

Imparare a programmare con Haiku

Lezione 3

Scritto da DarkWyrn

Traduzione in italiano, per Haiku Italia, a cura di

Giuseppe Gargaro & Francesca Mora

Finora abbiamo appreso le basi della programmazione, in che modo scrivere una funzione e come iniziare a cercare i bug del codice. Questa volta inizieremo a conoscere i diversi tipi di dati e i modi per memorizzarli e farli girare.

Con la manipolazione diretta si possono fare solo un tot di cose come `return 1 + 1;` che è il motivo per cui abbiamo le **variabili** in programmazione. Una variabile è semplicemente un contenitore per conservare l'informazione, e come qualsiasi contenitore reale, può avere diverse forme, dimensioni, e utilizzo. La creazione di una variabile è semplicemente una questione di dichiarazione della sua esistenza, come si vede qui sotto:

```
#include <stdio.h>

int main(void)
{
    // Dichiarazione di una variabile intera di nome a
    int a;

    // Dichiarazione di due in una volta: b e c
    int b, c;

    a = 1;
    b = 2;
    c = 3;

    // stampa i valori delle nostre variabili
    printf("a is %d, b is %d, and c is %d.\n",a,b,c);

    return a + b + c;
}
```

Le variabili possono essere dichiarate una alla volta, come la nostra variabile `a`, o più in una volta sola, come `b` e `c`. Oltre a dichiarare le variabili per il nostro uso nelle funzioni, molte volte ci arriveremo automaticamente poiché molte funzioni richiedono dati di input per fare il loro lavoro. Queste variabili sono chiamate **parametri** o **argomenti**.

I parametri della funzione sono dichiarati sia nella dichiarazione della funzione che nella definizione. Essi sono elencati in una lista separata da virgole. Le funzioni che non richiedono nessun argomento utilizzano la parola `void` per dirlo.

```
//Dichiarazione di una funzione con due argomenti.
int SomeFunction(int someNumber, int anotherNumber);

// Definizione di una funzione con due argomenti.
int MultiplyNumbers(int value, int secondValue)
{
    return value * secondValue;
}

// Una funzione che non ha bisogno di nessun argomento. Deve essere una davvero tranquilla.
int main(void)
{
    return MultiplyNumbers(2,3);
}
```

Quando chiamiamo una funzione con parametri, come ad esempio `MultiplyNumbers()` nell'esempio precedente, non elenchiamo il tipo accanto a ciascuna di esse. Il compilatore registra i tipi e ci avverte quando facciamo errori.

Parlando dei tipi, ci sono più tipi di dati in C++ non solo gli interi. Ogni tipo richiede una diversa quantità di spazio nella memoria, misurata in byte. E' importante ricordarlo perché il numero di byte che occupa una variabile ha un impatto diretto sulla quantità di informazioni che può contenere. Questa differenza è evidente nella gamma di valori che contiene un char paragonato a uno short. Le dimensioni dei tipi variano da piattaforma a piattaforma, ma ecco un elenco abbastanza buono per Haiku su un processore a 32-bit.

Tipo	Dimensione (bytes)	Gamma	Descrizione
char	1	-128 a 127	Lettere – ogni variabile char contiene solo una lettera
unsigned char	1	0 a 255	Lettere – ogni variabile char contiene solo una lettera
short	2	-32768 a 32767	Numeri interi
unsigned short	2	0 a 65535	Numeri interi
int	4	-2147483648 a 2147483647	Numeri interi
unsigned int	4	0 a 4294967295	Numeri interi
long	4	-2147483648 a 2147483647	Numeri interi
long long	8	-9223372036854775808 a 9223372036854775807	Numeri interi
unsigned long long	8	0 a 18446744073709551615	Numeri interi
float	4	3.4E +/- 38 (7 digits)	Numeri con una parte frazionaria (floating point)
double	8	1.7E +/- 308 (15 digits)	Numeri con una parte frazionaria (floating point)
long double	12		Numeri con una parte frazionaria (floating point)
bool	1	True or false	Vero o falso
wchar_t	2 o 4	Come short o int	Caratteri "ampi", che possono gestire caratteri internazionali. Come per il char ogni variabile contiene una lettera

Utilizzo di printf()

Vi ricordate la stranezza di `printf()` vista un po di tempo fa? Diamoci un'altra occhiata.

```
#include <stdio.h>

int main(void)
{
    int a;
    int b, c;

    a = 1;
    b = 2;
    c = 3;

    printf("a is %d, b is %d, and c is %d.\n",a,b,c);

    return a + b + c;
}
```

printf è una delle poche funzioni che accettano un numero variabile di argomenti, Il suo primo parametro è sempre una stringa. A seconda del numero dei segnaposto nella stringa, però, può o meno avere parametri aggiuntivi che seguono la stringa. Per esempio, nel nostro esempio precedente, la stringa ha tre segnaposto %d, uno per ogni a,b e c. Tre segnaposto, tre parametri "extra". Ecco alcuni degli altri segnaposto possibili per printf che useremo in seguito. Notate che questo non è un elenco esaustivo per printf - ci sono molte altre opzioni, ma queste per ora sono sufficienti.

Segnaposto	Tipo	Esempio di output
%c	Carattere	a
%d,%i	Intero decimale con segno	234
%e,%E	Notazione scientifica usando e/E	1.7e+5, 1.7E+5
%f	Numero in virgola mobile	3.14
%g	Numero in doppia precisione	3.14
%o	Intero in notazione ottale (base 8)	711
%u	Intero senza segno	255
%x, %X	Intero in notazione esadecimale (base 16)	0xff, 0xFF
%%	Segno percentuale	%

Operatori

Gli operatori ci danno la possibilità di lavorare con variabili e numeri senza chiamare funzioni. +, -, e * sono tutti esempi di operatori, ma C++ ne ha molti altri. Ecco gli operatori aritmetici che ci occorrono per ora.

Operatori	Operazione	Descrizione
a+b	addizione	somma b ad a
a-b	sottrazione	sottrae b ad a
a*b	moltiplicazione	moltiplica a per b
a/b	divisione	divide a per b

Operatori	Operazione	Descrizione
a%b	modulo	il resto di a diviso b
a=b	assegnazione	assegna ad a il valore di b
++a	pre-incremento	aggiunge 1 ad a prima che il resto dell'espressione sia valutato
a++	post-incremento	aggiunge 1 ad a dopo che il resto dell'espressione è stato valutato
--a	pre-decremento	sottrae 1 da a prima che il resto dell'espressione sia valutato
a--	post-decremento	sottrae 1 da a dopo che il resto dell'espressione è stato valutato
a += b	assegnazione con somma	forma abbreviata per a = a + b
a -= b	assegnazione con sottrazione	forma abbreviata per a = a - b
a *= b	assegnazione con moltiplicazione	forma abbreviata per a = a * b
a /= b	assegnazione con divisione	forma abbreviata per a = a / b
a %= b	assegnazione con resto	forma abbreviata per a = a % b

Gli operatori -- e ++ hanno bisogno di un'ulteriore spiegazione di quella possibile nella tabella. Diamo un'occhiata al codice per spiegarlo meglio.

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

```
int main(void)
{
    int a = 1;
    int b = 2;

    // I risultato qui è 3 perché aggiungiamo 1 ad a
    // Dopo viene calcolato a + b
    printf("a++ + b = %d\n",a++ + b);

    // Poiché abbiamo aggiunto 1 ad a, questo stampa un 4
    printf("a + b = %d\n",a + b);

    // Questo è 5 perché il compilatore aggiunge 1 ad a prima di effettuare il calcolo
    printf("++a + b = %d\n",++a + b);

    return 0;
}
```

Wow! Abbiamo visto un sacco di cose in questa lezione, ma utilizzandole tutte possiamo fare ogni sorta di cosa. Mettiamolo in pratica.

```

#include <stdio.h>

// math.h ci dà accesso a un sacco di funzioni matematiche. Lo stiamo
// includendo qui così possiamo accedere a sqrt(), che calcola le
// radici quadrate.
#include <math.h>

double hypotenuse(int a, int b)
{
    return sqrt((a*a) + (b*b));
}

int main(void)
{
    int a = 3;
    int b = 4;

    printf("For the triangle with legs %d and %d, the hypotenuse will be %g\n",
           a,b,hypotenuse(a,b));

    return 0;
}

```

hypotenuse() restituisce un double perchè vogliamo una precisione al di là dei numeri interi. Questo è anche il tipo che ritorna sqrt().

Caccia al bug

Caccia #1

Codice

```

int sum(int first, int second, int third)
{
    return first + second + third;
}

int main(void)
{
    int a = 3;
    int b = 4;
    printf("The sum is %d\n", sum(a,b,c));
    return 0;
}

```

Errori

foo.cpp: In function `int main()':
foo.cpp:14: error: `c' was not declared in this scope

Caccia #2

Codice

```
#include <stdio.h>

double distance(int x1, int y1, int x2, int y2)
{
    int deltax = x2 - x1;
    int deltay = y2 - y1;

    return sqrt((deltax * deltax) + (deltay * deltay));
}

int main(void)
{
    int x1,y1,x2,y2;

    x1 = 3;
    y1 = 3;

    x2 = 8;
    y2 = 3;

    printf("The distance between (%d,%d) and (%d,%d) is %g\n", x1,y1, x2,y2,
           distance(x1,y1,x2,y2));

    return 0;
}
```

Errori

foo.cpp: In function `double distance(int, int, int, int)':
foo.cpp:8: error: `sqrt' was not declared in this scope

Progetto

Utilizzando l'equazione $\text{Interesse} = \text{Capitale iniziale} * \text{tasso di interesse} * \text{unità di tempo}$, calcolate e stampate l'interesse semplice sostenuto per un capitale iniziale di 20.000 \$ ad un tasso del 5% al mese per 24 mesi. Utilizzate una funzione per fare i calcoli dell'interesse attuale.