iOS

Criando Aplicações para iPhone e iPad

A Linguagem C / Objective C

SoftBlue
www.softblue.com.br

Introdução ao Objective C

- Também conhecido como ObjC
- Baseada na linguagem C
- Criada por Brad Cox e Tom Love em 1980
- Licenciada em 1988 pela NeXT
- Principal linguagem de desenvolvimento para o Mac
- Linguagem dinâmica: maior velocidade
- Permite criação de classes em tempo real
- Utilização do Xcode neste módulo do curso

Principais tipos de dados (C)

- char

```
char meuChar;                   // char de uma única posição apenas declarado
char meuChar = "texto";         // char de uma única posição já inicializado
char meuChar[10];               // char de 10 posições apenas declarado
char meuChar[10] = "textotexto"; // char de 10 posições já inicializado
char meuChar[] = "texto";       // char cujo número de posições será calculado pelo
                                // compilador, no caso, 6 (5 letras + final de string)
```

- int

```
int meuInt;                     // variável do tipo int apenas declarada
int meuInt = 50;                // variável do tipo int já inicializada
```

- float

```
float meuFloat;                 // variável do tipo float apenas declarada
float meuFloat = 50.25;         // variável do tipo float já inicializada
```

- double

```
double meuDouble;               // variável do tipo double apenas declarada
double meuDouble = 50.25252;    // variável do tipo double já inicializada
```

- bool

```
bool meuBool;                   // variável do tipo bool apenas declarada
bool meuBool = TRUE;            // variável do tipo bool já inicializada
```

Arrays (C)

- Utilização de colchetes e o número de posições

```
int meuArrayInt[10];                // Cria um array de inteiros de 10 posições

meuArrayInt[0] = 54;
meuArrayInt[1] = 55;
meuArrayInt[2] = 56;
meuArrayInt[3] = 57;
meuArrayInt[4] = 58;

int meuInt = 2 * meuArrayInt[2];  // Armazena o valor 56 * 2 (112) em meuInt
```

0	1	2	3	4	5	6	7	8	9
54	55	56	57	58					

Constantes

- Pré compilação (#define)
  - A constante é substituída no código no momento de compilação

```
#define MINHA_CONSTANTE 1024
#define PI 3.14
```

- Tempo de execução (const)
  - A constante é criada na memória em tempo de execução

```
const int MINHA_CONSTANTE = 1024;
const double PI = 3.14;
```

Operadores e comandos

- Operadores matemáticos
- Operadores condicionais
- Operadores lógicos
- Comandos de controle
- Comandos de repetição

Operadores matemáticos

Para os exemplos apresentados, assuma:

```
double x;          // Variável "x" declarada como do tipo de dado double
```

= Atribui o valor da sua direita à variável a sua esquerda

```
x = 3;             // Resultado: x armazena o valor 3
```

+ Soma dois valores numéricos

```
x = 3 + 5;         // Resultado: x armazena o valor 8
```

- Subtrai dois valores numéricos

```
x = 3 - 5;         // Resultado: x armazena o valor -2
```

* Multiplica dois valores numéricos

```
x = 2 * 6;         // Resultado: x armazena o valor 12
```

/ Divide dois valores numéricos

```
x = 18 / 3;        // Resultado: x armazena o valor 6
```

% Obtém o resto da divisão entre dois valores numéricos

```
x = 19 % 3;        // Resultado: x armazena o valor 1
```

Operadores matemáticos

++ Incrementa em 1 o valor da variável acoplada

```
x = 3;
x++;               // Resultado: x neste momento armazena o valor 4
```

-- Decrementa em 1 o valor da variável acoplada

```
x = 3;
x--;               // Resultado: x neste momento armazena o valor 2
```

- É possível utilizar estes operadores em expressões e outros comandos

```
x = 3;
y = 2 + $x++; // Resultado: y = 5 e x = 4
```

- Diferença entre ++x e x++

```
x = 3;
y = 2 + ++x;  // Resultado: y = 6 e x = 4
```

Operadores matemáticos

Assumir x = 5 para o início de cada exemplo apresentado neste slide

+= Soma à variável à sua esquerda o valor apresentado em sua direita

```
x += 3;        // Resultado: x armazena o valor 8
```

-= Subtrai da variável à sua esquerda o valor apresentado em sua direita

```
x -= 3;        // Resultado: x armazena o valor 2
```

*= Multiplica a variável à sua esquerda pelo valor apresentado em sua direita

```
x *= 3;        // Resultado: x armazena o valor 15
```

/= Divide à variável à sua esquerda o valor apresentado em sua direita

```
x /= 3;        // Resultado: x armazena o valor 1.6666666666667
```

%= Atribui à variável da esquerda o resto de sua divisão pelo valor apresentado em sua direita

```
x %= 3;        // Resultado: x armazena o valor 2
```

Operadores condicionais

- Utilizados para avaliar uma condição

Operador	Significado
==	Compara se dois valores tem o mesmo valor, mesmo em tipos distintos
===	Compara se dois valores são idênticos
!=	Compara se dois valores são diferentes
!==	Compara se dois valores são diferentes inclusive no tipo de dado
!	Inverte o resultado (negação)
<	Compara se o valor da direita é menor que o da esquerda
>	Compara se o valor da direita é maior que o da esquerda
<=	Compara se o valor da esquerda é menor ou igual que o da direita
>=	Compara se o valor da esquerda é maior ou igual que o da direita

Operadores lógicos

- Utilizados para montar uma condição

Operador	Significado
!	Nega a condição informada (inverte seu resultado lógico)
&	Retorna verdadeiro se ambas as condições da esquerda e direita forem satisfeitas
|	Retorna verdadeiro se pelo menos uma das condições (da direita ou da esquerda) for satisfeita
&&	O mesmo que &, porém é otimizado: Se a condição do lado esquerdo for falsa, não verifica o lado direito da expressão. Resulta a operação toda em FALSA neste caso.
||	O mesmo que |, porém é otimizado: Se a condição do lado esquerdo for verdadeira, não verifica o lado direito da expressão. Resulta a operação toda em VERDADEIRA neste caso.

Comandos de controle e repetição

- if(condição) { } [else if(condição) { }] [else { }]
  - Permite executar um bloco de código se determinada condição for verdadeira

```
if(3 > 5) {
      // Não entra no primeiro if
}

if(1 < 10) {
      // Entra no segundo if
}
else {
      // Não entra no else do segundo if
}
```

```
int i = 5;
if(i == 3)
{
      // O valor de i é 3
}
else if(i == 4)
{
      // O valor de i é 4
}
else {
      // O valor de i não é 3 nem 4.
}
```

Comandos de controle e repetição

- Short if (if reduzido)
  - Forma reduzida de escrever o comando if
  - Deve ser utilizado somente em casos onde o bloco de código for curtíssimo
- condição ? bloco_true : bloco_false;

```
int x = 5;
int y;

if(x % 2 == 0) {
      // Caso x seja par
      y = 60;
}
else {
      // Caso x seja ímpar
      y = 61;
}
```

```
int x = 5;
int y;

y = x % 2 == 0 ? 60 : 61 ;

// Se x for par, atribuirá 60, caso contrário 61
```

Comandos de controle e repetição

- switch(valor) { cases }
  - Permite executar um bloco de código específico dependendo do valor avaliado

```
int i = 1;

switch(i)
{
      case 0:
            // O valor de i é 0
            break;
      case 1:
            // O valor de i é 1
            break;
      case 2:
            // O valor de i é 2
            break;
      default:
            // Nenhum
            break;
}
```

Comandos de controle e repetição

- for(inicialização; condição; incremento)
  - Permite criar um laço de repetição, executando para cada valor o mesmo bloco de código

```
for(int i = 0; i < 10; i++)
{
      // Exibir i
}

// Resultado: 0 1 2 3 4 5 6 7 8 9
```

Comandos de controle e repetição

- while(condição)
  - Permite executar várias vezes um bloco de código enquanto sua condição for verdadeira

```
int i = 0;

while(i < 5)
{
      // Exibir i
      i++;
}

// Resultado: 0 1 2 3 4
```

Comandos de controle e repetição

- do { } while (condição);
  - Similar ao while, com a diferença de que o do while tem o bloco de código executado obrigatoriamente uma vez antes de avaliar a condição

```
int i = 0;

do
{
      // Exibir i
      i++;
}
while(i < 5);

// Resultado: 0 1 2 3 4
```

Comandos de controle e repetição

- break
  - Permite interromper suspender o comando de repetição e ir para o próximo comando

```
for(int i = 0; i < 10; i++)
{
      if(i == 4)
      {
            break;
      }

      // Exibir i
}

// Resultado: 0 1 2 3
```

Comandos de controle e repetição

- continue
  - Permite instruir ao comando de repetição que interrompa esta iteração e avançe para a próxima iteração da repetição

```
for(int i = 0; i < 10; i++)
{
    if(i == 4)
    {
        continue;
    }

    // Exibir i
}

// Resultado: 0 1 2 3 5 6 7 8 9
```

Criando funções

- Permite centralizar blocos de código

Sintaxe

```
tipoDeRetorno  nomeDaFunção([parâmetros]) { }
```

Exemplo

```
int calcularDobro(int num) {
    int dobro = num * 2;
    return dobro;
}

int i = 5;
int iDobro = calcularDobro(i);
// Exibir iDobro

// Resultado:  10
```

Aulas práticas e manuais on-line

Assista agora às aulas práticas, que abordam os temas tratados nesta aula teórica, para consolidar seus conhecimentos.

Clique aqui para visualizar as aulas práticas disponíveis