```

- Another example with ambiguit problemy:

```
void foo(float);
void foo(double);
...
foo(1.5F); // calls foo(float)
foo(1.5); // calls foo(double)
foo(1); // Error!
```

- These are technically ambiguous:

```
void foo(int);
void foo(int &);
```

- However, these are considered different:

```
void foo(int *);
void foo(const int *);
```

Example 2

Run

```
#include <iostream>

void foo(int &) {
 std::cout << "int&\n";
}

void foo(const int &) {
 std::cout << "const int&\n";
}

int main()
{
  int i = 1;
  const int j = 2;
  foo(5);
  foo(i);
  foo(j);
  foo(i + j);
  foo((int &)j);
  return 0;
}
```

```
const int&
int&
const int&
const int&
int&
```

★ Overloading And Defaults 1 ☑

- There may also be problems with overloading and defaults.

Given the following overloaded function declarations:

```
void foo(int a, int b = 10);
void foo(int a);
```

Check function calls that you think are not compilable.

- ■ `foo(8);`
- ■ `foo(7, 2, 8);`
- ☐ `foo(5, 6);`

Inline Functions

- Normally when a function is called
  - Program execution transfers to the function.
  - When the function is complete, program execution returns to the calling function.
- You can declare a function inline, such that the compiler replaces the function call with the function's implementation.
  - Could increase program speed (avoids the program execution transfers).
  - Could increase program size.

Example 3

- To make a function inline, use the **inline** keyword before either the function prototype or the function definition (or both)

Run

```
#include <iostream>

inline double square(double x)
{
  return x * x;
}

int main()
{
  for (int i = 0; i < 20; i++)
    std::cout << square(i) << std::endl;
  return 0;
}
```

```
0
1
4
9
16
25
36
49
64
81
100
121
144
169
196
225
256
289
324
361
```

Rule of thumb

- Consider using an inline function if
  - The function can be written in one line and
  - The function is called often (ex: in a loop)
- The inline keyword is a "suggestion". The compiler need not make your function inline

★ Pop quiz 1 ☑

What is the correct way to define a function inline?

☐
```
inline double square(double x) {
  return x * x;
}
```

☐
```
double square(double x) inline {
  return x * x;
}
```

☐
```
double square(double x) {
  return x * x;
} inline;
```

By signing this document you fully agree that all information provided therein is complete and true in all respects.

Responder sign: