Basics

(c) 2011 Haim Michael. All Rights Reserved.

Comments

```
#include <iostream>

int main(int argc, char *argv[])
{
    /*
    simple program that prints out hello world
    and asks the user to enter 1 for exit.
    */
    int num = 0;
    std::cout << "hello world";
    while(num!=1)
    {
        //asking the user to enter 1 for exit
        std::cout << "\n\nenter 1 to exit\n";
        std::cin >> num;
    }
    return 0;
}
```

C Style Comments

C++ Style Comments

(c) 2011 Haim Michael. All Rights Reserved.

Program Building

- Building a C++ program includes three stages. The first involves with the preprocessor that recognizes metainformation within the code. The second involves with the compiler that compiles the source code into machine readable object files. The thrid involves with linking the objects into a single application.

(c) 2011 Haim Michael. All Rights Reserved.

Preprocessor Directives

- The preprocessor directives start with the # character. They instruct the preprocessor to perform various operations before the compilation starts.
- The include directive is one of the most commonly in use preprocessor directive.

(c) 2011 Haim Michael. All Rights Reserved.

The `include` Directive

- One of the common preprocessor directives is the include directive. It tells the preprocessor to take everything that exists in a specific header file and embed it within the current file.
- We will usually use the header files for declaring functions that will be defined in another separated file.
- One of the common header files we will usually include in our code is the iostream file. It declares the input and th eoutput mechanisms in C++.

(c) 2011 Haim Michael. All Rights Reserved.

The `include` Directive

- In C, the names of the included files usually end with .h (e.g. <stdio.h>).
- In C++ the suffix is omitted (e.g. <iostream>).

(c) 2011 Haim Michael. All Rights Reserved.

The `main` Function

- This function is the entry point of the application. Its returned value is of the type `int`.
- Its returned value indicates about the program status. It is the exit status of the process. When the returned value is 0 it means that the process has terminated successfully.

(c) 2011 Haim Michael. All Rights Reserved.

The `main` Function

```
#include <iostream>

int main(int argc, char *argv[])
{
    std::cout << "hello world";
    return 0;
}
```

(c) 2011 Haim Michael. All Rights Reserved.

I/O Streams

- This `printf()` function belongs to C. When coding in C++ we use `std:cout` instead.
- We can use the `<<` operator for passing over multiple data of varying types to be sent down the output stream sequentially on a single line of code.

(c) 2011 Haim Michael. All Rights Reserved.

I/O Streams

- The `std:endl` represents the end of line character. When the output stream encounters `std:endl` it will output everything that has been sent down the stream and move forward to the next line. Using `std:endl` is the same as using `'\n'`.
- We can use `std::cin` for getting input from the user.

(c) 2011 Haim Michael. All Rights Reserved.

I/O Streams

```
#include <iostream>

int main(int argc, char *argv[])
{
    int a = 0, b = 0, c, num = 0;
    std::cout << "simple calculator"
        << std::endl << "-----------------" << std::endl;
    std::cout << "a=";
    std::cin >> a;
    std::cout << "b=";
    std::cin >> b;
    c = a + b;
    std::cout << "sum=" << c << std::endl;
    while(num!=1)
    {
        //asking the user to enter 1 for exit
        std::cout << "\n\nenter 1 to exit\n";
        std::cin >> num;
    }
    return 0;
}
```

(c) 2011 Haim Michael. All Rights Reserved.

Namespaces

- We can use namespaces for solving the problem of naming conflicts between different pieces of code.
- We can specify the namespace both in `*.h` and in `*.cpp` files.

(c) 2011 Haim Michael. All Rights Reserved.

Namespaces

person.h

```
#include <iostream>

namespace records
{
    class Person
    {
        public:
            Person();
            void info();  //print out information about the person
            void setFirstName(std::string fName);
            std::string getFirstName();
            void setLastName(std::string lName);
            std::string getLastName();
            void setId(int idVal);
            int getId();
            void details();
        private:
            std::string firstName;
            std::string lastName;
            int id;
    };
}
```

(c) 2011 Haim Michael. All Rights Reserved.

Namespaces

```
#include "person.h"
#include <iostream>

using namespace std;

namespace records
{
    Person::Person()
    {
        firstName = "";
        lastName = "";
        id = 0;
    }
    void Person::setFirstName(string fName)
    {
        firstName = fName;
    }
    void Person::setLastName(string lName)
    {
        lastName = lName;
    }
```

(c) 2011 Haim Michael. All Rights Reserved.

Namespaces

```
    void Person::setId(int idVal)
    {
        id = idVal;
    }
    string Person::getLastName()
    {
        return lastName;
    }
    string Person::getFirstName()
    {
        return firstName;
    }
    int Person::getId()
    {
        return id;
    }
    void Person::details()
    {
        cout << firstName << " " << lastName << " " << id << std::endl;
    }
}
```

(c) 2011 Haim Michael. All Rights Reserved.

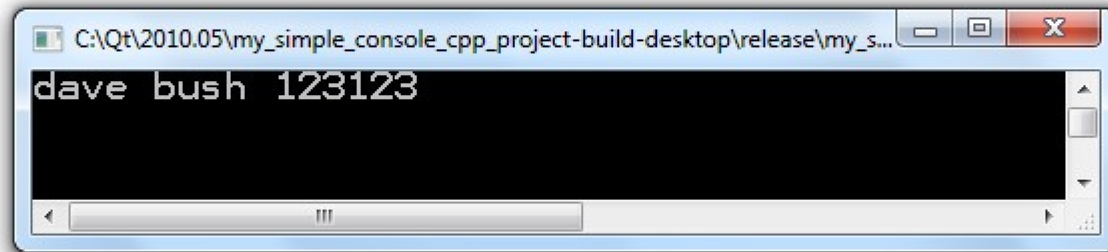
Namespaces

main.cpp

```
#include <iostream>
#include "person.h"

using namespace records;

int main(int argc, char *argv[])
{
    int num = 0;
    Person *ob = new Person();
    ob->setFirstName("dave");
    ob->setLastName("bush");
    ob->setId(123123);
    ob->details();
    std::cin >> num;
}
```

(c) 2011 Haim Michael. All Rights Reserved.

Namespaces



(c) 2011 Haim Michael. All Rights Reserved.

Namespaces

- Any code that falls within a namespace block can call other code within the same namespace without explicitly prepending the namespace.

(c) 2011 Haim Michael. All Rights Reserved.

The `using` Directive

- We can use the `using` directive in order to avoid prepending the namespaces.

```
#include <iostream>
#include "person.h"

using namespace records;

int main(int argc, char *argv[])
{
    int num = 0;
    Person *ob = new Person();  //avoiding records::Person
    ob->setFirstName("dave");
    ob->setLastName("bush");
    ob->setId(123123);
    ob->details();
    std::cin >> num;
}
```

(c) 2011 Haim Michael. All Rights Reserved.

The `using` Directive

- Each single source file can contain multiple `using` directives. Nevertheless, doing so might place us back to the same names conflicts problem we try to avoid.

(c) 2011 Haim Michael. All Rights Reserved.

The `using` Directive

- The using directive can also be used to refer a particular item within the namespace.

```
#include <iostream>
#include "person.h"

using std::cout;

int main(int argc, char *argv[])
{
    int num = 0;
    cout << "hello world";
    std::cin >> num;
}
```

(c) 2011 Haim Michael. All Rights Reserved.

Variables

- We can declare our variables anywhere in our code and use them within the current block below the line where they were declared.
- We can declare a variable without assigning any value to it. We can alternatively assign them with an initial value when we declare them. If we don't initialize a variable it will hold garbage.

```
int numA;
int numB = 123;
```

(c) 2011 Haim Michael. All Rights Reserved.

Variables

- The most common types include the following: `int`, `short`, `long`, `unsigned int`, `unsigned short`, `unsigned long`, `float`, `double`, `char` and `bool`.
- The C++ programming language doesn't provide a basic string type. Nevertheless, a standard implementation of string is provided as part of the standard library.

(c) 2011 Haim Michael. All Rights Reserved.

Casting

- We can conver (cast) the type of a given value into another type. There are three possible syntax options for casting.

```
int num = 4;
bool bobo = (bool) num;
bool bobo = bool (num);
bool bobo = static_cast<bool>(num);
```

(c) 2011 Haim Michael. All Rights Reserved.

Casting

- In some cases an automatic casting takes place. We should be aware of the risk for loosing data.

```
#include <iostream>
#include "person.h"

int main(int argc, char *argv[])
{
    int num = 0;
    double number = 2.5;
    num = number;
    std::cout << "num=" << num << std::endl;
    std::cin >> num;
    return 0;
}
```

(c) 2011 Haim Michael. All Rights Reserved.

Operators

- The most commonly in use operators include the following:

=, !, +, -, *, /, %, ++, --, +=, -=, *=, /=, %=, &, &=, |, <<, >>,

<<=, >>=, ^, ^=,

(c) 2011 Haim Michael. All Rights Reserved.

Enumerated Types

- Enumerated types allow us to define our own sequences.

  Behind the scenes, an enumerated type value is just an integer.

```
#include <iostream>
#include "person.h"
#include "stdio.h"

int main(int argc, char *argv[])
{
    typedef enum {summer,autumn,winter,spring} Season;
    Season temp = winter;
    std::cout << temp << std::endl;
    temp = summer;
    std::cout << temp << std::endl;
    getchar();
}
```

(c) 2011 Haim Michael. All Rights Reserved.

Structs

- Using structs we can encapsulate one or more existing types into a new type, such as a new type that describes a database record.
- Once we define a new struct type, each variable of that new struct type will include all fields we specified in our declaration for that new struct type.

(c) 2011 Haim Michael. All Rights Reserved.

Structs

```
#include <iostream>

namespace infosys
{
    typedef struct
    {
        std::string title;
        std::string author;
        int pages;
    }
    Book;
}
```

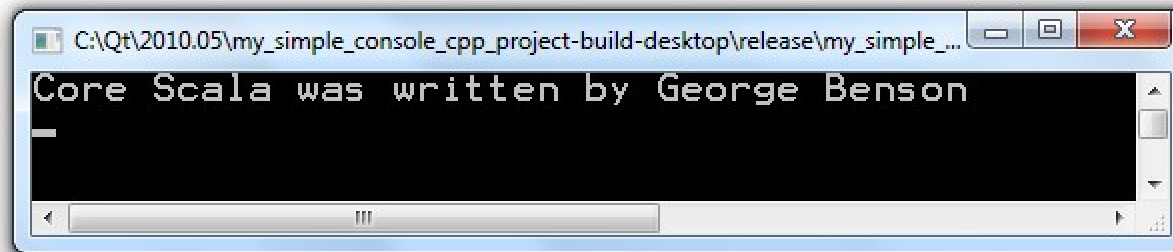
(c) 2011 Haim Michael. All Rights Reserved.

Structs

```
#include <iostream>
#include "book.h"
#include "stdio.h"

using namespace infosys;

int main(int argc, char *argv[])
{
    Book coreScala;
    coreScala.author = "George Benson";
    coreScala.pages = 254;
    coreScala.title = "Core Scala";
    std::cout << coreScala.title << " was written by "
       << coreScala.author << std::endl;
    getchar();
}
```

(c) 2011 Haim Michael. All Rights Reserved.

Structs



(c) 2011 Haim Michael. All Rights Reserved.

The if..else Statement

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

int main(int argc, char *argv[])
{
    int num = 7.2;
    if(num>0)
    {
        std::cout << "positive" << std::endl;
    }
    else
    {
        if(num<0)
        {
            std::cout << "negative" << std::endl;
        }
        else
        {
            std::cout << "zero" << std::endl;
        }
    }
    getchar();
    return 0;
}
```

(c) 2011 Haim Michael. All Rights Reserved.

The Trenary Operator

```
(condition)?expression1:expression2
```

The Condition We Want To Check

The Value of The Whole Expression if Condition is True

The Value of The Whole Expression if Condition is False

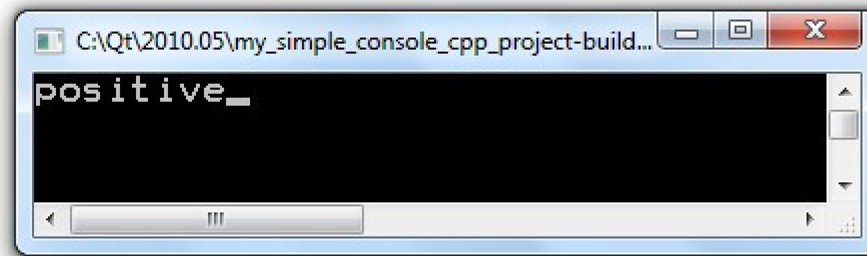
(c) 2011 Haim Michael. All Rights Reserved.

The Trenary Operator

```
#include <iostream>
#include "book.h"
#include "stdio.h"

int main(int argc, char *argv[])
{
    int num = 7.2;
    std::string msg = (num>0)?"positive":(num<0)?"negative":"zero";
    std::cout << msg;
    getchar();
    return 0;
}
```

(c) 2011 Haim Michael. All Rights Reserved.

The Trenary Operator



The Output

(c) 2011 Haim Michael. All Rights Reserved.

The Switch Case Statement

```
#include <iostream>
#include "book.h"
#include "stdio.h"

int main(int argc, char *argv[])
{
    int num = 2;
    switch(num)
    {
        case 1:
            std::cout << "one" << std::endl;
            break;
        case 2:
            std::cout << "two" << std::endl;
            break;
```

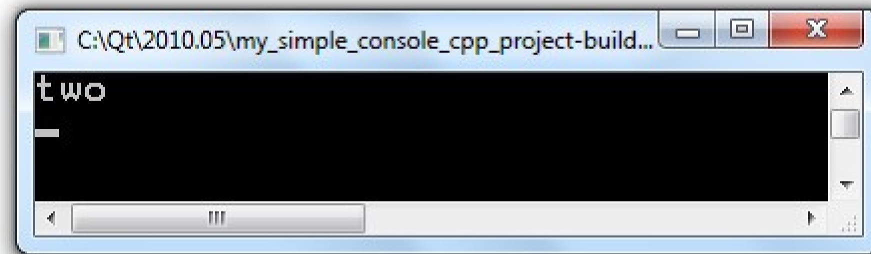
(c) 2011 Haim Michael. All Rights Reserved.

The Switch Case Statement

```
        case 3:
            std::cout << "three" << std::endl;
            break;
        default:
            std::cout << "other" << std::endl;
            break;
    }
    getchar();
    return 0;
}
```

(c) 2011 Haim Michael. All Rights Reserved.

The Switch Case Statement



The Output

(c) 2011 Haim Michael. All Rights Reserved.

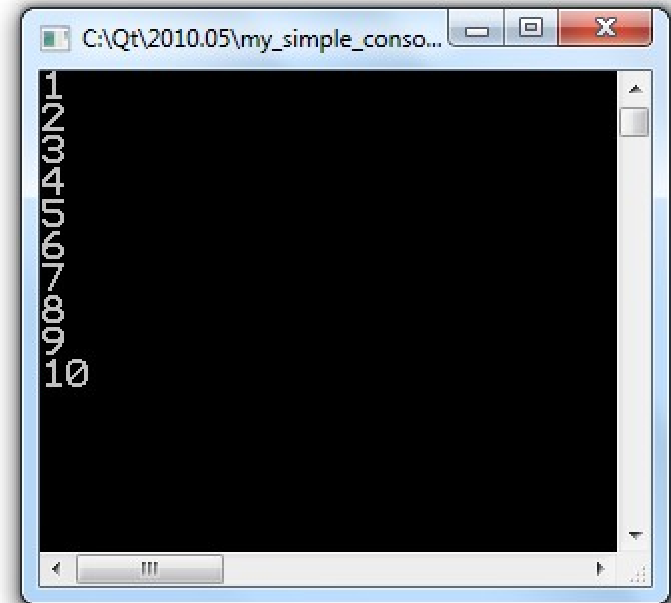
Conditional Operators

- The conditional operators purpose is to compare two expressions. They result in `true` or `false`.
- The C++ programming language includes the following conditional operators: `<`, `<=`, `>`, `>=`, `==`, `<`, `!=`, `!`, `&`, `|`, `&&` and `||`.
- The `&&` and the `||` operators are short circuit ones. When using these operators the evaluation stops when the final result is certain.

(c) 2011 Haim Michael. All Rights Reserved.

The while Loop

```
#include <iostream>
#include "book.h"
#include "stdio.h"

int main(int argc, char *argv[])
{
    int num=1;
    while(num<=10)
    {
        std::cout<<num<<std::endl;
        num++;
    }
    getchar();
    return 0;
}
```



(c) 2011 Haim Michael. All Rights Reserved.

The do..while Loop

```
#include <iostream>
#include "book.h"
#include "stdio.h"

int main(int argc, char *argv[])
{
    int num=1;
    do
    {
        std::cout<<num<<std::endl;
        num++;
    }
    while(num<=10);
    getchar();
    return 0;
}
```

(c) 2011 Haim Michael. All Rights Reserved.

The for Loop

```
#include <iostream>
#include "book.h"
#include "stdio.h"

int main(int argc, char *argv[])
{
    for(int i=1; i<=10; i++)
    {
        std::cout << i << std::endl;
    }
    getchar();
    return 0;
}
```

(c) 2011 Haim Michael. All Rights Reserved.

Arrays

- An array holds a series of values. The values should be of the same type. We can access each one of the values by its position in the array.

```
int vec[10];
vec[0] = 12;
vec[1] = vec[0] + 3;
```

(c) 2011 Haim Michael. All Rights Reserved.

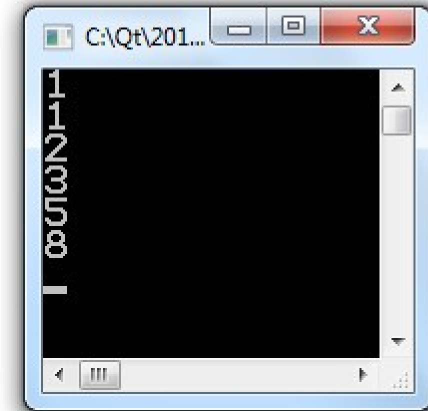
Arrays

- When declaring an array we must specify its size. Once the size is specified we cannot change it.
- We cannot specify the size to be the value of a variable. It must be a constant value.
- The first element of the array is always at position 0. The last element of the array is always at position equals to the size of the array minus 1.

(c) 2011 Haim Michael. All Rights Reserved.

Arrays

```
#include <iostream>
#include "book.h"
#include "stdio.h"

int main(int argc, char *argv[])
{
    int vec[6];
    vec[0] = 1;
    vec[1] = 1;
    vec[2] = vec[1] + vec[0];
    vec[3] = vec[2] + vec[1];
    vec[4] = vec[3] + vec[2];
    vec[5] = vec[4] + vec[3];
    for(int i=0; i<6; i++)
    {
        std::cout << vec[i] << std::endl;
    }
    getchar();
    return 0;
}
```



(c) 2011 Haim Michael. All Rights Reserved.

Functions

- We usually define functions in order to make the program more understandable. Decomposing the code into concise functions eases our way in developing our program and in its maintenance.
- When the function is been used within the same file where it was defined there is no need in a header file.
- When the function is been used within other files as well we should declare it within a header file and place its definition in a separated source file.

(c) 2011 Haim Michael. All Rights Reserved.

Functions

- We usually define functions in order to make the program more understandable. Decomposing the code into concise functions eases our way in developing our program and in its maintenance.
- When the function is been used within the same file where it was defined there is no need in a header file.
- When the function is been used within other files as well we should declare it within a header file and place its definition in a separated source file.

(c) 2011 Haim Michael. All Rights Reserved.

Functions

```
double sum(double a, double b);
double multiply(double a, double b);
double divide(double a, double b);
double difference(double a, double b);
```

utils.h

(c) 2011 Haim Michael. All Rights Reserved.

Functions

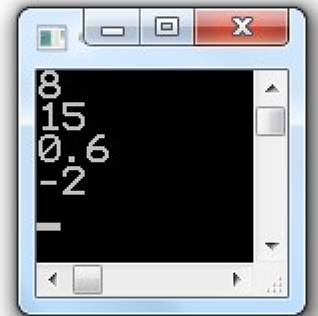
utils.cpp

```
double sum(double a, double b)
{
    return a+b;
}

double multiply(double a, double b)
{
    return a*b;
}

double divide(double a, double b)
{
    return a/b;
}

double difference(double a, double b)
{
    return a-b;
}
```

(c) 2011 Haim Michael. All Rights Reserved.

Functions

```
#include <iostream>
#include "stdio.h"
#include "utils.h"

int main(int argc, char *argv[])
{
    std::cout << sum(3,5) << std::endl;
    std::cout << multiply(3,5) << std::endl;
    std::cout << divide(3,5) << std::endl;
    std::cout << difference(3,5) << std::endl;
    getchar();
    return 0;
}
```



(c) 2011 Haim Michael. All Rights Reserved.

The Heap and The Stack

- The memory our application uses is composed of two parts. The heap and the stack.
- The stack works as a collection of cards. Each time there is a call for executing a function a new card that holds the memory required for the new execution is added on top of the stack.
- The heap is the area of memory we use when dynamically allocating new memory.

(c) 2011 Haim Michael. All Rights Reserved.

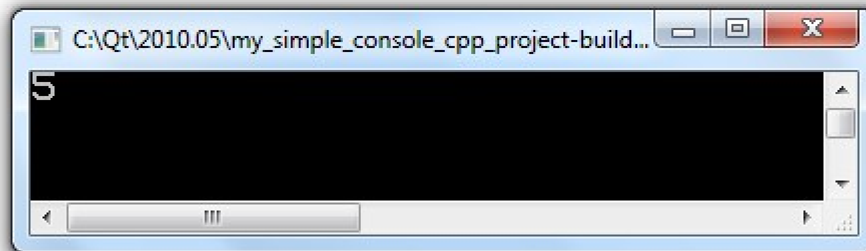
Pointers

- Pointer is a data type that its value (memory address) refers directly to another value stored in another place in the memory we work with.
- We can use the new operator for allocating a new memory area in the heap. The returned value is the address of that area.

(c) 2011 Haim Michael. All Rights Reserved.

Pointers

```
#include <iostream>
#include "stdio.h"
#include "utils.h"

int main(int argc, char *argv[])
{
    int* num = new int;
    *num = 5;
    int* other = num;
    std::cout << *other << std::endl;
    getchar();
}
```



(c) 2011 Haim Michael. All Rights Reserved.

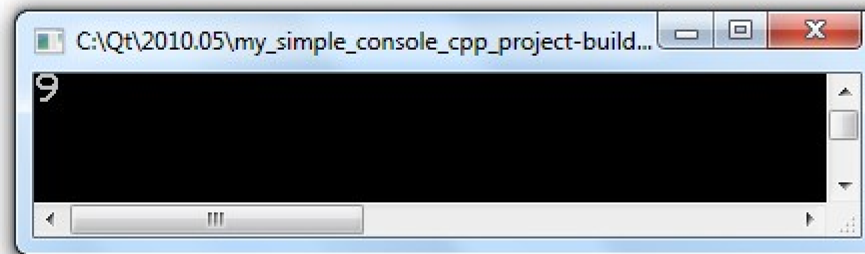
Pointers

- Placing the & operator together with a variable we shall get its address.
- Placing the * operator together with a variable that holds an address we shall get the value at that address.

(c) 2011 Haim Michael. All Rights Reserved.

Pointers

```
#include <iostream>
#include "stdio.h"
#include "utils.h"

int main(int argc, char *argv[])
{
    int a = 8;
    int* temp = &a;
    *temp = 9;
    std::cout << a;
    getchar();
}
```



(c) 2011 Haim Michael. All Rights Reserved.

Pointers

- The -> (arrow) operator allows us an easy usage of pointers.

```
#include <iostream>
#include "stdio.h"
#include "rectangle.h"

int main(int argc, char *argv[])
{
    Rectangle *ob = new Rectangle();
    ob->setWidth(4);
    ob->setHeight(3);
    std::cout << ob->area() << std::endl;
    getchar();
    return 0;
}
```

(c) 2011 Haim Michael. All Rights Reserved.

Pointers

```
class Rectangle
{
public:
    Rectangle();
    void setWidth(double w);
    void setHeight(double h);
    double area();
private:
    double width;
    double height;
};
```

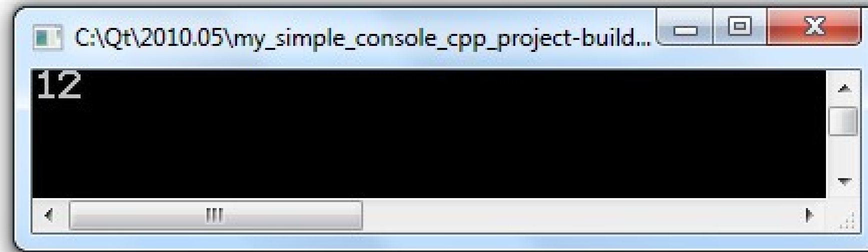
(c) 2011 Haim Michael. All Rights Reserved.

Pointers

```
#include "rectangle.h"

Rectangle::Rectangle()
{
}

void Rectangle::setWidth(double w)
{
    if(w>0) width=w;
}

void Rectangle::setHeight(double h)
{
    if(h>0) height=h;
}

double Rectangle::area()
{
    return width*height;
}
```

(c) 2011 Haim Michael. All Rights Reserved.

Pointers



The Output

(c) 2011 Haim Michael. All Rights Reserved.

The Heap and The Stack

- The way the stack works the compiler must be able to determine the exact required memory size during the compilation.
- For that reason, it is impossible to create an array that its size is unknown during the compilation time.

```
...
int size = 10;
int vec[size];    //doesn't compile!
...
```

(c) 2011 Haim Michael. All Rights Reserved.

The Heap and The Stack

- Although some compilers do support that, the C++ spec still wasn't updated and most compilers don't support it.
- The solution involves with allocating the array dynamically.

```
...
int* vec;
vec = new int[size];
...
```

- Once the memory is allocated we can use the array just as any other array.

(c) 2011 Haim Michael. All Rights Reserved.

Strings

- We can represent a string the same way it was done in C. Each string as an array of characters.

```
...
char arr[10] = "hello friends";
char* arr = "hello friends";
...
```

- We can alternatively use the `std::string` type. This type wraps the array of characters.

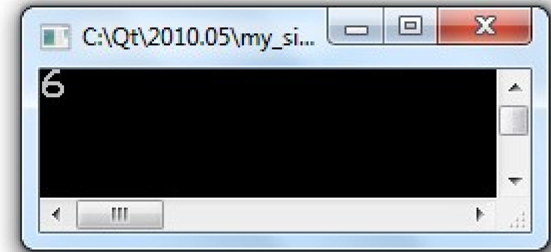
```
...
std::string str = "hello friends";
...
```

(c) 2011 Haim Michael. All Rights Reserved.

By Reference

- We can use the `&` operator for specifying parameters we want to get their values by reference.

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

void doSomething(int &num)
{
    num++;
}

int main(int argc, char *argv[])
{
    int temp = 5;
    doSomething(temp);
    std::cout << temp;
    getchar();
    return 0;
}
```



(c) 2011 Haim Michael. All Rights Reserved.

Exceptions Handling

- An exception is an unexpected situation. When an exceptional situation is detected an exception is thrown. Another piece of code can catch that exception and respond.
- When the `throw` command is executed the funciton immediately terminates and nothing is returned.
- If the code where the exception was thrown is surrounded with `try` and `catch` block then the catch segment will get the exception and handle it.

(c) 2011 Haim Michael. All Rights Reserved.

Exceptions Handling

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

int divide(int a, int b)
{
    if (b==0) throw std::exception();
    return a/b;
}
```

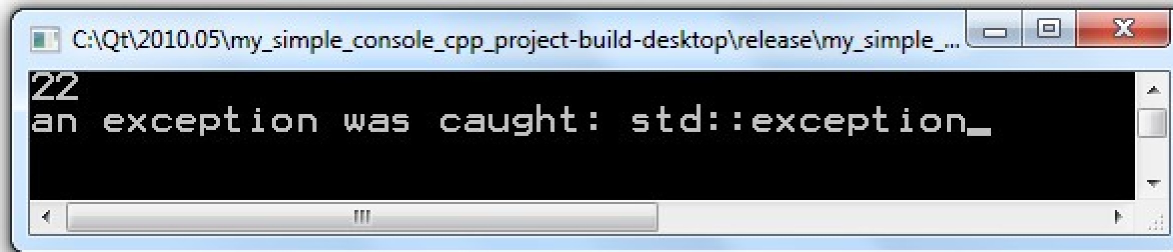
(c) 2011 Haim Michael. All Rights Reserved.

Exceptions Handling

```
int main(int argc, char *argv[])
{
    try
    {
        int temp = 0;
        temp = divide(44,2);
        std::cout << temp << std::endl;
        temp = divide(5,0);
        std::cout << temp << std::endl;
    }
    catch(std::exception e)
    {
        std::cout << "an exception was caught: " << e.what();
    }
    getchar();
    return 0;
}
```

(c) 2011 Haim Michael. All Rights Reserved.

Exceptions Handling



The Output

(c) 2011 Haim Michael. All Rights Reserved.

Constants

- Defining a variable together with the `const` keyword ensures that the assigned value cannot be changed.

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

int main(int argc, char *argv[])
{
	const int num = 5;
	//num = 9;
	getchar();
	return 0;
}
```

(c) 2011 Haim Michael. All Rights Reserved.

Constants

- Defining a parameter together with the `const` keyword ensures that the passed over value won't chang.

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

int doSomething(const int* temp)
{
        *temp = 99;
        int total = 2 * *temp;
        return total;
}

int main(int argc, char *argv[])
{
        int num = 5;
        ...
}
```

(c) 2011 Haim Michael. All Rights Reserved.

Object Oriented Programming

- C++ is an object oriented programming language. We can define classes and instantiate them.
- The declaration of the class should be within a separated header file.
- The definition should be within a separated source code cpp file that its name is identical to the name of the header file.

(c) 2011 Haim Michael. All Rights Reserved.

Object Oriented Programming

person.h

```
#include <iostream>

namespace records
{
    class Person
    {
        public:
            Person();
            void info();  //print out information about the person
            void setFirstName(std::string fName);
            std::string getFirstName();
            void setLastName(std::string lName);
            std::string getLastName();
            void setId(int idVal);
            int getId();
            void details();
        private:
            std::string firstName;
            std::string lastName;
            int id;
    };
}
```

(c) 2011 Haim Michael. All Rights Reserved.

Object Oriented Programming

```
#include "person.h"
#include <iostream>

using namespace std;

namespace records
{
    Person::Person()
    {
        firstName = "";
        lastName = "";
        id = 0;
    }
    void Person::setFirstName(string fName)
    {
        firstName = fName;
    }
    void Person::setLastName(string lName)
    {
        lastName = lName;
    }
```

(c) 2011 Haim Michael. All Rights Reserved.

Object Oriented Programming

```
    void Person::setId(int idVal)
    {
        id = idVal;
    }
    string Person::getLastName()
    {
        return lastName;
    }
    string Person::getFirstName()
    {
        return firstName;
    }
    int Person::getId()
    {
        return id;
    }
    void Person::details()
    {
        cout << firstName << " " << lastName << " " << id << std::endl;
    }
}
```

(c) 2011 Haim Michael. All Rights Reserved.

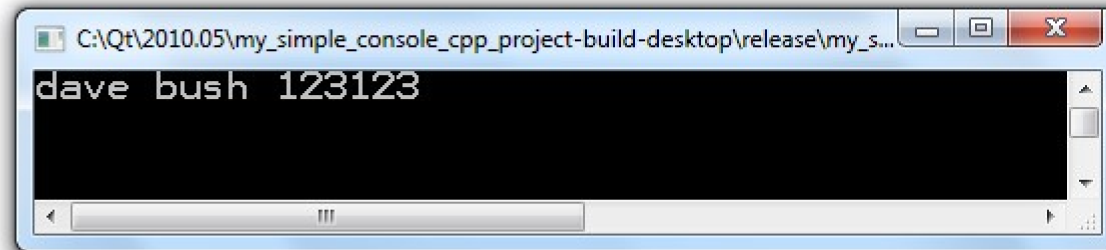
Object Oriented Programming

```
#include <iostream>
#include "person.h"

using namespace records;

int main(int argc, char *argv[])
{
    int num = 0;
    Person *ob = new Person();
    ob->setFirstName("dave");
    ob->setLastName("bush");
    ob->setId(123123);
    ob->details();
    std::cin >> num;
}
```

main.cpp

(c) 2011 Haim Michael. All Rights Reserved.

Object Oriented Programming



(c) 2011 Haim Michael. All Rights Reserved.

Basics

(c) 2011 Haim Michael. All Rights Reserved.

(c) 2011 Haim Michael. All Rights Reserved.

Comments

```
#include <iostream>

int main(int argc, char *argv[])
{
    /*
    simple program that prints out hello world
    and asks the user to enter 1 for exit.
    */
    int num = 0;
    std::cout << "hello world";
    while(num!=1)
    {
        //asking the user to enter 1 for exit
        std::cout << "\n\nenter 1 to exit\n";
        std::cin >> num;
    }
    return 0;
}
```

C Style Comments

C++ Style Comments

(c) 2011 Haim Michael. All Rights Reserved. 2

Program Building

- Building a C++ program includes three stages. The first involves with the preprocessor that recognizes metainformation within the code. The second involves with the compiler that compiles the source code into machine readable object files. The thrid involves with linking the objects into a single application.

(c) 2011 Haim Michael. All Rights Reserved. 3

Preprocessor Directives

- The preprocessor directives start with the # character. They instruct the preprocessor to perform various operations before the compilation starts.
- The include directive is one of the most commonly in use preprocessor directive.

(c) 2011 Haim Michael. All Rights Reserved.

(c) 2011 Haim Michael. All Rights Reserved.

The `include` Directive

- One of the common preprocessor directives is the include directive. It tells the preprocessor to take everything that exists in a specific header file and embed it within the current file.
- We will usually use the header files for declaring functions that will be defined in another separated file.
- One of the common header files we will usually include in our code is the iostream file. It declares the input and th eoutput mechanisms in C++.

(c) 2011 Haim Michael. All Rights Reserved.

(c) 2011 Haim Michael. All Rights Reserved.

The `include` Directive

- In C, the names of the included files usually end with .h (e.g. <stdio.h>).
- In C++ the suffix is omitted (e.g. <iostream>).

(c) 2011 Haim Michael. All Rights Reserved. 6

The `main` Function

- This function is the entry point of the application. Its returned value is of the type `int`.
- Its returned value indicates about the program status. It is the exit status of the process. When the returned value is 0 it means that the process has terminated successfully.

(c) 2011 Haim Michael. All Rights Reserved.

(c) 2011 Haim Michael. All Rights Reserved.

The `main` Function

```
#include <iostream>

int main(int argc, char *argv[])
{
    std::cout << "hello world";
    return 0;
}
```

(c) 2011 Haim Michael. All Rights Reserved.

(c) 2011 Haim Michael. All Rights Reserved.

I/O Streams

- This `printf()` function belongs to C. When coding in C++ we use `std:cout` instead.
- We can use the << operator for passing over multiple data of varying types to be sent down the output stream sequentially on a single line of code.

(c) 2011 Haim Michael. All Rights Reserved.

(c) 2011 Haim Michael. All Rights Reserved.

I/O Streams

- The `std:endl` represents the end of line character. When the output stream encounters `std:endl` it will output everything that has been sent down the stream and move forward to the next line. Using `std:endl` is the same as using `'\n'`.
- We can use `std::cin` for getting input from the user.

(c) 2011 Haim Michael. All Rights Reserved.

(c) 2011 Haim Michael. All Rights Reserved.

I/O Streams

```
#include <iostream>

int main(int argc, char *argv[])
{
    int a = 0, b = 0, c, num = 0;
    std::cout << "simple calculator"
        << std::endl << "------------------" << std::endl;
    std::cout << "a=";
    std::cin >> a;
    std::cout << "b=";
    std::cin >> b;
    c = a + b;
    std::cout << "sum=" << c << std::endl;
    while(num!=1)
    {
        //asking the user to enter 1 for exit
        std::cout << "\n\nenter 1 to exit\n";
        std::cin >> num;
    }
    return 0;
}
```

(c) 2011 Haim Michael. All Rights Reserved. 11

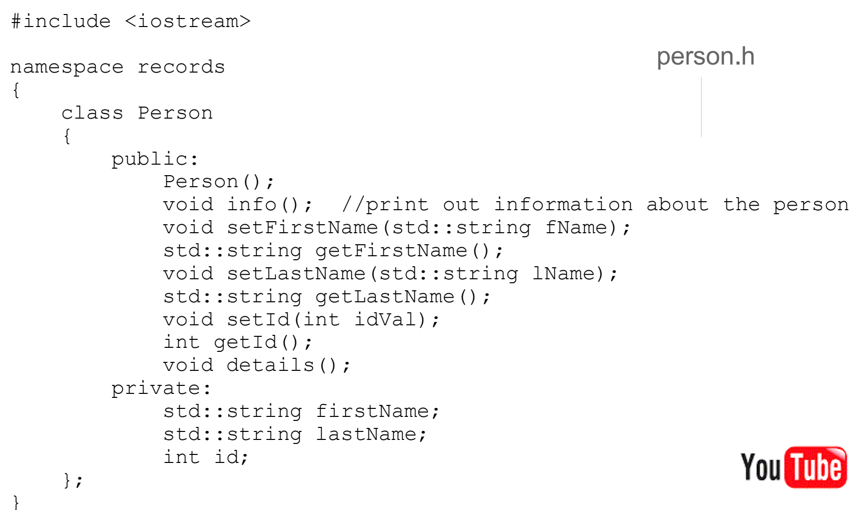
Namespaces

- We can use namespaces for solving the problem of naming conflicts between different pieces of code.
- We can specify the namespace both in `*.h` and in `*.cpp` files.

(c) 2011 Haim Michael. All Rights Reserved.

(c) 2011 Haim Michael. All Rights Reserved.

Namespaces



```
#include <iostream>

namespace records
{
    class Person
    {
        public:
            Person();
            void info();  //print out information about the person
            void setFirstName(std::string fName);
            std::string getFirstName();
            void setLastName(std::string lName);
            std::string getLastName();
            void setId(int idVal);
            int getId();
            void details();
        private:
            std::string firstName;
            std::string lastName;
            int id;
    };
}
```


(c) 2011 Haim Michael. All Rights Reserved. 13

(c) 2011 Haim Michael. All Rights Reserved.

Namespaces

person.cpp

```
#include "person.h"
#include <iostream>

using namespace std;

namespace records
{
    Person::Person()
    {
        firstName = "";
        lastName = "";
        id = 0;
    }
    void Person::setFirstName(string fName)
    {
        firstName = fName;
    }
    void Person::setLastName(string lName)
    {
        lastName = lName;
    }
```

(c) 2011 Haim Michael. All Rights Reserved. 14

Namespaces

```
    void Person::setId(int idVal)
    {
        id = idVal;
    }
    string Person::getLastName()
    {
        return lastName;
    }
    string Person::getFirstName()
    {
        return firstName;
    }
    int Person::getId()
    {
        return id;
    }
    void Person::details()
    {
        cout << firstName << " " << lastName << " " << id << std::endl;
    }
}
```

(c) 2011 Haim Michael. All Rights Reserved.

(c) 2011 Haim Michael. All Rights Reserved.

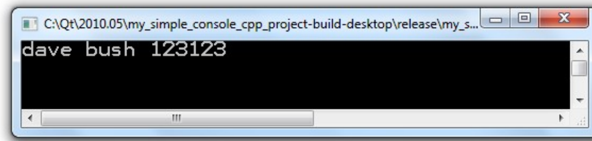
Namespaces

main.cpp

```
#include <iostream>
#include "person.h"

using namespace records;

int main(int argc, char *argv[])
{
    int num = 0;
    Person *ob = new Person();
    ob->setFirstName("dave");
    ob->setLastName("bush");
    ob->setId(123123);
    ob->details();
    std::cin >> num;
}
```

(c) 2011 Haim Michael. All Rights Reserved. 16

Namespaces



(c) 2011 Haim Michael. All Rights Reserved.

(c) 2011 Haim Michael. All Rights Reserved.

Namespaces

- Any code that falls within a namespace block can call other code within the same namespace without explicitly prepending the namespace.

The `using` Directive

- We can use the `using` directive in order to avoid prepending the namespaces.

```
#include <iostream>
#include "person.h"

using namespace records;

int main(int argc, char *argv[])
{
    int num = 0;
    Person *ob = new Person();  //avoiding records::Person
    ob->setFirstName("dave");
    ob->setLastName("bush");
    ob->setId(123123);
    ob->details();
    std::cin >> num;
}
```

(c) 2011 Haim Michael. All Rights Reserved.

The `using` Directive

- Each single source file can contain multiple `using` directives. Nevertheless, doing so might place us back to the same names conflicts problem we try to avoid.

(c) 2011 Haim Michael. All Rights Reserved.

(c) 2011 Haim Michael. All Rights Reserved.

The `using` Directive

- The using directive can also be used to refer a particular item within the namespace.

```
#include <iostream>
#include "person.h"

using std::cout;

int main(int argc, char *argv[])
{
    int num = 0;
    cout << "hello world";
    std::cin >> num;
}
```

(c) 2011 Haim Michael. All Rights Reserved.

(c) 2011 Haim Michael. All Rights Reserved.

Variables

- We can declare our variables anywhere in our code and use them within the current block below the line where they were declared.
- We can declare a variable without assigning any value to it. We can alternatively assign them with an initial value when we declare them. If we don't initialize a variable it will hold garbage.

```
int numA;
int numB = 123;
```

(c) 2011 Haim Michael. All Rights Reserved.

(c) 2011 Haim Michael. All Rights Reserved.

Variables

- The most common types include the following: `int`, `short`, `long`, `unsigned int`, `unsigned short`, `unsigned long`, `float`, `double`, `char` and `bool`.
- The C++ programming language doesn't provide a basic string type. Nevertheless, a standard implementation of string is provided as part of the standard library.

(c) 2011 Haim Michael. All Rights Reserved. 23

Casting

- We can conver (cast) the type of a given value into another type. There are three possible syntax options for casting.

```
int num = 4;
bool bobo = (bool) num;
bool bobo = bool (num);
bool bobo = static_cast<bool>(num);
```

(c) 2011 Haim Michael. All Rights Reserved.

(c) 2011 Haim Michael. All Rights Reserved.

Casting

- In some cases an automatic casting takes place. We should be aware of the risk for loosing data.

```
#include <iostream>
#include "person.h"

int main(int argc, char *argv[])
{
    int num = 0;
    double number = 2.5;
    num = number;
    std::cout << "num=" << num << std::endl;
    std::cin >> num;
    return 0;
}
```

Operators

- The most commonly in use operators include the following:

  =, !, +, -, *, /, %, ++, --, +=, -=, *=, /=, %=, &, &=, |, <<, >>, <<=, >>=, ^, ^=,

(c) 2011 Haim Michael. All Rights Reserved.

(c) 2011 Haim Michael. All Rights Reserved.

Enumerated Types

- Enumerated types allow us to define our own sequences.

  Behind the scenes, an enumerated type value is just an integer.

```
#include <iostream>
#include "person.h"
#include "stdio.h"

int main(int argc, char *argv[])
{
    typedef enum {summer,autumn,winter,spring} Season;
    Season temp = winter;
    std::cout << temp << std::endl;
    temp = summer;
    std::cout << temp << std::endl;
    getchar();
}
```

(c) 2011 Haim Michael. All Rights Reserved.

Structs

- Using structs we can encapsulate one or more existing types into a new type, such as a new type that describes a database record.
- Once we define a new struct type, each variable of that new struct type will include all fields we specified in our declaration for that new struct type.

(c) 2011 Haim Michael. All Rights Reserved.

(c) 2011 Haim Michael. All Rights Reserved.

Structs

```
#include <iostream>

namespace infosys
{
    typedef struct
    {
        std::string title;
        std::string author;
        int pages;
    }
    Book;
}
```

(c) 2011 Haim Michael. All Rights Reserved.

(c) 2011 Haim Michael. All Rights Reserved.

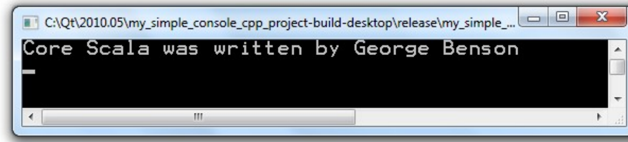
Structs

```
#include <iostream>
#include "book.h"
#include "stdio.h"

using namespace infosys;

int main(int argc, char *argv[])
{
    Book coreScala;
    coreScala.author = "George Benson";
    coreScala.pages = 254;
    coreScala.title = "Core Scala";
    std::cout << coreScala.title << " was written by "
        << coreScala.author << std::endl;
    getchar();
}
```

(c) 2011 Haim Michael. All Rights Reserved.

(c) 2011 Haim Michael. All Rights Reserved.

Structs



(c) 2011 Haim Michael. All Rights Reserved.

(c) 2011 Haim Michael. All Rights Reserved.

The if..else Statement

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

int main(int argc, char *argv[])
{
    int num = 7.2;
    if(num>0)
    {
        std::cout << "positive" << std::endl;
    }
    else
    {
        if(num<0)
        {
            std::cout << "negative" << std::endl;
        }
        else
        {
            std::cout << "zero" << std::endl;
        }
    }
    getchar();
    return 0;
}
```

(c) 2011 Haim Michael. All Rights Reserved.

The Trenary Operator

```
(condition)?expression1:expression2
```

The Condition We Want To Check

The Value of The Whole Expression if Condition is True

The Value of The Whole Expression if Condition is False

(c) 2011 Haim Michael. All Rights Reserved. 33

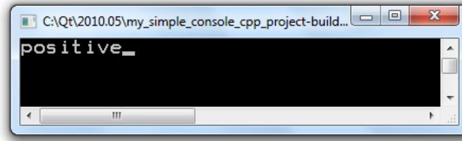
The Trenary Operator

```
#include <iostream>
#include "book.h"
#include "stdio.h"

int main(int argc, char *argv[])
{
    int num = 7.2;
    std::string msg = (num>0)?"positive":(num<0)?"negative":"zero";
    std::cout << msg;
    getchar();
    return 0;
}
```

(c) 2011 Haim Michael. All Rights Reserved.

(c) 2011 Haim Michael. All Rights Reserved.

The Trenary Operator



The Output

(c) 2011 Haim Michael. All Rights Reserved.

(c) 2011 Haim Michael. All Rights Reserved.

The Switch Case Statement

```
#include <iostream>
#include "book.h"
#include "stdio.h"

int main(int argc, char *argv[])
{
    int num = 2;
    switch(num)
    {
        case 1:
            std::cout << "one" << std::endl;
            break;
        case 2:
            std::cout << "two" << std::endl;
            break;
```

(c) 2011 Haim Michael. All Rights Reserved.

(c) 2011 Haim Michael. All Rights Reserved.

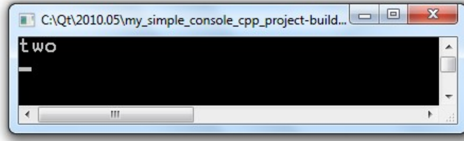
The Switch Case Statement

```
        case 3:
            std::cout << "three" << std::endl;
            break;
        default:
            std::cout << "other" << std::endl;
            break;
    }
    getchar();
    return 0;
}
```

(c) 2011 Haim Michael. All Rights Reserved.

(c) 2011 Haim Michael. All Rights Reserved.

The Switch Case Statement



The Output

(c) 2011 Haim Michael. All Rights Reserved.

(c) 2011 Haim Michael. All Rights Reserved.

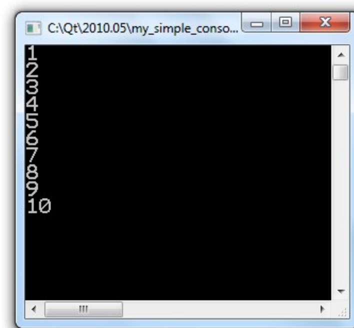
Conditional Operators

- The conditional operators purpose is to compare two expressions. They result in `true` or `false`.
- The C++ programming language includes the following conditional operators: <, <=, >, >=, ==, <, !=, !, &, |, && and ||.
- The && and the || operators are short circuit ones. When using these operators the evaluation stops when the final result is certain.

(c) 2011 Haim Michael. All Rights Reserved. 39

The while Loop

```
#include <iostream>
#include "book.h"
#include "stdio.h"

int main(int argc, char *argv[])
{
    int num=1;
    while(num<=10)
    {
        std::cout<<num<<std::endl;
        num++;
    }
    getchar();
    return 0;
}
```



(c) 2011 Haim Michael. All Rights Reserved.

(c) 2011 Haim Michael. All Rights Reserved.

The do..while Loop

```
#include <iostream>
#include "book.h"
#include "stdio.h"

int main(int argc, char *argv[])
{
    int num=1;
    do
    {
        std::cout<<num<<std::endl;
        num++;
    }
    while(num<=10);
    getchar();
    return 0;
}
```

(c) 2011 Haim Michael. All Rights Reserved.

(c) 2011 Haim Michael. All Rights Reserved.

The for Loop

```
#include <iostream>
#include "book.h"
#include "stdio.h"

int main(int argc, char *argv[])
{
    for(int i=1; i<=10; i++)
    {
        std::cout << i << std::endl;
    }
    getchar();
    return 0;
}
```

(c) 2011 Haim Michael. All Rights Reserved.

(c) 2011 Haim Michael. All Rights Reserved.

Arrays

- An array holds a series of values. The values should be of the same type. We can access each one of the values by its position in the array.

```
int vec[10];
vec[0] = 12;
vec[1] = vec[0] + 3;
```

(c) 2011 Haim Michael. All Rights Reserved.

(c) 2011 Haim Michael. All Rights Reserved.

Arrays

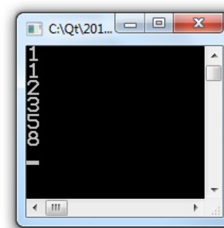
- When declaring an array we must specify its size. Once the size is specified we cannot change it.
- We cannot specify the size to be the value of a variable. It must be a constant value.
- The first element of the array is always at position 0. The last element of the array is always at position equals to the size of the array minus 1.

(c) 2011 Haim Michael. All Rights Reserved.

(c) 2011 Haim Michael. All Rights Reserved.

Arrays

```
#include <iostream>
#include "book.h"
#include "stdio.h"

int main(int argc, char *argv[])
{
    int vec[6];
    vec[0] = 1;
    vec[1] = 1;
    vec[2] = vec[1] + vec[0];
    vec[3] = vec[2] + vec[1];
    vec[4] = vec[3] + vec[2];
    vec[5] = vec[4] + vec[3];
    for(int i=0; i<6; i++)
    {
        std::cout << vec[i] << std::endl;
    }
    getchar();
    return 0;
}
```



(c) 2011 Haim Michael. All Rights Reserved.

(c) 2011 Haim Michael. All Rights Reserved.

Functions

- We usually define functions in order to make the program more understandable. Decomposing the code into concise functions eases our way in developing our program and in its maintenance.
- When the function is been used within the same file where it was defined there is no need in a header file.
- When the function is been used within other files as well we should declare it within a header file and place its definition in a separated source file.

(c) 2011 Haim Michael. All Rights Reserved.

(c) 2011 Haim Michael. All Rights Reserved.

Functions

- We usually define functions in order to make the program more understandable. Decomposing the code into concise functions eases our way in developing our program and in its maintenance.
- When the function is been used within the same file where it was defined there is no need in a header file.
- When the function is been used within other files as well we should declare it within a header file and place its definition in a separated source file.

(c) 2011 Haim Michael. All Rights Reserved.

(c) 2011 Haim Michael. All Rights Reserved.

Functions

```
double sum(double a, double b);
double multiply(double a, double b);
double divide(double a, double b);
double difference(double a, double b);
```

utils.h

(c) 2011 Haim Michael. All Rights Reserved.

(c) 2011 Haim Michael. All Rights Reserved.

Functions

utils.cpp

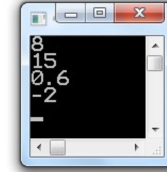
```
double sum(double a, double b)
{
    return a+b;
}

double multiply(double a, double b)
{
    return a*b;
}

double divide(double a, double b)
{
    return a/b;
}

double difference(double a, double b)
{
    return a-b;
}
```

(c) 2011 Haim Michael. All Rights Reserved.

(c) 2011 Haim Michael. All Rights Reserved.

Functions

```
#include <iostream>
#include "stdio.h"
#include "utils.h"

int main(int argc, char *argv[])
{
    std::cout << sum(3,5) << std::endl;
    std::cout << multiply(3,5) << std::endl;
    std::cout << divide(3,5) << std::endl;
    std::cout << difference(3,5) << std::endl;
    getchar();
    return 0;
}
```



(c) 2011 Haim Michael. All Rights Reserved.

(c) 2011 Haim Michael. All Rights Reserved.

The Heap and The Stack

- The memory our application uses is composed of two parts. The heap and the stack.
- The stack works as a collection of cards. Each time there is a call for executing a function a new card that holds the memory required for the new execution is added on top of the stack.
- The heap is the area of memory we use when dynamically allocating new memory.

(c) 2011 Haim Michael. All Rights Reserved.

Pointers

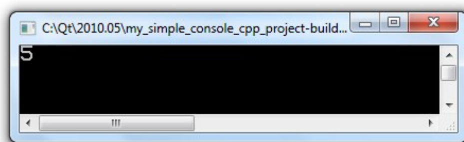
- Pointer is a data type that its value (memory address) refers directly to another value stored in another place in the memory we work with.
- We can use the new operator for allocating a new memory area in the heap. The returned value is the address of that area.

(c) 2011 Haim Michael. All Rights Reserved.

(c) 2011 Haim Michael. All Rights Reserved.

Pointers

```
#include <iostream>
#include "stdio.h"
#include "utils.h"

int main(int argc, char *argv[])
{
    int* num = new int;
    *num = 5;
    int* other = num;
    std::cout << *other << std::endl;
    getchar();
}
```



(c) 2011 Haim Michael. All Rights Reserved.

(c) 2011 Haim Michael. All Rights Reserved.

Pointers

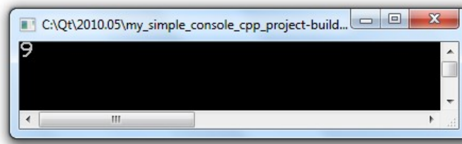
- Placing the & operator together with a variable we shall get its address.
- Placing the * operator together with a variable that holds an address we shall get the value at that address.

(c) 2011 Haim Michael. All Rights Reserved.

(c) 2011 Haim Michael. All Rights Reserved.

Pointers

```
#include <iostream>
#include "stdio.h"
#include "utils.h"

int main(int argc, char *argv[])
{
    int a = 8;
    int* temp = &a;
    *temp = 9;
    std::cout << a;
    getchar();
}
```



(c) 2011 Haim Michael. All Rights Reserved. 55

Pointers

- The -> (arrow) operator allows us an easy usage of pointers.

```
#include <iostream>
#include "stdio.h"
#include "rectangle.h"

int main(int argc, char *argv[])
{
    Rectangle *ob = new Rectangle();
    ob->setWidth(4);
    ob->setHeight(3);
    std::cout << ob->area() << std::endl;
    getchar();
    return 0;
}
```

(c) 2011 Haim Michael. All Rights Reserved.

(c) 2011 Haim Michael. All Rights Reserved.

Pointers

```
class Rectangle
{
public:
    Rectangle();
    void setWidth(double w);
    void setHeight(double h);
    double area();
private:
    double width;
    double height;
};
```

(c) 2011 Haim Michael. All Rights Reserved.

(c) 2011 Haim Michael. All Rights Reserved.

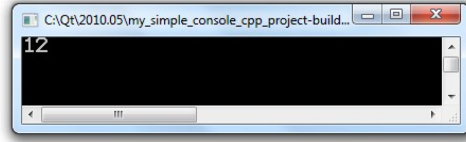
Pointers

```
#include "rectangle.h"

Rectangle::Rectangle()
{
}

void Rectangle::setWidth(double w)
{
    if(w>0) width=w;
}

void Rectangle::setHeight(double h)
{
    if(h>0) height=h;
}

double Rectangle::area()
{
    return width*height;
}
```

(c) 2011 Haim Michael. All Rights Reserved.

(c) 2011 Haim Michael. All Rights Reserved.

Pointers



The Output

(c) 2011 Haim Michael. All Rights Reserved.

(c) 2011 Haim Michael. All Rights Reserved.

The Heap and The Stack

- The way the stack works the compiler must be able to determine the exact required memory size during the compilation.
- For that reason, it is impossible to create an array that its size is unknown during the compilation time.

```
...
int size = 10;
int vec[size];    //doesn't compile!
...
```

(c) 2011 Haim Michael. All Rights Reserved. 60

The Heap and The Stack

- Although some compilers do support that, the C++ spec still wasn't updated and most compilers don't support it.
- The solution involves with allocating the array dynamically.

```
...
int* vec;
vec = new int[size];
...
```

- Once the memory is allocated we can use the array just as any other array.

(c) 2011 Haim Michael. All Rights Reserved.

(c) 2011 Haim Michael. All Rights Reserved.

Strings

- We can represent a string the same way it was done in C. Each string as an array of characters.

```
...
char arr[10] = "hello friends";
char* arr = "hello friends";
...
```

- We can alternatively use the `std::string` type. This type wraps the array of characters.

```
...
std::string str = "hello friends";
...
```

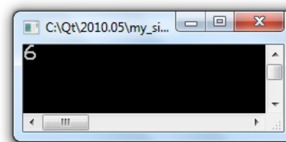
(c) 2011 Haim Michael. All Rights Reserved.

(c) 2011 Haim Michael. All Rights Reserved.

By Reference

- We can use the & operator for specifying parameters we want to get their values by reference.

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

void doSomething(int &num)
{
    num++;
}

int main(int argc, char *argv[])
{
    int temp = 5;
    doSomething(temp);
    std::cout << temp;
    getchar();
    return 0;
}
```



(c) 2011 Haim Michael. All Rights Reserved.

(c) 2011 Haim Michael. All Rights Reserved.

Exceptions Handling

- An exception is an unexpected situation. When an exceptional situation is detected an exception is thrown. Another piece of code can catch that exception and respond.
- When the `throw` command is executed the funciton immediately terminates and nothing is returned.
- If the code where the exception was thrown is surrounded with `try` and `catch` block then the catch segment will get the exception and handle it.

(c) 2011 Haim Michael. All Rights Reserved.

(c) 2011 Haim Michael. All Rights Reserved.

Exceptions Handling

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

int divide(int a, int b)
{
    if (b==0) throw std::exception();
    return a/b;
}
```

(c) 2011 Haim Michael. All Rights Reserved.

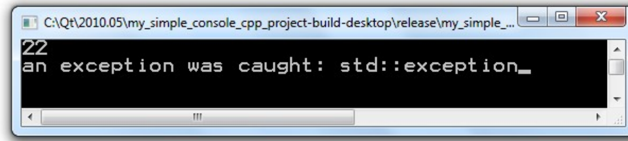
(c) 2011 Haim Michael. All Rights Reserved.

Exceptions Handling

```
int main(int argc, char *argv[])
{
    try
    {
        int temp = 0;
        temp = divide(44,2);
        std::cout << temp << std::endl;
        temp = divide(5,0);
        std::cout << temp << std::endl;
    }
    catch(std::exception e)
    {
        std::cout << "an exception was caught: " << e.what();
    }
    getchar();
    return 0;
}
```

(c) 2011 Haim Michael. All Rights Reserved.

Exceptions Handling



The Output

(c) 2011 Haim Michael. All Rights Reserved.

(c) 2011 Haim Michael. All Rights Reserved. 67

Constants

- Defining a variable together with the `const` keyword ensures that the assigned value cannot be changed.

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

int main(int argc, char *argv[])
{
	const int num = 5;
	//num = 9;
	getchar();
	return 0;
}
```

(c) 2011 Haim Michael. All Rights Reserved.

(c) 2011 Haim Michael. All Rights Reserved.

Constants

- Defining a parameter together with the `const` keyword ensures that the passed over value won't chang.

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

int doSomething(const int* temp)
{
        *temp = 99;
        int total = 2 * *temp;
        return total;
}

int main(int argc, char *argv[])
{
        int num = 5;
        ...
}
```

(c) 2011 Haim Michael. All Rights Reserved.

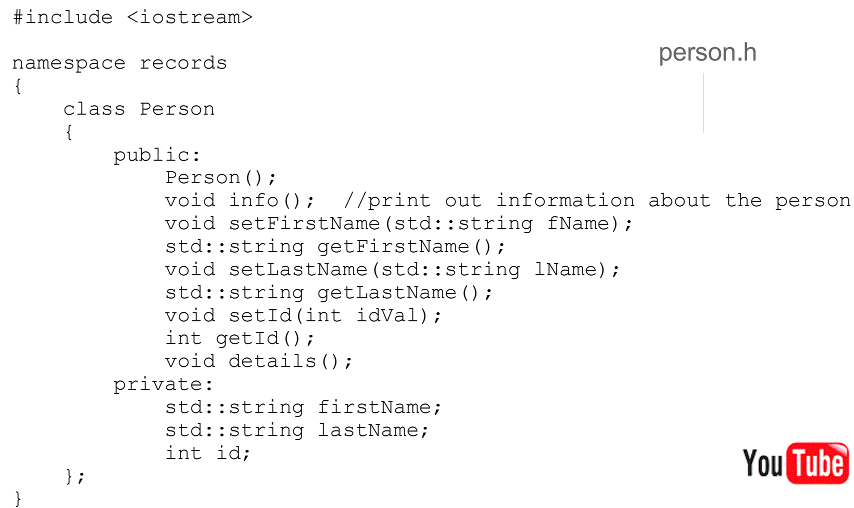
Object Oriented Programming

- C++ is an object oriented programming language. We can define classes and instantiate them.
- The declaration of the class should be within a separated header file.
- The definition should be within a separated source code cpp file that its name is identical to the name of the header file.

(c) 2011 Haim Michael. All Rights Reserved.

(c) 2011 Haim Michael. All Rights Reserved.

Object Oriented Programming



```
#include <iostream>

namespace records
{
    class Person
    {
        public:
            Person();
            void info();  //print out information about the person
            void setFirstName(std::string fName);
            std::string getFirstName();
            void setLastName(std::string lName);
            std::string getLastName();
            void setId(int idVal);
            int getId();
            void details();
        private:
            std::string firstName;
            std::string lastName;
            int id;
    };
}
```


(c) 2011 Haim Michael. All Rights Reserved.

(c) 2011 Haim Michael. All Rights Reserved.

Object Oriented Programming

person.cpp

```
#include "person.h"
#include <iostream>

using namespace std;

namespace records
{
    Person::Person()
    {
        firstName = "";
        lastName = "";
        id = 0;
    }
    void Person::setFirstName(string fName)
    {
        firstName = fName;
    }
    void Person::setLastName(string lName)
    {
        lastName = lName;
    }
```

(c) 2011 Haim Michael. All Rights Reserved.

Object Oriented Programming

```
    void Person::setId(int idVal)
    {
        id = idVal;
    }
    string Person::getLastName()
    {
        return lastName;
    }
    string Person::getFirstName()
    {
        return firstName;
    }
    int Person::getId()
    {
        return id;
    }
    void Person::details()
    {
        cout << firstName << " " << lastName << " " << id << std::endl;
    }
}
```

(c) 2011 Haim Michael. All Rights Reserved.

(c) 2011 Haim Michael. All Rights Reserved. 73

Object Oriented Programming

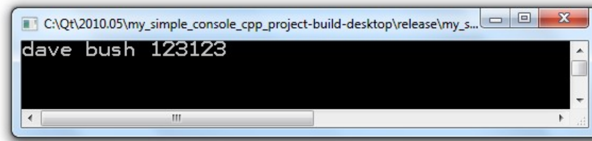
main.cpp

```
#include <iostream>
#include "person.h"

using namespace records;

int main(int argc, char *argv[])
{
    int num = 0;
    Person *ob = new Person();
    ob->setFirstName("dave");
    ob->setLastName("bush");
    ob->setId(123123);
    ob->details();
    std::cin >> num;
}
```

(c) 2011 Haim Michael. All Rights Reserved.

(c) 2011 Haim Michael. All Rights Reserved.

Object Oriented Programming



(c) 2011 Haim Michael. All Rights Reserved.

(c) 2011 Haim Michael. All Rights Reserved.