Técnicas de Programação

Procedimentos e Funções

Prof. Angelo Loula
UEFS

Procedimentos e Funções

- Programas cada vez maiores
  - Aumento da complexidade
  - Repetição de trechos de código
- Como atacar um problema de cada vez?
  - "dividir-para-conquistar": dividir problemas em subproblemas menores e mais tratáveis, e se ainda for muito complexa, dividi-se em subproblemas ainda menores e assim sucessivamente.
- Como evitar repetições?
  - Modularização

Procedimentos e Funções

- Modularização
  - Simplificar a codificação
    - Evitar a repetição do mesmo trecho de código
    - Reaproveitar código existente
    - Facilitar modificações
  - Organizar o programas em partes que podem ser compreendidas isoladamente
    - Melhorar a estruturação do programa
    - Facilitar o entendimento do programa

Procedimentos e Funções

- Números primos de 1 a N

```
Solicite o valor de N

Para cada número entre 1 e N

      Teste se o número é primo

      Se for primo, mostre o número
```

Procedimentos e Funções

- Números primos de 1 a N

```
int main() {
  int N, num;
  scanf("%d", &N );
  for(num=1; num<=N; num++){
    if( ehprimo( num )==1 ){
        printf("%d", num );
    }
  }
}
```

- Mas e este comando: **`ehprimo(num)`**?
  - Um sub-programa que precisa ser definido!

Procedimentos e Funções

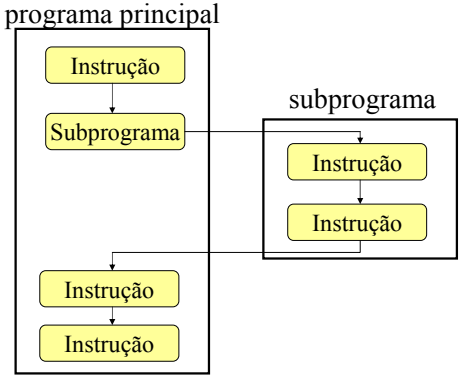
- Subprogramas
  - Um subprograma não pode ser executado diretamente.
  - Ele precisa ser "invocado" ou "chamado" pelo programa principal.
  - Esta chamada desvia o fluxo de controle para o subprograma
  - Um subprograma pode invocar outro subprograma e assim sucessivamente.
  - Podem ser parametrizados, auxiliando reaproveitamento.

Procedimentos e Funções

- **Caso ideal**
  - Subprograma:
    - Implementação de exatamente um algoritmo.
    - Cada algoritmo em um subprograma.
  - Programa principal:
    - Seqüência de vários algoritmos.
    - Delega execução de tarefas para os respectivos subprogramas (algoritmos).

Procedimentos e Funções

- Desvio do fluxo de controle



Procedimentos e Funções

- Funções
  - Função matemática
    - Definição: $f(x) = x^2 - 2x + 10$
    - Uso: $f(2) = 10$
  - Já utilizamos funções no C
    - tam=strlen(str);
    - y=abs(x);
    - etc...
  - Valor(es) de entrada → **Função** → Valor Saída

Procedimentos e Funções

- Declaração e definição:

```
tipo_saída nome_funcao(parametros){
 /*variáveis*/

  comandos;
  return resultado;
}
```

- Chamada:

```
nome_funcao(parametros)
```


Procedimentos e Funções

- Lista de parâmetros
  - Após o nome da função e deve estar entre parênteses
  - Cada parâmetro (ou argumento) precedido por seu tipo
  - Mais de um parâmetro: separar por vírgula
- Saída (Retorno)
  - Tipo do valor retornado especificado na definição
  - Não indicado, assume-se retorno do tipo inteiro
  - O comando **`return`** indica o valor retornado pela função
  - Este valor pode ser representado por uma constante, uma variável ou por expressões, respeitando o tipo
  - Uma função encerra assim que encontra o comando **`return`**
  - **`return`** sem valor faz a função finalizar

Procedimentos e Funções

- Criando novas funções

Se fosse um programa:

```
int main(){
  int fat, cont;
  int x;

  scanf("%d",&x);
  fat=1;
  for(cont=1; cont<=x; cont++){
    fat=fat * cont;
  printf("%d",fat);
}
```

Procedimentos e Funções

- Criando novas funções

Para uma função:

Tipo de Saída

Entrada

```
int fatorial(int x){
  int fat, cont;

  fat=1;
  for(cont=1; cont<=x; cont++){
    fat=fat * cont;
  return fat;
}
```

Valor de Saída

Procedimentos e Funções

- Números primos de 1 a N

```
int ehprimo(int numero){
  int pr, dv;

  pr = 1;
  for(dv = 2; dv < numero; dv++){
    if ((numero % dv) == 0){
       pr = 0;
    }
  }
  return pr;
}
```



```
int ehprimo(int numero){
  int pr, dv;

  pr = 1;
  for(dv = 2; dv < numero; dv++){
    if ((numero % dv) == 0){
       pr = 0;
    }
  }
  return pr;
}

int main() {
  int N, num;
  scanf("%d", &N );
  for(num=1; num<=N; num++){
    if( ehprimo( num )==1 ){
       printf("%d", num );
    }
  }
}
```

declaração e definição

chamada

Procedimentos e Funções

- Função main ()
  - É a função principal de um programa em linguagem C
  - Um programa começa a ser executado do início da função main() e termina quando a última linha é executada ou quando o comando return é chamado
  - O tipo padrão da função main() é int

```
void main() { ... }
int main() { ... }
int main(int argc, char *argv[]) { ... }
```

Procedimentos e Funções

- Procedimentos
  - Função que não retorna valor
    - Exemplo: exibição de um menu
  - Tipo de retorno **`void`**
  - Parâmetro(s) de Entrada → Procedimento

```
void exibe_letra (char letra) {
   printf ("%c", letra);
}
```


Procedimentos e Funções

- Protótipo de função
  - Trecho de código que permite declarar uma função antes de definí-la, já que na linguagem C toda a função precisa ser declarada antes de ser chamada por outras funções
  - Útil quando os programadores querem que a função main () esteja no início do arquivo fonte

```
tipo_saída nome_funcao(parametros);
```



```
int ehprimo(int numero);

int main() {
  int N, num;
  scanf("%d", &N );
  for(num=1; num<=N; num++){
    if( ehprimo( num )==1 ){
        printf("%d", num );
    }
  }
}
int ehprimo(int numero){
  int pr, dv;

  pr = 1;
  for(dv = 2; dv < numero; dv++){
    if ((numero % dv) == 0){
       pr = 0;
    }
  }
  return pr;
}
```

declaração

chamada

definição

Procedimentos e Funções

- Escopo de variáveis
  - Variáveis das funções só existem dentro da função
    - Variáveis locais
  - Parâmetros também são locais, a princípio
  - Variáveis/parâmetros das funções não retêm seus valores de uma chamada para outra
  - Variáveis do programa principal podem ser acessadas dentro das funções se passadas como parâmetros
    - Cuidado com uso direto de variáveis de escopo externo à função, como variáveis globais acessadas diretamente.

Procedimentos e Funções

- Passagem de Parâmetros

```
int calcula_media(int a, int b, int c){
 int media: integer;
 media = (a+b+c) / 3;
 return (media);
}
...
int main(){
  ...
  m = calcula_media(10,20,30);
}
```

- a,b,c assumem os valores 10, 20 e 30

Procedimentos e Funções

```
int calcula_media(int a, int b, int c){
  int media: integer;
  media = (a+b+c) / 3;
  return (media);
}
...
int main(){
  ...
  x=10; y=20; z=30;
  m = calcula_media(x,y,z);
}
```

- a,b,c assumem **cópia** dos valores de x, y e z
  - estão em lugares diferentes da memória
  - alterações em a,b e c não alteram x, y, z

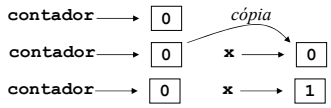
Procedimentos e Funções

- Passagem de parâmetros
  - Passagem por valor
    - Passagem do conteúdo da variável
    - Parâmetro do mesmo tipo da variável
    - Independência entre variáveis e parâmetros
    - Valores das variáveis são copiados para os parâmetros
    - Mudanças em parâmetros não afetam as variáveis
  - Passagem por referência
    - Passagem do endereço da variável
    - Parâmetro do tipo apontador
    - Dependência entre variáveis e parâmetros
    - Parâmetros apontam para o conteúdo das variáveis
    - Mudanças em parâmetros refletem nas variáveis

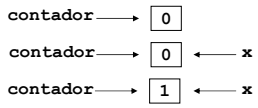
Procedimentos e Funções

- Passagem de parâmetros

```
void incVal(int x);
   x=x+1;
}
int main(){
  int contador=0;
  incVal(contador);
}
```




```
void incRef(int *x){
   *x=*x+1;
}
int main(){
  int contador=0;
  incRef(&contador);
}
```



Procedimentos e Funções

- Vetores como Parâmetros
  - Vetores também podem ser passados como parâmetros para funções, mas são sempre passados por referência
  - Implicação: ao alterar valores dos elementos de um vetor passado como parâmetro, as alterações são "percebidas" no escopo da função chamadora

```
void mostra_vetor (int v[LIM], int n);
```

equivale a

```
void mostra_vetor (int v[], int n);
```

equivale a

```
void mostra_vetor (int *v, int n);
```


Procedimentos e Funções

```
int quadrado(int x){
   return (x*x);
}
int main(){
  int num=10;
  num=quadrado(num);
}
```

```
void quadrado(int *x){
   return (*x)*(*x);
}
int main(){
  int num=10;
  quadrado(&num);
}
```



```
void MaiorMenor(int x, int y, int *maior, int *menor){
  if (x > y){
    maior=x;
    menor:=y;
  } else {

     maior:=y;
     menor:=x;
  }
}
int main(){
 int  num1,num2,maximo, minimo;

 scanf("%d %d",&num1,&num2);

 MaiorMenor(num1,num2,&maximo,&minimo);

 printf("Máximo: %d, Mínimo: %d", *maximo, *minimo);
}
```