Pesquisa I

Leonardo Aguilar Murça
Algoritmos e Estrutura de Dados I

17 de março de 2018

Tópico 1: Formas de imprimir texto com acentuação (Bibliotecas e Tabela ASCII)

```
#include <iostream>
#include <locale.h> //Biblioteca para inclusão de caracteres especiais

using namespace std;

int main(){

setlocale(LC_ALL, "Portuguese") //Seta o idioma padrão como Português
cout << "Olá Mundo!"' << endl;
cout << "\205"' << endl; //O número 205 equivale ao 'à' na tabela ASCII

return 0;
}
```

No código acima vimos que para incluir os caracteres especiais em um programa, precisamos incluir a biblioteca `<locale.h>` e setar dentro do `int main()` o idioma para Português. Além disso, podemos representar o código correspondente do caractere desejado da tabela ASCII (em octal) através da barra invertida. Como por exemplo o caractere da letra 'a' craseada(à) como vimos no código acima. Saída: `Olá mundo!  à`

Tópico 2: Formas de entrada/leitura de dados caracteres com espaços em branco

```
#include <iostream>

using namespace std;

int main(){

char nome[256];
cout << "'Digite seu nome completo:"' << endl;
cin.getline(nome, 256); //Lê todos os caracteres do vetor nome
cout << "Seu nome completo é:  ≪<< nome << endl;
```

```
return 0;
}
```

No código acima vimos que para se ler o conteúdo inteiro, inclusive os espaços em branco, de uma variável do tipo `char` ou do tipo `string`, precisamos utilizar a diretiva `getline`. Esse recurso é bastante útil quando precisamos armazenar o nome completo de um usuário em uma única variável. PS: No caso de string, a leitura de dados é diferente: `getline (cin, nome-da-string)`. Entrada: Leonardo Aguilar. Saída: `Leonardo Aguilar`

Tópico 3: Funções `getch() e getche()`

Exemplo 01: `getch()`

```
#include <iostream>
#include <conio.h> //Biblioteca que contém a função getch()

using namespace std;

int main() {

cout << "Digite um caractere qualquer:"' << endl;
getch(); //Finaliza a leitura quando digitado qualquer caractere

return 0;
}
```

A função `getch()` lê o caractere do teclado e não permite que seja impresso na tela. Como getche(), esta função não aceita argumentos e devolve o caractere lido para a função que a chamou.Para que ela funcione, é necessário a inclusão da biblioteca `<conio.h>`. Entrada: 1. Saída:

Exemplo 02: `getche()`

```
#include <iostream>
#include <conio.h> //Biblioteca que contém a função getche()

using namespace std;

int main() {

cout << "Digite um caractere qualquer:  "' << endl;
getche(); // Finaliza a leitura digitado qualquer caractere (Imprime na tela)

return 0;
```

```
}
```

A função `getche()` lê o caractere do teclado e permite que seja impresso na tela. Esta função não aceita argumentos e devolve o caractere lido para a função que a chamou.Para que ela funcione, é necessário a inclusão da biblioteca `<conio.h>`.Entrada: 1. Saída: `1`.

Tópico 4: Manipuladores de campo de impressão

Exemplo 01: `setw()`

```
#include <iostream>
#include <iomanip> //Biblioteca que contém a função setw()

using namespace std;

int main(){

int caixas = 45;
cout << "Número de caixas:"' << setw(10) << caixas << endl;

return 0;
}
```

Para definir a largura da impressão de uma variável usa-se o `setw(n)`, onde n é o número de caracteres desejado para a impressão desta variável. Note que o `setw()` só afeta a próxima variável do cout. Saída: `45`.

Exemplo 02: `setprecision()`

```
#include <iostream>
#define pi 3.14159265

using namespace std;

int main(){

cout.precision(5);
cout << pi << endl;

return 0;
}
```

A diretiva `setprecision()` é usada para definir o limite de casas decimais a serem impressas de uma varíavel com ponto flutuante. No código acima, utilizamos o exemplo do famoso número irracional π, onde é impresso somente suas 4 primeiras casas decimais. Nesse exemplo, utilizamos a sintaxe `cout.precision(5)` que irá funcionar para todos as impressões

de número com ponto flutuantes no código. Caso queira utilizar apenas para uma determinada variável, utilizamos: `cout << setprecision(5) << beta << endl;` onde irá definir o limite de casas decimais a serem impressas apenas do variável `beta`.

Exemplo 03: `setfill()`

```
#include <iostream>
#include <iomanip> // Biblioteca que contém a função setfill()

using namespace std;

int main() {

cout << setfill('x') << setw(10);
cout << 77 << endl;

return 0;
}
```

A diretiva `setfill()` trabalha em conjunto com a função `setw()`, uma vez que ela seleciona o caractere que deverá preencher as colunas em branco iniciais de um campo. No exemplo acima, as 8 primeiras colunas em branco são preenchidas com o caractere "x"e as últimas duas com 77. Saída: `xxxxxxxx77`.

Exemplo 04: `setiosflags()`

```
#include <iostream>
#include <iomanip> // Biblioteca que contém a função setiosflags()

using namespace std;

int main() {

cout << hex;
cout << setiosflags(ios::showbase); //Mostra a base (em hex) do número 100
cout << 100 << endl;

return 0;
}
```

A função `setiosflags()` seleciona o modo de apresentação de um número (com ponto decimal, notação científica, etc). No exemplo acima, essa função indicará a base do número 100 em hexadecimal ao ser impresso na tela. Saída: `0x64`.

Tópico 05: Impressão de caracteres gráficos

```
#include <iostream>

using namespace std;

int main(){

cout << "\n\n"; cout << "\n \xDC\xDC\xDB\xDB\xDB\xDB\xDC\xDC";
cout << "\n \xDF0\xDF\xDF\DF\xDF0\xDF";

return 0;
}
```

Já vimos como imprimir caracteres ASCII usando o objeto `cout`. Os caracteres gráficos e outros não-padrões requerem outra maneira de escrita para serem impressos. A forma de representar um caractere de código acima de 127 da tabela ASCII é:

```
\xdd
```

onde `dd` representa o código do caractere na base hexadecimal. Como `\xdd` é um caractere, ele pode ser representado entre aspas duplas. No código acima usamos esse caracteres especiais para imprimir na tela o formato de um carro como vimos logo abaixo: